Testability and redundancy techniques for improved yield and reliability of CMOS VLSI circuits

Djamel Abderrahmane Bensouiah

TESTABILITY AND REDUNDANCY TECHNIQUES FOR IMPROVED YIELD AND RELIABILITY OF CMOS VLSI CIRCUITS

by
Djamel Abderrahmane Bensouiah

This thesis is submitted to the University of Durham
in candidature for the degree of Doctor of Philosophy

School of Engineering and Computer Science
University of Durham
May 1992

Abstract

The research presented in this thesis is concerned with the design of fault-tolerant integrated circuits as a contribution to the design of fault-tolerant systems. The economical manufacture of very large area ICs will necessitate the incorporation of fault-tolerance features which are routinely employed in current high density dynamic random access memories. Furthermore, the growing use of ICs in safety-critical applications and/or hostile environments in addition to the prospect of single-chip systems will mandate the use of fault-tolerance for improved reliability.

A fault-tolerant IC must be able to detect and correct all possible faults that may affect its operation. The ability of a chip to detect its own faults is not only necessary for fault-tolerance, but it is also regarded as the ultimate solution to the problem of testing. Off-line periodic testing is selected for this research because it achieves better coverage of physical faults and it requires less extra hardware than on-line error detection techniques.

Tests for CMOS stuck-open faults are shown to detect all other faults. Simple test sequence generation procedures for the detection of all faults are derived. The test sequences generated by these procedures produce a trivial output, thereby, greatly simplifying the task of test response analysis. A further advantage of the proposed test generation procedures is that they do not require the enumeration of faults.

The implementation of built-in self-test is considered and it is shown that the hardware overhead is comparable to that associated with pseudo-random and pseudo-exhaustive techniques while achieving a much higher fault coverage through the use of the proposed test generation procedures. The consideration of the problem of testing the test circuitry led to the conclusion that complete test coverage may be achieved if separate chips cooperate in testing each other's untested parts. An alternative approach towards complete test coverage would be to design the test circuitry so that it is as distributed as possible and so that it is tested as it performs its function.

Fault correction relies on the provision of spare units and a means of reconfiguring the circuit so that the faulty units are discarded. This raises the question of *what is the optimum size of a unit*? A mathematical model, linking yield and reliability is therefore developed to answer such a question and also to study the effects of such parameters as the amount of redundancy, the size of the additional circuitry required for testing and reconfiguration, and the effect of periodic testing on reliability. The stringent requirements on the size of the reconfiguration logic is illustrated by the application of the model to a typical example. Another important result concerns the effect of periodic testing on reliability. It is shown that periodic off-line testing can achieve approximately the same level of reliability as on-line testing, even when the time between tests is many hundreds of hours.

DECLARATION

I hereby declare that the work reported in this thesis has not been submitted for any other degree.

ACKNOWLEDGEMENTS

I would like to thank my two supervisors, Dr. M. J. Morant and S. Johnson, for their help and support and for their careful reading of the text.

I also acknowledge the financial support of the ministry of higher education of Algeria.

List of Contents

CHAPTER 1

CHAPTER 2

CHAPTER 3

CHAPTER 4

Chapter 1
Introduction

The era of Very Large Scale Integration (VLSI) has been going on since the late 70's early 80's [1]. This is a long period by the standards of the semiconductor industry where rapid progress is the norm. A new era of higher levels of integration would have been expected to be in place by now. The two likely successors to the VLSI era are Ultra Large Scale Integration (ULSI) and Wafer Scale Integration (WSI).

In ULSI, higher levels of integration are achieved through the reduction in the minimum feature sizes*. This is the trend that has been followed since the first ICs were introduced. So far, it has allowed the successful production of 64-bit microprocessors and 64-Mbit DRAMs. However, there are limits to this trend, some of which are theoretical or fundamental, such as the disappearance of the transistor effect below a certain channel length [3, 6], while others are practical or economical, such as the colossal investment required for sub-micron technology.

In WSI, higher levels of integration are attainable by increasing the area occupied by chips, up to a whole wafer. The major limitation in this case being the daunting problem of defects in manufacture. Furthermore, the prospect of a perfect manufacturing process is extremely unlikely. However, this problem is of a practical nature, rather than

* Our interpretation of ULSI is the increase of the number of devices per chip through the reduction of device dimensions and without significant increases in chip area. However, because of its unpopularity [2] and the reluctance of many authors to using the term WSI, some publications use the term ULSI to refer to very large area ICs. Even Maly et. al. [2] use the term 'ULSI-WSI' to refer to such ICs. Most publications on scaling theory [3, 4, 5] use the term ULSI. With this interpretation, ULSI does not require defect-tolerance. On the other hand, WSI does not require, but neither does it exclude the reduction in device dimensions.

fundamental and there exists design techniques that address this problem successfully as illustrated by commercial defect-tolerant DRAMs. Hence, from the manufacturability point of view, defect-tolerance might make large area ICs economically feasible.

At the application level, other considerations, besides manufacturability, also have to be addressed. Chief among them is reliability. Even when the ICs supplied by a manufacturer contain no defects and no weaknesses, they are still susceptible to numerous phenomenon that might cause a chip to fail. Depending on the application, the consequences of chip failures can range from a slight inconvenience to very dramatic ones, especially with the increasing use of ICs in many different areas, often with hostile environments and/or safety critical applications. Furthermore, with higher levels of integration, a system that previously consisted of few tens or few hundreds chips will be designed around one or two chips. In this case, the reliability of the system will be dominated by the reliability of the chips themselves. Hence, failure-tolerance is also a very desirable feature for highly integrated circuits.

Even when current systems are integrated onto single chips or wafers, there is no doubt that new applications will emerge that will require a number of such highly integrated or wafer scale circuits, resulting in systems consisting of billions of devices. This raises the question of how to ensure the integrity of such systems given that chips consisting of only few hundred thousand devices are already causing headaches in testing and fault diagnosis tasks. Test requirements in a fault-tolerant integrated circuit are even more stringent since precise information concerning the presence of faults and their locations is necessary for performing corrective actions.

1.1 OBJECTIVES AND FOCUS OF THE RESEARCH

The main objective of the research presented in this thesis is the development of design techniques to improve the yield and reliability of ICs by making them tolerant to faults.

Preliminary investigations suggested a breakdown of this main objective into the following sub-objectives:

(i) **To attempt to derive a method of testing for all detectable faults:** Complete information on every type of fault is required to undertake corrective actions. This information can only be obtained by aiming at the detection of **all** faults, rather than just the high percentage of faults thought to be adequate by earlier workers.

(ii) **To apply the method developed in (i) to the implementation of built-in self-test:** As part of the requirement of detecting all faults, we must also find out how to test the BIST circuitry itself. Another point of focus is the investigation of the effect of partitioning on the hardware overhead associated with BIST.

(iii) **To identify a way of implementing fault-tolerance that is compatible with the methods of (i) and (ii).**

(iv) **To develop a method for assessing the effects of design decisions on both yield and reliability:** The design decisions of interest include the amount of redundancy, the level at which redundancy is introduced (i.e., the size of the replaceable units), and off-line periodic testing *vs* continuous checking, all of which were to be considered.

The yield and the reliability of ICs are continually enhanced through improvements in the manufacturing process, but this is beyond the control of most IC designers. For the same reason, design techniques that rely on the addition or modification of some processing steps are not considered.

The nature of the faults and the way in which they affect the operation of circuits depend on the technology used. CMOS, being the dominant technology for high density ICs, is selected for this research. A special emphasis is given to transistor stuck-open faults which are peculiar to this technology. In this thesis, a stuck-open fault is defined as any fault that would prevent conduction from the source to the drain of a transistor. This may happen as a result of breaks in the connections or contacts between the source/drain and other parts of the circuit or because of the impossibility of driving the gate to the required voltage level to turn the transistor on.

The special emphasis on CMOS stuck-open faults is due to the stated aim of this work which is to detect all faults rather than just a high percentage.

1.2 OUTLINE OF THE THESIS

The concept of fault-tolerance appeared with the first applications of digital systems. Hence, there is an abundant literature on the subject. Chapter Two is a review of this extensive subject with particular emphasis on those areas of relevance to the implementation of fault-tolerant ICs. The sources of faults, their effects on the operation of circuits, and the different abstraction levels used to model faults are discussed. The treatment of fault detection mechanisms highlights the differences between on-line and off-line approaches. In particular, it is found that the former cannot provide high fault-coverage and it also requires high hardware overheads, compared to the latter. In the discussion of fault-correction techniques, a clear distinction is made between defect-tolerance and failure-tolerance. Defect-tolerance deals with manufacturing defects and it has very limited capabilities for dealing with field failures, whereas failure tolerance techniques also have the potential of dealing with manufacturing defects.

A major aspect of off-line fault-detection is the generation of tests that establish the presence or absence of faults. Test generation is a complex problem, even for classical faults. In Chapter Three, it is shown that tests derived for stuck-open faults can also detect all other detectable faults. This result is used to derive simple test generation procedures for CMOS combinational circuits that produce test sequences for the detection of all stuck-open faults. An important attribute of the procedures presented in Chapter Three is that the generated test sequences should produce a trivial fault-free response. This represents a novel solution to the problem of test response analysis. The fact that these procedures generate test sequences, as opposed to test patterns, is a further advantage, since fault enumeration is no longer required.

The hardware implementation of Built-In Self-Test (BIST) is considered in Chapter Four. It is shown that, using the procedures of Chapter Three, BIST can be implemented with a hardware overhead comparable to that associated with pseudo-random and pseudo-exhaustive testing, while achieving a much higher fault coverage. The problem of the detection of faults in the test circuitry itself is also considered and it is proposed that the most effective solution is to design the test circuits in such a way that they are tested while performing their functions.

In Chapter Five, new yield and reliability models for fault-tolerant ICs are developed. The models are relatively simple and yet, they incorporate important effects such as the dependence of reliability on the manufacturing yield and the impact of off-line periodic testing on reliability. The illustration of the application of the models reveals the crucial importance of the size of the test and reconfiguration logic in determining whether there is any yield or reliability improvement. The reconfiguration logic is often ignored in many proposed fault-tolerance strategies, and even when it is not ignored, the assumptions concerning its size are shown to have an effect in determining the optimum amount of redundancy and the optimum size of the replaceable unit.

The main argument that can be used against off-line periodic testing is that the chip may produce errors between tests. It is shown in Chapter Five that this is nearly as likely as a complete failure of a continuously checked chip, provided that the time between tests is short enough: **up to many hundred hours.**

Chapter 2
Related Issues to Fault-tolerance: Introduction and Review

2.1 INTRODUCTION

Fault-tolerance for electronic circuits is an involved subject, comprising many aspects of design, manufacturing, and testing. Research on the subject began in the early 1950's. In these early days, the research was motivated mainly by the high unreliability of electronic systems. In the following years, and as electronic systems began appearing in many safety-critical applications, research into fault-tolerance became mandatory. This resulted in many techniques and methods for implementing fault-tolerant systems. This Chapter is a review of the previous research work and an introduction to the many aspects related to fault-tolerance and their relevance in implementing fault-tolerant integrated circuits.

A fault-tolerant system is a system that would continue to provide correct operation despite the presence of faults. The fault-tolerance attribute can be described in terms of two factors: detection of faults and correction of faults. This does not imply that the two factors are always physically separated, but only that they can be separated conceptually [7].

The presence of a fault in a circuit results in errors. An error is said to have occurred if the output of the circuit deviates from its specified behaviour. A fault is the physical phenomenon that produces errors. Faults may originate in any of the three phases of the life of an integrated circuit: design, manufacturing, and operation. The types of fault in each of these phases are discussed in Section 2.2. Section 2.3 discusses the effects of faults on circuit operation, the knowledge of which is necessary for deriving fault detection and fault correction techniques, discussed in Sections 2.4 and 2.5, respectively.

and 2.5, respectively. Faults introduced in the manufacturing stage have a direct effect on the yield, and an indirect effect on reliability, whereas faults occurring during normal operation affect the reliability only. Yield and reliability are the two main criteria for evaluating fault-tolerance strategies. They are discussed in Section 2.6. Section 2.7 summarises the Chapter.

2.2 SOURCES AND TYPES OF FAULTS

Faults may be introduced into an integrated circuit in the design phase, at the manufacturing stage, or during normal operation.

2.2.1 Design Mistakes

In the design phase of an IC, a top-down approach is usually adopted, as illustrated in Fig. 2.1. *Design mistakes* may be introduced at any of the transitions between levels. Note that the term used is *design mistakes* instead of the commonly used terms *design errors* or *design faults*. We do so firstly to avoid the circular situation (design fault→fault→error or design error→fault→error) and secondly to stress the fact that they should not occur. However, because of the increasingly complex ICs that VLSI has made possible, and the usual practice of having such chips designed by large teams of designers, design mistakes do occur even in the best design offices [8].

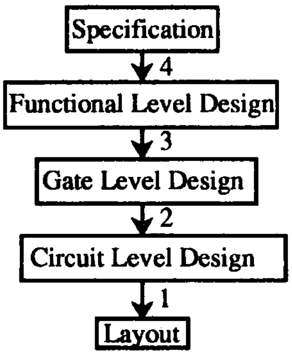


Figure 2.1 The top-down approach to IC design.

Design mistakes likely to be introduced in transitions 1 and 2 (incorrect transistor dimensions, layout prone to failure, circuit design mistakes, etc ...) can usually be avoided by the use of efficient CAD tools such as design rule and electrical rule checkers combined with circuit extractors. Mistakes in transitions 3 and 4 are much harder to avoid, although extensive and comprehensive simulation can deal with transition 3 to a certain extent. Transition 4 is the most crucial. It prompted the introduction of the concept of *formal verification* based on theorem proving techniques. However these methods are still in their infancy [9, 10]. One of the early results of these methods was in the design of the VIPER processor intended for use in high reliability applications [11, 12], the basic idea being in the use of a very simple processor architecture so that formal verification techniques could be employed.

Design ~~faults~~ mistakes are best handled by ensuring that they do not occur in the first place, which means providing good specification tools and a means of evaluating the proposed design before proceeding to realisation [13].

2.2.2 Manufacturing Defects

Faults occurring in the manufacturing stage are due to various defects introduced by a less than perfect processing. The manufacturing phase is considered to comprise all the stages that start with the raw silicon material and end with a packaged chip. Defects that affect the silicon die and those introduced in the assembly/packaging stage are discussed separately, because of their different nature and because of the fact that fault-tolerance strategies tend to deal with faults affecting the die only.

a) Die-Related Manufacturing Defects

Defects that affect the silicon die have been well studied in the literature because of their prime importance in yield forecasting and process improvement [2]. In most cases, processing defects can be classified as either global or local. Global defects affect large areas of the wafer or the IC, and they are usually easily detected at an early stage in the manufacturing process, well before the dicing of the wafer (through visual inspection in the case of area and line defects and by the continuous monitoring (test site observation) in the case of global process parameter variations). Their frequency should be low in a well controlled processing line.

Local defects, on the other hand, affect small areas of the wafer or die and are usually considered as spot defects. They are much more numerous than global defects and tend to be randomly distributed over the wafer.

Besides their global or local nature, processing defects can also be classified according to whether they are caused by photolithography errors or by process quality errors [14]. Basically, photolithographic errors result in a missing or extra feature on any of the layers of the IC. Table 2.1 is a list of such defects for a CMOS process.

Process quality errors result from instability in the process conditions or from variations in the physical properties of the chemicals used [2]. They are listed in Table 2.2. Most process quality errors will result in a change in the I-V (current-voltage) characteristic of the MOS device, which in turn may cause higher power consumption, lower

Table 2.1 Main Photolithographic Errors for a Standard CMOS Process.

Layer	Defect	Possible effects
Diffusion	Missing	Shorts to GND or VDD
	Extra	Short circuits
Implant	Missing or wrong	Variations in V_T
	Extra or wrong	Variations in V_T
Gate oxide	Missing	- Missing device - Short between gate and channel regions
	Extra	Parasitic device
Polysilicon	Missing	- Missing device - Floating gate - Open circuits
	Extra	- Parasitic device - Short circuits
Field oxide	Missing	- Shorts between poly and metal - Shorts between poly and substrate - Shorts between metal and substrate - Shorts between poly/metal and active regions.
	Extra	Open circuits
Metal	Extra	Short circuits
	Missing	Open circuits

noise margins, higher propagation delays, etc. These are usually referred to as performance faults. However, in extreme cases, process quality errors also result in breaks and short circuits.

b) Package and Assembly-Related Manufacturing Defects

The wire bond is considered to be one of the weakest areas of device packaging [15]. Poor bonding pressure can lead to bonds with a low fracture strength, while high bonding pressure may result in substrate cracking or partial detachment of the die from the carrier. If the loop formed between the semiconductor and the lead frame droops too much there is a susceptibility to shorts between adjacent bond wires. On the other hand, if the loop is too tight, the tension created may lead to fractures.

In plastic encapsulated devices, the plastic compounds used may force the bond wires against each other resulting in shorts. In hermetic packages, debris from the materials used in packaging and assembly may be introduced, later becoming loose and causing short circuits. Poor bonding may also enhance the formation of intermetallic compounds through gold-aluminium interdiffusion [15].

2.2.3 Operational Failures

During the operational phase, faults occur because of physical failures. Chip failure implies that the chip was initially good and then, for some reason, it became faulty. Hence, those faulty chips that escape the manufacturer's screening process and then 'fail' in operation are not considered to have failed since they were never good. Chip failures may be due either to *weaknesses* introduced during the manufacturing phase or to ageing. In the same way as for manufacturing defects, failures are classified into those affecting the die and those related to the packaging/assembly process.

a) Die-Related Operational Failures

Many of the failures occurring in normal operation can be related to manufacturing defects. A processing defect may not be severe enough to cause a chip to fail the manufacturer's tests, but the presence of the defect constitutes a weakness that can be aggravated in operation because of temperature, voltage or mechanical stress.

There are many failure mechanisms that affect the metallisation. The most important is electromigration. It is caused by the continuous impact of electrons on the aluminium grains in high current density situations, resulting in a movement of aluminium atoms in the direction of electron flow, giving rise to voids [15]. Contact regions are also subject to migration of aluminium atoms into silicon or Si atoms into Al which results

Table 2.2 CMOS Process Quality Errors.

Affected Layer	Defect	Possible Effects
Substrate	Crystal imperfections Doping levels	Excessive leakage currents Variations in V_T
Implant	Ion implant dose Implant energy	Variations in V_T Resistance variation of implanted regions
Gate oxide	Bad quality Pinholes Thickness variations Contamination	Variations in V_T Short circuits
Polysilicon	Bad quality Variation in thickness (over or under etching)	Variations in channel length Variation in RC time constants
Field oxide	Contamination Thickness variations Pinholes	Short circuits Variations in RC time constants (capacitance of conductors to substrate)
Metal	Grain size Microcracks Metal liftoff	Short circuits Open circuits
Contact windows	High resistance Spiking Bad metal coverage	Short circuits Open circuits RC time constant variations
Photoresist	Contamination Over or under exposure Thickness variations	Over or under etching of poly, metal or field oxide

in open circuits or spiking of Al into Si. Corrosion is also a major cause of failure of the metallisation, especially in plastic encapsulated devices where the porosity of the plastic material is responsible for the diffusion of moisture through the package and down to the metallisation [15]. Due to their lower power, CMOS devices are more susceptible to this mechanism than bipolar where the high power dissipation at the semiconductor surface reduces the moisture content. Microcracks are yet another type of failure affecting the metallisation layer. They are due to badly processed oxide steps where the metallisation layer is spread thinly and weakened, enhancing its susceptibility to electromigration.

The oxide is also prone to many types of failures. The gate oxide is susceptible to breakdown under the influence of high electric fields generated by electrical overstress or static discharge [15]. Ion contamination at the manufacturing stage, introduced by human contact, processing or packaging materials, causes the ions to accumulate at the oxide-silicon interface of the gate region of MOS devices under the influence of the electric field. Inversion layers may also be created outside the active regions of transistors because of the movement of electrical charges through the oxide, resulting in threshold voltage shifts, shorts between adjacent active regions, or the formation of parasitic transistors. Another important effect occurs when the electrons in the channel region acquire a high enough energy (hot electrons) to cross the Si-SiO_2 interface, getting trapped in the gate oxide and resulting in a shift of the threshold voltage [15].

Radiation constitutes another source of failures in the field. In addition to radiation produced by trace impurities of radioactive elements in the packaging materials, semiconductor devices are also prone to external radiation sources [16, 17]. Radiation may result in threshold voltage shifts, reduction in transconductance, 'soft' errors in programmed devices, or the activation of parasitic elements. Other external factors that may cause failures include heat, mechanical stress, and electromagnetic interferences [15, 18].

b) Package-Related Operational Failures

The commonly observed failures affecting the package [15] are:-

(i) Bonding failures: In most cases, failures are due to bonds lifting, giving complete open-circuits. This can occur due to formation of intermetallic compounds or insufficient bonding pressure in manufacture. High resistance bonds can also be formed by intermetallics.

(ii) Die Attachment: The die, or part of it, may break loose under excessive mechanical/electrical stress. The die attachment is the first link in dissipating the heat generated in the silicon die. If a void is present, then a local hot spot is formed, leading to further failure.

(iii) Particulate contamination: Any loose particles introduced during the assembly process may not be detected in screening, or alternatively, particles may become loose after the assembly process as a result of breakages within the device itself. Conducting particles found in failed devices include ends of bond wires, particles of silicon from scribing, gold flakes, and solder balls from cover sealing. Basically, they may all cause short circuits. These have been reported as the primary cause of 'one-off' failures, prompting the introduction of X-ray inspection as well as Particle Impact Noise Detection (PIND) tests for high reliability devices [18].

2.2.4 The Effects of Scaling

The trend toward greater chip complexity has been accomplished by scaling down feature sizes and increasing die sizes. As device dimensions are reduced, most of the defects and failures listed in the previous sections become more prominent for several reasons. The first is the increasing role played by smaller defects which are much more numerous than larger ones [2]. These smaller defects would not have been fatal in non-scaled devices, but they will cause serious yield and reliability problems when they become commensurate with the reduced device dimensions [19].

Another reason for increased problems with small devices is that the reduction in device dimensions has so far been carried out following *constant voltage scaling* which is worse for chip power density, current density, and electric field strengths than *constant field scaling*. High power dissipation leads to higher chip temperatures and thus higher failure rates for those mechanisms with positive activation energies [19]. Higher current densities make metallisation interconnects more susceptible to electromigration, while higher electric field strengths have an impact on hot-electron injection and dielectric breakdown. On the other hand, if the power supply is scaled down with the dimensions, then signal strength and noise margins will suffer. However, capacitive coupling of signal lines will be worse in scaled down devices in both constant field and constant voltage scaling [19].

From a packaging point of view, larger dies are more susceptible to cracking of the passivation layer or the die itself. Higher lead counts can also be detrimental due to the increased probability of a bad or weak bond [19].

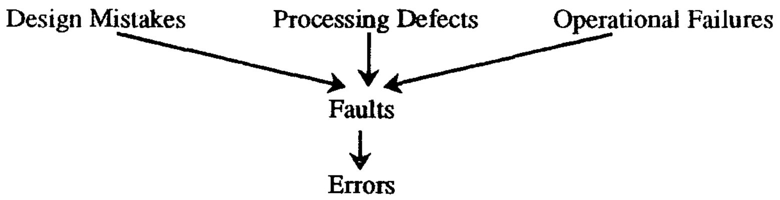


Figure 2.2 Cause-effect relationships.

2.3 FAULT MODELLING: THE EFFECTS OF FAULTS

It was stated previously that errors are caused by faults which, in tum, may be caused by design mistakes, processing defects, or operational failures, as illustrated in Fig. 2.2.

The effects of processing defects and operational failures, listed in the previous section, ranged from short and open circuits to threshold voltage variations and excessive leakage currents. If we want to demonstrate the presence or absence of a fault in a circuit we need to be able to test it so as to produce an error at the output of the circuit, because it is impractical, and most often impossible, to check every interconnect for short or open circuits and every MOS device for the correct threshold voltage. An error is simply the appearance of an output which deviates from the expected one.

The problem is then how to make a fault at one of the thousands of MOS devices or interconnects inside a VLSI chip, such as a threshold voltage variation for example, produce an error at the output of the circuit. Clearly, a general solution to this problem would be very difficult. The problem is usually simplified by making some assumptions about the behaviour of circuits under fault conditions. The process of constructing these assumptions about faults is called *fault modelling*, and the resulting fault assumptions are called a *fault model.*

Hayes [20] defines fault modelling as 'the systematic and precise representation of physical faults (defects and failures) in a form suitable for simulation and test generation. This representation usually involves the definition of abstract or logical faults that produce approximately the same behaviour as the actual physical fault'.

Faults and fault effects may be considered at many different levels: technological, structural, or device levels, as in [21], and circuit, switch, gate or functional levels, as in [20, 22], in order of increasing levels of abstraction. The effects of the failures and defects listed in Section 2.2 belong to the circuit level. At this level, faults are characterised by changes in voltages, currents, or propagation delays which do not lead to efficient fault modelling, so that higher levels of abstraction are usually required. In the following we will discuss the different abstraction levels commonly adopted to deal with faults.

2.3.1 Logic Level Fault Modelling

Logic level modelling is the most widely used abstraction level in test generation and (fault) simulation of digital circuits. At this level, circuits are represented by an interconnection of logic gates, and all defects and failures are assumed to result in a single node being permanently stuck-at logic one or logic zero [23]. The model was originally proposed as a means by which logic circuits could be tested without the need to apply every possible input [23].

The popularity of the single stuck-at fault model is illustrated by its widespread use, virtually as an industry standard, even for present day VLSI chip testing. There are three main reasons for this popularity. Firstly, the model proved simple and cost-effective for testing logic circuits made up of discrete components mounted on a PCB (the technology for which it was initially proposed). Secondly, even after the introduction of SSI and MSI TTL devices, it still proved adequate, since many of the typical faults in TTL resulted in stuck-at lines. Finally, and most importantly, almost all automatic tools for test related activities (test pattern generation, fault simulation, testability measures) are still based on this model.

The simplicity brought about by the stuck-at fault model stems primarily from it being a gate level model. This can be illustrated by considering the circuit level representation of a single logic gate. The circuit model has many components and interconnects, all subject to a variety of faults. The logic model of the gate has only a single component and few interconnects that are subject to a restricted set of faults [20]. In fact, in the stuck-at fault model, only the inputs and outputs of the gate are considered as possible fault sites; the gate itself is assumed to be fault-free.

An important question about the usefulness of such a simple model is whether physical faults at the circuit level can really be modelled by permanent stuck-at 0's and 1's at the input or output of logic gates. Several studies have attempted to answer this question with results ranging from those still accepting the validity of the model [22, 24, 25] to those doubting its usefulness [26, 27, 28, 29]. However, all such studies agree on the fact that there are certainly many physical faults that cannot be modelled as stuck-at faults.

In studies that found the model acceptable, the main argument is usually that a test derived for single stuck-at faults would detect many of the circuit level faults that cannot be modelled as line stuck-at. Studies that found the model inaccurate propose its extension to include other faults that are usually observed in circuits, such as shorts between lines, multiple lines stuck-at, and transistor stuck-on and stuck-open faults.

In the extension of the stuck-at fault model to include shorts, or bridging faults, it is assumed that the shorted lines take the same logical value which is the AND or the OR of the values on the lines involved. With this extension, it is clear that some bridging faults cannot be detected: an AND-type short between the inputs to an AND gate, or an OR-type short between the inputs to an OR gate, are examples. Necessary and sufficient conditions for the undetectability of bridging faults are given in [30], together with the cases where an undetectable bridging fault invalidates a test set derived for single stuck-at faults.

A short between nodes a and b where the value of a is a function of b, or b is a function of a, is called a feedback bridging fault. Shorts that create feedback loops represent an additional difficulty because they may introduce asynchronous behaviour and they may cause oscillations even in a combinational network [31].

A major problem in test generation is due to the very large number of possible bridging faults, compared to the number of single stuck-at faults. In a circuit containing n lines, there are $2n$ possible single stuck-at faults, compared with $\binom{n}{r}$ possible bridging faults involving r lines. This number can be reduced by looking at the actual layout and considering only bridging faults between nodes that are physically close. However, in a typical VLSI circuit, the number of bridging faults is so large that it is prohibitively expensive to consider each fault individually and derive a test for it. Historically, bridging faults have often been detected by aiming at a high level coverage of single stuck-at faults (in the high 90%) [32]. Mei [24] has shown that, indeed, many bridging faults are detected by single stuck-at test sets, and the remaining faults can be detected by a **reordering** of test vectors or a modification of the single stuck-at test generation algorithm.

In a recent study [33], simulation runs were performed on sample circuits to assess the effectiveness of stuck-at test sets in detecting bridging faults. It was found that stuck-at test sets provide over 98% coverage of bridging faults which is judged (by the authors of the paper) to be inadequate for today's VLSI chips. Their proposed technique to increase the bridging fault coverage is, again, to reorder the stuck-at test vectors so that **all nodes change value as often as possible**.

Multiple stuck-at faults present the same difficulties as bridging faults: their very large number and the fact that they might invalidate single stuck-at test sets. Hence, they have usually been dealt with in a similar manner, i.e., no attempt is made to derive tests for individual multiple stuck-at faults, instead, there have been attempts to show that most multiple stuck-at faults are detected by single stuck-at test sets.

There are $3^n - 1$ possible multiple stuck-at faults in a circuit consisting of n nodes. Even with the use of techniques such as fault equivalence and fault collapsing [34, 35] to reduce the number of faults that need to be considered, it is not possible to bring the total number of faults to manageable limits, especially for large circuits.

For certain classes of circuits, such as two-level combinational networks, and 'restricted' internal fanout-free circuits [36], it has been shown that any single stuck-at test set will detect all multiple faults. However, most practical circuits do not belong to these classes and for general combinational circuits, the undetectability of multiple stuck-at faults by single stuck-at test sets is due to fault masking. A detectable fault f1 is said to be masked by another fault f2, with respect to a test set T, if T detects f1 but fails to detect the simultaneous occurrence of f1 and f2. The task of determining all the conditions under which fault masking occurs is as complex as the task of considering all multiple faults for test generation [37, 38]. In addition, simulating all multiple faults to determine which of them are covered is not practical, although it has been partially carried out for a small circuit, the 74LS181 4-bit ALU [39]. In this study, 16 different complete single stuck-at test sets, generated by different methods, were simulated against all possible double stuck-at faults. Only four test sets (the longest ones) detected all double faults. For the remaining test sets, there were between 1 and 30 undetected double stuck-at faults.

The approach usually adopted in estimating the effectiveness of single stuck-at test sets in detecting multiple faults is to theoretically evaluate some lower bounds on the multiple fault coverage of these test sets [37, 38]. Impressive multiple fault coverage figures can be obtained by such theoretical studies. For example, Jacob et.al [37] show that, in a circuit having three or more outputs, at least 99.67% of all multiple faults are guaranteed to be detected by any single stuck-at test set, irrespective of the size of the circuit. In a small circuit, consisting of 100 nodes, this leaves 1.4×10^{45} undetected multiple faults (the total number of multiple faults is 5.15×10^{47})!

The great majority of multiple faults are of relatively high multiplicities, and these faults are easily detected because of their drastic effect on circuit operation. However, low multiplicity faults, say up to 5 or 6, are more likely to occur in practice and they are more difficult to detect. In [37], it was found that for a circuit of 10000 lines and 5 outputs, the lower bound on the overall multiple fault coverage of any single stuck-at test set was 99.9929%. Using the same calculations employed in [37], we find that the lower bound on the coverage of double faults is only around 50%.

Physical faults that cause a transistor to be permanently on or permanently off, independently of the gate voltage value, cannot be readily modelled at the gate level [40]. An obvious reason for this is that the gate level representation hides the internal structure of the gate. Another reason is that transistor stuck-on faults typically result in certain nodes assuming intermediate voltage levels, while transistor stuck-open faults give rise to high impedance states. Attempts to adapt the gate level model to these types of fault are reported in [41, 42]. The approach is to replace the pull-down and the pull-up networks of each gate by separate networks of logic gates controlling a flip-flop that simulates the memory effect due to stuck-open faults. Besides the complexity of such workaround circuits, they contain nodes that have no correspondence in the real circuits.

The above discussion has shown the limitations of gate level modelling. With the advent of VLSI, all the three reasons for the popularity of the single stuck-at fault model, mentioned earlier, are becoming irrelevant. First, the simplicity brought about by the model is more than offset by the exponential growth in circuit densities, and the resulting large numbers of faults that need to be considered. Secondly, there is definite evidence that many physical faults can result in faulty behaviour that cannot be modelled by stuck-at nodes in a gate-level schematic [26, 27, 28, 29]. Thirdly, the tools used for automatic test pattern generation, fault simulation, and testability analysis are facing increasing problems for chips of VLSI complexities because of the prohibitive computational costs involved.

From the above discussion, we can formulate the following two main problems in modelling circuits at the gate-level:

- For VLSI circuits, the number of primitive elements (gates) becomes very large, leading to a prohibitive number of faults.
- The fault model at the gate level does not cover all realistic physical faults.

Unfortunately, efforts have been made to solve the above problems separately and this has led to the situation where a solution to one problem exacerbates the other.

2.3.2 Functional Level Fault Modelling

Functional-level modelling is usually adopted to cope with the complexity of VLSI circuits. At the functional level, digital circuits are represented by an interconnection of functional units. Each unit is described in terms of the function it implements rather than its internal structure. As an example, a decoder circuit can be described functionally as having n inputs and 2^n outputs, and in normal operation, only one output line is activated, corresponding to the input address.

The most general functional fault model is based on the appearance of arbitrary changes in the truth table of a combinational circuit or in the state table of a sequential circuit [20]. In the presence of a fault a combinational circuit with m inputs can therefore be changed to any of the $2^{2^{n}}-1$ other m–input combinational circuits. The disadvantage of such an approach to fault modelling is that test generation based on this model essentially requires exhaustive testing. A simpler alternative approach that has been suggested consists of considering stuck-at faults on the inputs and outputs of functional units themselves ('pin faults'). However, this was shown to be inadequate even for simple functional units [43].

A more useful approach is to look at a particular function to determine whether any functional fault models are suggested by the function itself. Continuing with the example of the decoder circuit, this approach would lead to the following set of functional faults [22]:

- Instead of the correct line, an incorrect line is activated.
- In addition to the correct line, an incorrect line is also activated.
- No line is activated.

For a general function, deriving a list of functional faults, as above, would normally require an extensive analysis of the effects of physical faults at lower levels of abstraction. This analysis needs to be carried out only at the first time a particular function is used. However, because there is no limit to the number of possible functional units, such analyses have been done only for the most commonly used building blocks in VLSI chips (decoder, multiplexer, PLA, RAM, etc.) [29, 44, 45].

It should be noted that 'functional testing' was in common use before the introduction of the stuck-at fault model which caused a shift towards 'structural testing'. The renewed interest in functional testing, by test engineers in the first instance, has been prompted partly by the unavailability of structural information for many off-the-shelf components (microprocessors and their support circuitry). But perhaps the main reason for the current interest in functional modelling for test generation and fault simulation is that it is hoped that it will bring the same reduction in computational complexity brought about by the replacement of gate level simulation by functional simulation. However, this does not yet appear to be an immediate prospect.

2.3.3 Switch Level Fault Modelling

Switch level modelling was developed in the 1970's primarily to meet the requirements of the then emerging MOS technologies. It was made possible by the fact that an MOS transistor has a switching behaviour that is closer to that of an ideal switch, unlike a bipolar transistor.

At the switch level, logic circuits are modelled as a set of nodes connected by transistor switches. MOS transistors are modelled as resistive switches, with the resistance being a function of the dimensions of the transistor. Nodes have capacitances associated with them. In addition to the two logic levels 0 and 1, nodes also have strength levels, which take into account the impedance of the node, the impedances of the other nodes that are connected to it through MOS transistors, and the conductances of these connecting transistors. For a more formal description of these concepts see [46, 47].

Switch level modelling is very similar to circuit level modelling as in SPICE for example. However, in switch level modelling, transistor resistances, node values, strengths and capacitances are confined to a small set of possible discrete values, unlike the analog circuit level modelling. It is more accurate than gate level modelling, for both the behaviour and the structure of MOS circuits, while avoiding the high computational costs associated with analog electrical models.

The fact that switch level models operate on a direct representation of transistor structure makes it possible to model important phenomenon in MOS digital circuits, such as bidirectionality of signal flow and dynamic charge storage. Furthermore, MOS logic circuits often contain structures, such as pass transistors and dynamic latches, that have no correspondence in logic gates. For instance, a CMOS complex gate, implementing a non-primitive logic function, can be represented by a network of primitive gates at the logic level, as shown in Fig. 2.3. However, it can be seen that some of the nodes on the gate level representation do not exist in the real circuit, while some of the nodes in the real circuit have no correspondence in the gate level representation.

The most appealing feature of switch level modelling is the ease of modelling faults without the need for the complex work-around circuits that have to be added for gate level modelling, especially for CMOS faults. For instance, stuck-open and stuck-on faults can be modelled by setting an MOS transistor resistance to a very large (infinite) value or to zero. Alternatively, a stuck-open fault can be modelled by inserting a series transistor with the gate tied to a fixed value. Similarly, stuck-on faults can be modelled by adding parallel transistors. Line breaks and bridging faults can be modelled by just breaking or adding connections.

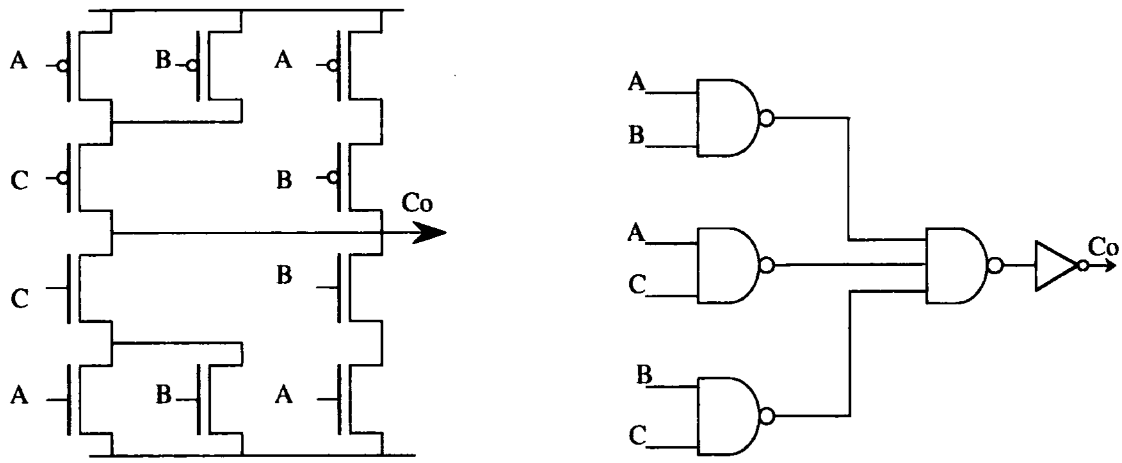


Figure 2.3 A CMOS complex gate and its gate level representation.

Rajsuman et.al. [48, 49] found that switch level modelling of bridging faults in NMOS and CMOS complex gates can be in good agreement with the equivalent circuit level modelling. The only difference between the two was that when the switch level analysis results in an indeterminate value, the circuit level analysis can often predict a definite voltage. However, this voltage is so dependent on the circuit parameters used that it is best taken as indeterminate.

Hayes [46] claims that the adoption of switch level modelling of fault-free circuits results in only a moderate increase in computational costs, whereas Bryant [47] claims that it can be implemented without incurring any increase at all, compared to gate level modelling. However, when it comes to fault modelling, the switch level is clearly more computationally intensive, since the number of primitive elements is much larger, and the set of possible faults per primitive element is also larger than in gate level models.

2.3.4 Layout Level Fault Modelling

Analyzing the effects of manufacturing faults at the layout level is not as widespread as the higher modelling levels mentioned in the previous sections. In fact, only a single research group advocates its use [50, 51, 52], and it is included here only because of the somewhat controversial results reported by this group.

In their first paper [50], the authors describe the method, called 'inductive fault analysis', by which they relate point defects to faulty behaviour at circuit level and then logic level. The method consists of injecting spot defects of random sizes, at random locations on the layout, and then extracting the resulting electrical circuit and analyzing it to determine any deviations from the fault-free circuit. The sizes and locations of the defects are randomly generated so that they follow probability distribution functions usually observed on processing lines. The observed faults are then ranked according to their frequency of occurrence, hence the label 'realistic'.

The results of the application of inductive fault analysis to n-MOS circuits are reported in [51]. It is indicated that only 10% of the simulated defects actually produce significantly faulty behaviour at the circuit level. Among these, only 28% of the faults can readily be modelled by line stuck-at faults at the gate level. It is also pointed out that most of the transistor stuck faults can be 'implicitly' modelled by the classical line stuck-at fault model, which is not surprising for an n-MOS circuit. This makes the classical fault model able to account for 64% of all possible faults. Although the simulation involved an n-MOS circuit, it is pointed out that less than 3% of the faults were transistor stuck-off faults.

In their latest paper [52], the authors apply their method to CMOS circuits. It is found that, for the simulated circuits, the classical single stuck-at fault model accounts for less than half of the faults. Furthermore, it is claimed that the adoption of switch level modelling provides little improvement. It is claimed that only 1% of the faults result in the sequential behaviour due to transistors being stuck-off. It should be noted that 70% of the transistors in the example circuits used by the authors are either pass transistors or parts of an inverter. Although the authors recognise that this might be the reason for the very low percentage of stuck-open faults resulting in sequential behaviour, they still maintain that this type of fault does not deserve the considerable research effort devoted to it in recent years. Shen [53] considers such type of faults as being arbitrary, and the research devoted to it as an academic exercise.

It should be noted that a large proportion of the simulated defects reported in the above papers resulted in breaks. The authors do not seem to realise that a break is a potential source of CMOS stuck-open faults. For example, in the NAND gate, shown in Fig. 2.4(a), four out of the seven possible breaks shown result in stuck-open faults, whereas in the EX-OR gate at (b) all possible breaks result in a stuck-open faults.

Besides the controversial results, inductive fault analysis is believed to go against the philosophy of integrating design and test activities. If a chip designer has to wait until the layout phase is completed to find out that the chip is difficult to test, then it would be very expensive to redesign the chip so that it is easily testable. Furthermore, the suggested approach of submitting a ranked fault list for test pattern generation is attractive only to the person generating the tests: it allows him/her to claim impressive fault coverage, since tests are generated for the most likely faults and these faults are given a higher weight in computing the fault coverage. From a user's point of view, the

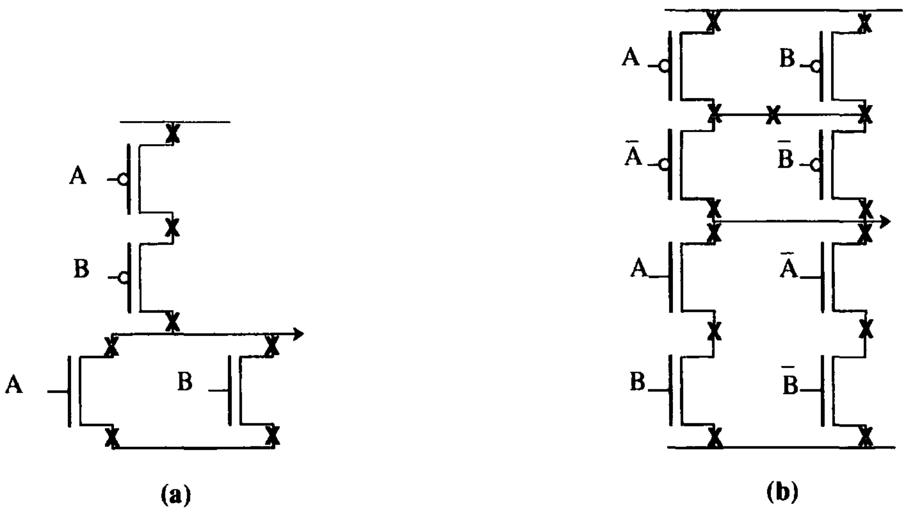


Figure 2.4 Possible sites of CMOS stuck-open faults.

presence of an undetectable fault is unacceptable, even if the fault has a low probability of occurrence.

2.4 FAULT DETECTION

Fault detection is the cornerstone of any fault-tolerance strategy. Several techniques for fault detection have been developed over the years. They can be broadly classified as either off-line or on-line detection techniques. In off-line fault detection, the normal operation of the circuit is interrupted and a stimuli are applied to the circuit to produce errors at the outputs if a fault is present. In on-line techniques, error detection is performed during normal operation by monitoring the output of the circuit or by some form of repetition of the function of the circuit. Duplication is the standard for comparison with all other fault detection techniques, since it is the simplest and it covers a wide variety of fault types.

There is a fundamental difference between off-line fault detection and on-line error detection that is often overlooked. In off-line techniques, stimuli are applied to the circuit so that, if a fault is present, an erroneous output is produced. Thus, they effectively detect the presence of faults by producing errors at the outputs. On-line error detection, on the other hand, will detect a fault only if and when the fault produces an error. For example, single error detecting schemes will, eventually, detect only those faults that result in a single error at the output. This does not imply that all single faults are eventually detected, since a single fault can cause more than one output to be erroneous. When deriving a test for a fault which causes more than one output to be erroneous, the fault is considered to be easily detectable, whereas in on-line detection techniques, this situation is undesirable.

The various fault detection techniques are discussed in the next subsections.

2.4.1 On-Line Error Detection Using Coding Techniques

Coding techniques, as applied to digital circuits, are based on the idea that if, for an m-output circuit, only k out of the 2^m possible output combinations may occur in a fault-free circuit , then the occurrence of any of the remaining $2^m - k$ output combinations is an indication of the presence of a fault. The outputs of the circuit under consideration (CUC) are continuously monitored by another circuit, called a checker, for the occurrence of invalid outputs. The combination of a CUC and its checker then becomes a self-checking circuit (see Fig. 2.5a). Formally, the k valid output combinations are called 'code words' and the remaining $2^m - k$ output combinations are called 'non-code words'. For example, in a decoder circuit only one output line should be activated at a time. This fact can be used to design a simple circuit that will check for correct operation as illustrated in Fig. 2.5(b) where the outputs are assumed to be active high. Here, all the output code words will have exactly a single 1 and seven 0s. Note that if a fault in the decoder causes an incorrect line to be activated, then the checker will not detect it. This is because the fault has changed one code word into another, incorrect, code word. In addition, a stuck-at 1 fault on the output of the checker is undetectable. Solutions to the above problems, and some others, will be discussed in the following.

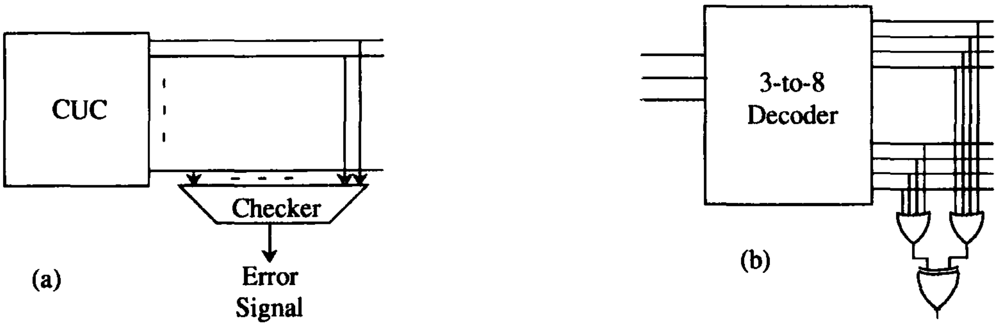


Figure 2.5 A self-checking circuit (a) and a self-checking decoder (b).

If a circuit function allows all 2^m possible output combinations, which is the case in most practical circuits, then there are two approaches to the use of coding techniques for error detection. In the first approach, the number of outputs is increased to some value

m_1 so that there would be $2^{m_1 - m}$ output combinations that are non-code words. In addition, if the faults on the inputs to the circuit are to be detected, then it is necessary to increase the number of inputs so that a proportion of the input combinations would be invalid, or non-code words. Although the number of inputs and outputs can be augmented in an arbitrary fashion, and the circuit redesigned accordingly, in practice, this augmentation is carried out using error detecting codes [54, 55]. Once a code has been selected, the circuit must then be redesigned in such a way that:

1) any fault in the circuit must cause the output to be either a non-code word or the correct code word, but never an incorrect code word, and

2) for each fault in the circuit, there must exist at least one input code word that would produce a non-code word at the output, if the fault is present.

A circuit that has property 1) is said to be *fault-secure.* Fault-secureness ensures that the circuit output is either correct or, if it is incorrect, it is a non-code word and hence it is detected by the checker. A circuit having property 2) is said to be *self-testing.* This property ensures that every fault will eventually produce a non-code word, and therefore, will be detected by the checker. A circuit that is both fault-secure and self-testing is called a *totally self-checking*, TSC, circuit [54].

To address the problems of faults in the checker itself, it is necessary to make the checker totally self-checking as well. However, since the inputs to the checker, i.e., the outputs of the circuit, are already encoded, the design of a totally self-checking checker is usually simpler than the design of a TSC circuit. Figure 2.6 shows the resulting general form of a totally self-checking network when both the circuit under consideration and the checker are totally self-checking.

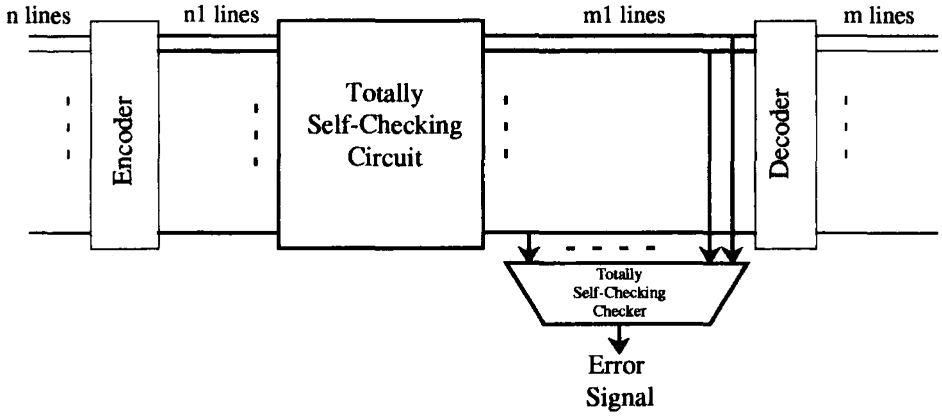


Figure 2.6 The general form of a totally self-checking network.

It should be noted that if the encoder and the decoder are considered to be part of the network, then it is no longer a TSC network. Faults in the encoder may result in an incorrect code word being applied to the circuit, and so that some faults will not be detected. The detection of faults within the encoder requires that the encoder itself be totally self-checking, i.e, a second encoder is needed before the first one. Similarly, the detection of faults in the second encoder requires a third encoder, and so on, *ad finitum.* The same problem exists on the output side: if faults in the decoder circuit are to be detected, then the outputs of the decoder must form a code that is decoded by another decoder. These problems can be lessened, but not eliminated, if the inputs to the CUC are already encoded, and if the outputs do not need to be decoded before being used by subsequent units. This requires that the same coding scheme be used uniformly throughout the system.

There is no systematic method for designing an arbitrary circuit so that it is totally self-checking. Most, if not all, publications on the subject concentrate on the design of totally self-checking checkers for different codes, and the design approach usually consists of designing a checker and then verifying that it is TSC.

The second approach in using coding techniques for concurrent error detection does not require the redesign of the CUC so that it is TSC, as illustrated in Fig. 2.7. It is therefore somewhat simpler than the first approach. Only separable codes can be used in such a manner and most applications of coding techniques in digital circuits use this approach. The only section that is TSC is the comparator even though such a structure is able to detect all single errors. Note that this is different from single fault detection. A single fault in the CUC may cause more than one erroneous output that may not be detected.

The output code generator in this approach is simply an encoder. It is clear that the output code prediction requires some knowledge of the function implemented by the CUC. In the simple case of an adder circuit using a residue code, for example, the output code prediction involves generating the check bits of the input operands and summing these check bits. In more general cases, the output code prediction circuitry can be as complex as the functional circuit.

The absence of any uniform error detecting code that can be used in any digital circuit has resulted in the situation where many of the practical implementations of on-line error detection employ a variety of techniques. For example, data storage and transmission are protected by parity coding, arithmetic operations use residue coding or duplication, and random logic is usually duplicated.

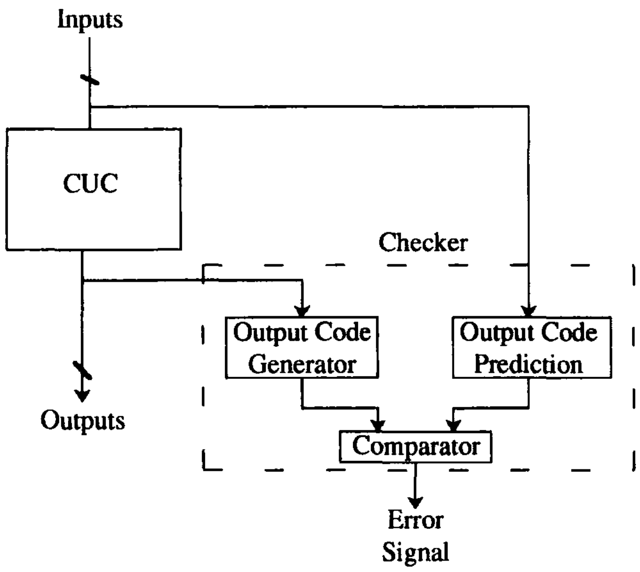


Figure 2.7 Second approach to concurrent error detection.

An attempt has been made by Sayers et. al. [56, 57] to show that residue codes can be used as a unified error detection scheme. However, as early as 1970 [58], it was shown that the application of residue codes to adder circuits, for which it is most suited, is more costly, in terms of hardware overhead, than the simpler parity coding, unless the addition operands are already provided with the check bits. In a more recent paper [59], Elliot and Sayers prefer to use the simpler parity codes in a design which is virtually identical to the one presented in [57].

Crouzet and Landrault [60] evaluated the sizes of the encoders and the checkers for different codes, and found that the least expensive was parity coding followed by simple duplication, and the most expensive was residue coding.

In a recent study, Yen et. al. [61] describe the application of coding techniques to a microprogram control unit, where parity coding is used for storage elements, and Berger codes are used for PLAs, with duplication of some parts of the chip. This is typical of most on-line error detection applications. It should be noted that in circuits containing large ROMs or RAMs, impressive results have been reported for area overhead of on-line error detection schemes, since the addition of a single check bit will provide single error detection in these circuits. In some cases, the chip is deliberately designed using ROMs and PLAs, instead of random logic, so that these coding techniques can be efficiently implemented [62].

The discussion in this section is meant to highlight the following characteristics of coding techniques:

- The hardware overhead is high.
- The coverage of physical faults is low.
- There is no coding scheme that can be used effectively for different functions.
- The problems associated with the detection of faults in the added circuitry are much worse than in the case of off-line fault detection techniques.

2.4.2 On-Line Error Detection Using Time Redundancy

Time redundancy techniques attempt to reduce the extra hardware at the expense of using extra time. The basic concept is that the same hardware is used in different ways at different times so that comparison of the outputs will allow errors to be detected.

The simplest approaches to time redundancy techniques were developed for iterative circuits where they require a doubling of the execution time. Recomputing with shifted operands (RESO) [63, 64], achieves error detection by performing the same operation twice. In the first phase, the inputs are applied to the iterative circuit and the outputs are stored in a register for later comparison. In the second phase, the inputs are first shifted to the left by k positions, say, and then applied to the circuit. The output is then shifted to the right by k positions and compared with the previous output. Any disperancy indicates the presence of an error. For one-dimensional iterative arrays, this method is straightforward and requires simply the extension of the iterative array by k cells, a comparator, a register, and the circuitry for operands shifting. The extension of this method to two-dimensional arrays is described in [65]. In a similar approach, called *recomputing with swapped operands* [66], the inputs are swapped in the second phase, instead of being shifted.

Although these approaches use time redundancy, they still require a large amount of extra hardware. In [66], it is shown that the application of RESO technique to an adder results in a larger overhead than with a simple duplication. A new technique is proposed for the adder circuit in [66] which requires less hardware than duplication and uses two phases to complete the addition of two operands. The n-bit adder, as well as the operands, is divided into lower and upper halves with each half consisting of $n/2$ bits. In the first phase, the lower halves of the operands are added in both halves of the adder, the results are compared and one result is stored to represent the lower half of the final output. In the second phase, the same operation is repeated with the upper halves of the operands. The reduction in hardware is due to the fact that the register and the comparator are less than half the size of those required in the RESO approach. Also, the selection between upper and lower operand halves requires less circuitry than the shifting of operands. This method is also shown to be faster than the RESO technique.

Another approach, which does not require a circuit to be an iterative array, is *alternating logic*. In this technique, all binary variables are required to be alternating. A binary variable x_i is said to be alternating, denoted by the 2-tuple $(x_i, \overline{x_i})$, if its true value during one time interval is followed by its complemented value during the next time interval [67], i.e., $(x_i, \overline{x_i})$ assumes values from the set $\{01, 10\}$. A circuit in which the output variable alternates when synchronised alternating variables are applied as inputs, is called an alternating circuit. In order for an arbitrary function f to be implemented by an alternating circuit, it must satisfy the condition $f(\overline{X}) = \overline{f(X)}$, which is the definition of a self-dual function. It is evident that not all combinational functions are self-dual. However, it can be proved that any arbitrary function f of n variables can be represented as a self-dual function f^* of $n + 1$ variables. Moreover, the added variable is essentially a clock which changes the function realised from f^*, when the clock is 1, to $\overline{f^*}$ when the clock is 0 [67].

By analogy with the theory of totally self-checking circuits, an alternating circuit is said to be totally self-checking if for every fault there exists at least one alternating input $(X, \overline{X})$ which produces a non-alternating output (the self-testing property), and also that there exists no input $(X, \overline{X})$ which produces an erroneous alternating output (the fault-secureness property). However, it is known that not all alternating circuits are fault-secure. Only those faults that result in a non-alternating output are detected. For example, if a stuck-at fault is sensitised by both input vectors X and $\overline{X}$, it will result in an incorrect alternating output. Conditions under which this situation may occur are discussed in [67]. In addition, the transformation of an arbitrary function into a self-dual function may require a 100% increase in hardware.

A class of *self-exercising* combinational circuits is introduced in [68] which tries to combine the properties of alternating logic and totally self-checking circuits. In addition to the requirements of alternating logic, that all logic variables must be alternating variables, all variables are also duplicated in such a way that a zero on a line in the normal circuit will be represented by a (01) on the two lines of the self-exercising circuit and a logic 1 is represented by (10) (dual-rail variables). This results in a massive hardware redundancy, even more than Triple Modular Redundancy, TMR[55], and offers only error-detection and not error correction.

An approach, which does not belong to the category of on-line error-detection techniques, but is worth mentioning in this section, uses the idle time in a two phase design to apply tests to the *dormant logic* in a circuit [69]. Dormant logic is that part of a circuit that does not participate in data processing during some interval of time. The approach does not provide on-line error-detection in that the dormant logic is exercised by test vectors instead of the actual data, but this is done concurrently with normal operation. The approach is conceptually simple, and the additional hardware may also be used for off-line built-in test.

Another approach, close to the above one, is presented in [70], where the normal operation of a self-checking microprocessor is periodically interrupted to apply part of an off-line test sequence. This points to the important fact that even for on-line fault detection, off-line testing is still necessary. This has already been recognised explicitly in [71] and implicitly in [72].

2.4.3 Off-Line Fault Detection

Off-line fault detection, or simply testing, is carried out by manufacturers to ensure that the devices contain no manufacturing defects, by the users of such devices in what is called *incoming inspection* before the devices are assembled into systems, and it is also performed as part of maintenance activities.

All testing involves three steps: test pattern generation, test evaluation, and test application. In the following, these three steps are described in more detail mainly to highlight the difficulties encountered in each of them.

a) Test Pattern Generation

The test pattern generation task consists of deriving a set of test vectors that will detect all faults from the assumed fault set. It is a tedious task, even for computers and, therefore, it was one of the first digital design related activity to be investigated for automation [73].

Many algorithms and procedures have been developed over the years for Automatic Test Pattern Generation, ATPG, of which the D-algorithm [74] is the most prominent. Other ATPG approaches [75, 73], developed subsequently, are basically improvements over the D-algorithm, in terms of computational efficiency.

The problem of fault detection in general combinational circuits has been shown to be NP-complete [76], i.e., it is unlikely to be possible to find an algorithm that would derive a test for a fault in a time that is polynomial in the number of input lines and the number of gates. Polynomial time algorithms have been developed only by restricting the structure of combinational circuits such as by limiting the number of inputs to each gate, or by restricting the interconnection patterns between gates etc. [77]. According to empirical studies [78], the cost of generating tests for 100% coverage of single stuck-at faults is proportional to the square of the number of gates.

Recent developments in ATPG have used heuristics and other techniques borrowed from artificial intelligence in an attempt to reduce the computational costs [79, 80]. Some researchers propose the elimination of ATPG altogether and suggest the use of randomly generated test patterns or exhaustive testing [81, 82]. The interest in random testing stems from the fact that the first few vectors detect a large number of stuck-at faults, regardless of the method used for their generation. However, as the number of random vectors increases, the number of faults detected by each vector decreases. In exhaustive testing, the problem of excessive test lengths due to large number of inputs is tackled by partitioning [81, 83]. The circuit is partitioned in such a way that the partition with the largest number of inputs can be tested in a reasonable time. Note that the general problem of circuit partitioning is NP-complete [83, 84] as the problem of test derivation.

The major disadvantage of random and exhaustive testing is that they both are based on the stuck-at fault model. Also, they require much longer test sequences than deterministic test generation. There might even be *random pattern resistant faults* [85] that cannot be detected by random test sequences. This led some researchers to 'bias' the random pattern generation to improve the fault coverage and decrease the test sequence lengths [86, 87, 88], whereas others attempted to design circuits so that they are testable by random patterns [89, 90]. Another problem with random testing is in estimating the fault coverage [91, 92]. Fault simulation is the principal means of determining the fault coverage, but given the longer test sequences, this requires excessive computational times.

While the problem of ATPG for combinational circuits is considered as 'solved' by some [54] and only 'trackable' by others [92], the problem of ATPG for sequential circuits has still not received an acceptable solution [93]. Bennetts in [94] gives several examples to illustrate the difficulties encountered in generating tests for sequential circuits. One of the main difficulties seems to be the fact that a sequence of patterns is required to sensitize a fault and another sequence is required to propagate the fault to an observable output, whereas in the case of combinational circuits, a single test pattern performs both the sensitisation and the fault propagation.

Various attempts have been made to extend the D-algorithm to sequential circuits [54]. However, 'it is very easy to design a circuit that defeats the proposed procedures' [94] and there is still no general purpose algorithm that suits complex sequential circuits.

The current trend is to constrain the design of sequential circuits to some structured design methodologies, as will be discussed in Section 2.4.2.

All the above problems are exacerbated to a large extent if non-classical faults, such as CMOS stuck-open and stuck-on faults, are taken into consideration.

Finally, it should be noted that all test pattern generation algorithms and procedures that rely on fault enumeration, i.e., consideration of each individual fault separately and derivation of a test for it, are of limited use when the number of faults becomes very large as in VLSI.

b) Test Evaluation

The test evaluation task is the process of estimating the effectiveness of a test set in detecting the faults. Fault simulation is the principal means of evaluating the effectiveness of test sequences derived by the test generation process [95]. The list of target faults is often kept deliberately small in order to reduce the computational costs of test pattern generation. Once a test sequence has been derived, fault simulation is used to see whether it also detects other faults not included in the fault list. Another important use of fault simulation is in the test generation process itself. After an input pattern has been derived for testing for a particular fault, it is usual to invoke the fault simulator to see what other faults are also detected by this vector so that they can be dropped from the fault list [92, 96].

For a circuit consisting of G gates, the cost of fault simulation is proportional to G^n, where $n \geq 2$ [78]. Various methods have been developed to speed up fault simulation, ranging from purely software methods [95] to dedicated hardware accelerators [97].

c) Test Application

Test application has traditionally been carried out on automatic test equipment that feed the circuit with test vectors and evaluates its output response. Test application is carried out differently, depending on whether the chip is tested by itself or as part of a printed circuit board. The former situation occurs after the chip manufacturing stage and at the incoming inspection, whereas the latter occurs after the PCB assembly stage and in maintenance activities. In the following, the problems involved in the test application to a chip by itself will be discussed, noting that these problems are aggravated when applying test patterns to a chip that is mounted on a PCB.

The test sequence is applied to the chip by equipment that must also store the expected output response for comparison with the actual chip response. Two major drawbacks result from this approach. The first is that it is difficult to exercise the chip at its maximum operating speed so that the usual approach is to apply the test sequence at a much lower frequency (static testing), and then check some of the timing parameters of the chip, such as propagation delays, with a few extra input patterns (dynamic testing). This is clearly unacceptable, since the dynamic testing is not comprehensive enough. The second problem is that, for a typical VLSI chip, the amount of data to be stored on the tester (test sequence and expected response) can very easily become enormous and exceed the tester storage capability.

Many solutions have been devised to address this problem . A first method consists simply of comparing the chip output response to another similar chip that is assumed to be fault-free (called a *golden unit*). In all other methods, the basic idea is to compress the output response into a 'signature' and then compare it with a precomputed correct signature, instead of performing a bit-by-bit comparison. The various methods differ in the techniques used for the data compression. In transition counting [98], the number of transitions in the output response constitutes the signature. In signature analysis [92], the output response is represented by a polynomial. It is divided by the characteristic polynomial of the signature analyzer, and the remainder of this division is compared to a precomputed remainder.

It is clear that two completely different output sequences can have the same transition count or the same remainder, when divided by a certain polynomial. Therefore, not all erroneous output sequences will be caught by test response methods that rely on data compression. Furthermore, it is difficult to analyze the conditions under which this problem occurs [99, 100].

2.4.4 Design for Testability

Design for testability aims at avoiding the problems discussed above. The concept was introduced by workers involved in testing activities who realised, long before the advent of VLSI [101], that testability issues should be addressed at the design stage. Another reason that prompted considerable research in the field was the unsolved problem of test generation for sequential circuits. This resulted in guidelines for designers which consisted in avoiding circuit structures that are difficult to test, and the provision of some test aids in the design, such as addition of test points, global reset lines, etc. An extensive list of such guidelines can be found in [93].

The above guidelines are usually qualified as *ad-hoc* techniques, in that, each particular design requires special consideration for their application. More structured testability techniques appeared with the introduction of latch scan arrangements [102] that reduced the problem of testing sequential circuits into that of testing combinational circuits. All latch scan arrangement methods share the same principle, in that they aim to gain access to the latches within the circuit to control and observe them for test purposes. All methods achieve this aim by configuring the latches so that they form a shift register that is accessed by serially shifting in or shifting out bit patterns. The exception to this is random-access scan [103] where the latches are arranged in a configuration similar to that of a random access memory so that they can be accessed in a similar manner.

Configuring the latches for a scan arrangement so that they form a shift register requires additional area [104]. However, the need to devote a small part of a chip to testing purposes has been recognised and accepted, even by chip manufacturers, especially since the introduction of IBM's version of scan path, the Level-Sensitive Scan Design (LSSD) [92, 102].

A major drawback of scan latch arrangements is that the application of a test pattern to a combinational section involves the shifting in of a lengthy string of bits, and observing the output response involves a similar operation. This increases the time required to apply a single test vector by $2n$ times, where n is the number of latches in the design. Solutions to this problem consist of configuring the latches into a number of shift registers instead of a single one [105]. Other methods, called *partial scan* [106] advocate the incorporation of only some of the latches into the scan path.

The extension of scan path techniques for testing CMOS stuck-open faults presents some problems. A CMOS stuck-open fault is detected with an ordered pair of vectors: an initialisation vector followed by a test vector. In the process of shifting-in the second vector, after the application of the first one, the combinational circuit under test will see many input patterns that are very likely to invalidate the initialisation. One solution would be to use a 3-latch scan register, one of which is not included in the scan chain and is used to hold the initialisation input until the test input has been shifted-in. Liu and McCluskey [107] consider that the 3-latch scan register introduces too high an area overhead. Their proposed solution is based on the use of "simplified" two-pattern tests, where it is assumed that the all-1 input to a CMOS gate sets the output to 0 and the all-0 input sets the output to 1. If a gate receives both input x and $\overline{x}$ then they must be assigned the same value when applying the all-0 or the all-1 inputs. The conversion of a combinational circuit into a circuit that is testable with simplified two-pattern tests requires an area overhead ranging from 7% to 19% (before the introduction of scan-path). In addition, an embedded gate in the transformed circuit must not have both x and $\overline{x}$ as inputs, which is not always possible to ensure. If a circuit is testable with simplified two-pattern tests, then a simple modification of the scan-register can be used to test CMOS stuck-open faults in a scan-path design [107].

Boundary scan [108], is an extension of scan path techniques that is aimed at helping test and diagnosis of digital boards populated with VLSI devices and surface mounted components.

Once the latches have been reconfigured into shift registers, it takes little extra hardware to convert them into test pattern generators (typically Linear Feedback Shift Registers, LFSR's) and/or test response evaluators (signature analyzers). Performing such a conversion will result in a design that will test itself, Built-In Self-Test BIST, which has been described as the ultimate solution in design for testability [109]. Because such a conversion requires little extra hardware, above that required for a scan path, most of the reported approaches to BIST rely on pseudo-random testing [110, 111, 112], using test patterns generated by an LFSR with response evaluation by signature analysis. These approaches suffer from the same problems as in random testing, mentioned in the previous section, namely, the detection of non-classical faults is not addressed, the long near-exhaustive test sequences, and heavy reliance on simulation for the computation of fault-free signature and assessing the fault coverage. Pseudo-exhaustive testing alleviates some of the problems of pseudo-random testing [113, 81]. Exhaustive test sequences can detect all stuck-at faults.

The volume of data representing the test response of a circuit is directly related to the length of the test sequences, which are much longer in pseudo-random and pseudo-exhaustive testing than in the case where test sequences are generated in a deterministic manner. Therefore, the only option for processing the test response data is to compress it into a much smaller amount, referred to as *signature*. The different data compression approaches used in testing are described in [92].

An obvious limitation of test data compression is as follows: if a fault in the circuit under test produces an output response that has the same signature as the fault-free response, then the fault is not detected. Furthermore, it is clear that many different sequences can have the same number of 1's, the same number of transitions, or can yield the same remainder when divided by a given polynomial.

Signature analysis is, by far, the most popular approach. An m-bit long circuit response is compressed into an n-bit signature, where m is much larger than n. In this case, there are 2^n signatures and 2^m possible circuit responses, only one of which is correct. Therefore, there are approximately 2^{m-n} different sequences that get compressed into the same signature, so that $2^{m-n} - 1$ faulty circuit responses are compressed into the correct signature. If $n = 16$ and $m = 1024$, for example, then there are $2^{1008} - 1$

faulty responses that would escape detection. This phenomenon is known as *aliaising*. Aliaising errors effectively reduce the fault coverage of the input test sequence. Even when the input test sequence achieves a 100% fault coverage, some of the fault may escape detection if signature analysis is used to process the circuit response, thus, reducing the effective fault coverage. Furthermore, it is very difficult to determine all the conditions under which aliaising errors occur [100]. The determination of the fault-free signature also requires long simulation times.

Finally, it should be pointed that the literature contains many articles about the benefits and necessity of built-in self-test [114, 78]. This contrasts with the timid attempts made in commercially available integrated circuits [115, 116, 117, 118, 119]. If chip designers are reluctant to sacrifice silicon estate to improve testability, yield or reliability, they will be even more reluctant to sacrifice performance for the above goals. When a new microprocessor is released, the publicity stresses the number of MIPs rather than the cost of testing the microprocessor or its mean time to failure!

2.5 FAULT CORRECTION

Although fault-detection is important for fault-tolerance, it does not by itself provide any protection against faults. Having an indication of the presence of a fault is preferable to using an incorrect output, but it is sometimes necessary to correct the output from the faulty value.

In certain circumstances, a warning of the presence of a fault is quite adequate, such as in cases where temporary interruption of operation is not detrimental so that maintenance can be carried out. Note that in these cases, the duration of the interrupted operation and the cost of maintenance is directly related to the resolution of the fault-detection mechanism. The smaller the part reported as faulty, the higher the resolution, and thus, the shorter the down-time and the lower the maintenance cost. This fact alone makes the fault-detection attribute nearly as important as the fault-correction one, since a large proportion of the electronic systems in current use fall into this category.

However, there are many critical applications where interrupted operation cannot be tolerated and/or maintenance is impossible, and the number of such applications is on the increase because of the growing reliance on electronic systems in practically all aspects of our life (hospital patient monitoring equipment, aircraft and car electronics, real-time control computers, etc.). Generally speaking, interrupted operation cannot be tolerated in real-time systems where quick response is of paramount importance. Maintenance may also be impossible, or very costly, in remotely sited installations, such as spaceborn applications. In such situations, it is mandatory to provide the correct outputs despite of the presence of faults.

As far as integrated circuits are concerned, and as was discussed in Section 2.2, faults are the result of manufacturing defects and physical failures during normal operation. Normally, all chips containing manufacturing defects are thrown away. If possible, even those chips containing weaknesses that would result in an early failure are also discarded. However, as chip areas are increased and feature sizes are decreased from year to year, a large proportion of the manufactured chips may have to be discarded, resulting in lower yields and higher cost per chip. The response from manufacturers was to depart from the usual practice of throwing away defective chips by trying to repair them instead. So far, however, the only commercially available products for which fault-tolerance is routinely and successfully adopted are high density memories [120].

When a chip fails in normal operation the situation is very different. Currently, it is unthinkable to send back a chip to the manufacturer to be repaired (however, this might be an option when unit chip costs are high enough, and the cost of repairing them is sufficiently low; in which case the cost of repair will almost certainly be dominated by the cost of fault diagnosis). Other approaches, similar to the ones developed over the years to cope with electronic system unreliability have to be used for fault-tolerant integrated circuits.

Fault-tolerance strategies that are aimed at repairing manufacturing defects will be referred to here as *defect-tolerance.* Similarly, techniques developed for operational failures will be referred to as *failure-tolerance.* In the following sections, it will be seen that current defect-tolerance strategies cannot generally be used for reliability improvement, although many failure-tolerance techniques may also be used for yield improvement. The discussion does not address the question of whether 'what is good for yield is good for reliability' [19]. This question seems to be more involved with process engineering (which is beyond the scope of the thesis) rather than with chip design. For instance, to lessen the problems of electromigration in aluminium interconnects, layered structures have been suggested [121]. However, this requires a more complex process which is inherently more prone to defects so that it has a lower yield to achieve a longer lifetime. As far as chip design is concerned, it will be seen in Chapter 5 that the introduction of redundancy into a design can be beneficial to both yield and reliability, and that the only instances where trade-offs between the two might be necessary are in choosing the optimum amount of redundancy, the distribution of the redundancy and the way the chip is partitioned.

In the following sections, we will first discuss the various ways of repairing a defective chip, mentioning the reasons of their suitability or unsuitability for failure-tolerance. This is followed by a description of the many possible ways so far devised to tackle operational failures and their relevance to defect-tolerance.

2.5.1 Defect-Tolerance Strategies

As soon as manufacturers mastered their art in making chips, they started investigating ways of increasing the number of devices they can put on a single die to get higher densities and more functionality [122]. They were rapidly confronted with the problem of the occurrence of mainly random manufacturing defects, which was much worse in those times than today. Their approach was to limit the die size so that the probability of having a defect on it is reasonably small. However, the resulting die size limits were too small (and the feature sizes too large) for implementing the highly complex circuits aimed at. It became apparent that the presence of defects on the die has to be accepted, and the problem statement became 'how to get a correctly functioning chip out of a die containing defects?'. It was also realised that a solution to this problem will allow the manufacturing of whole-wafer circuits, with all the potential benefits that would result. Implementing wafer-scale integrated circuits was, and still is, the driving force behind the development of defect-tolerance strategies.

The earliest defect-tolerance approach was *discretionary wiring* [123, 124]. It consisted of probe testing the wafer before the circuits are completely connected and then tailoring a metallisation pattern to connect up the working cells. This approach suffered from three problems. The first is that probe testing might add defects to the wafer by probe damage or contamination. The second is that each individual wafer requires its own metallisation pattern because of the random distribution of defects across wafers. The third problem is that the final metallisation patterning must be defect-free. The problem of testing was addressed in [125] where it is suggested that an initial metallisation layer should be included solely to facilitate testing, and that it should be selectively removed afterwards (this work also addressed other testability issues). The problem of the number of individual chip metallisation patterns is addressed in [126] where it is shown that under certain circumstances, the same metallisation pattern may be used for several wafers. Another discretionary wiring technique, which was ahead of its time, was proposed by McKintosh and Green [127]. An electron-beam, controlled by a computer holding a map of the wafer's defective cells, is to be used to pattern the metallisation layer.

A disadvantage of discretionary wiring, and all methods that rely on mask programmability, is that they are not suitable for large production runs. The approach is also dismissed as a defect-tolerance strategy in many publications on wafer-scale integration [128], but it has been used successfully in a recent design of a wafer-scale FFT processor at Fujitsu [129]. The technique is also being considered as a solution to the problem of speed losses due to long interconnect in WSI [130].

In current defect-tolerance approaches, the interconnections between cells are interspersed with switches that are programmed, after manufacture and test, to connect or disconnect cells. The various methods differ in the technological implementation of the switching mechanism. Figure 2.8 shows a possible classification of the different switching mechanisms [131].

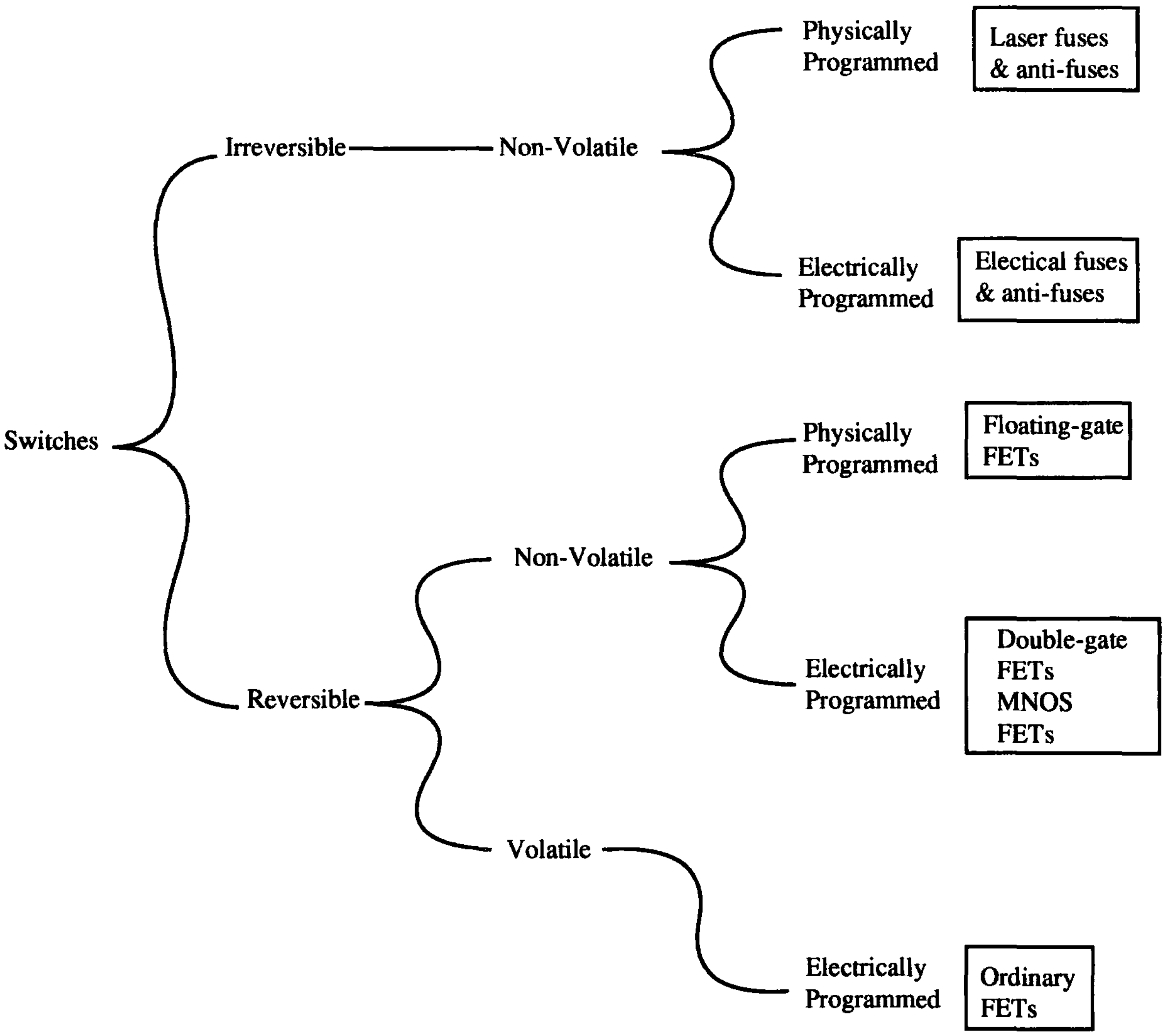


Figure 2.8 Classification of the switching mechanisms.

In laser restructuring, fuses and anti-fuses are laser-programmed to connect/disconnect cells to/from a network of cells [128]. Electrically-programmable fuses and anti-fuses [132, 133], where high electrical currents are used to make or break connections, are proposed to alleviate some of the deficiencies of laser restructuring, such as the requirement of expensive laser optics and low throughput. The use of floating-gate transistors programmed by an electron-beam, in place of the fuses and anti-fuses, has also been shown to be a viable approach [134].

Both laser-programmable and electrically-programmable fuses and anti-fuses are irreversible switches, so that once a defect-free cell connected to the network subsequently fails, it is not generally possible to disconnect it. In addition, laser programming works on unpackaged chips which makes the technique of very little use for field failures. Although electrically-programmed fuses and anti-fuses have the potential of repairing some operational failures, since a packaged chip can be programmed through external pins, their use is limited by the irreversible nature of the switching mechanism. A disadvantage of electrically-programmed fuses, over laser restructuring, is the requirement of access circuitry that must be able to handle the large currents used for programming. Floating-gate MOS transistor switches are theoretically reversible, through exposure to UV light for erasure, but because the programming is carried out with an e-beam, they present access problems once a chip has been packaged. Programming floating-gate transistors by applying high voltages to the drain-substrate junction, as in EPROMs, restricts their usage as a general purpose switch (ROM array have a simple regular access path for such programming since the drain of each floating-gate transistor is accessible through the bit lines).

Controlled floating-gate transistors and Metal-Nitride-Oxide-Silicon (MNOS) transistors, referred to as *double-gate* Fets [230] in Fig. 2.8, represent a better alternative for high voltage electrical programming than floating-gate transistors. However, so far the use of double-gate transistors has been confined to EEPROM products and Field Programmable Logic Devices (FPLDs). This switching mechanism has great potential for defect-tolerance and the repair of field failures, because of its reversible nature and because programming is performed from outside the chip, especially with the new breed of FPLDs represented by the general-purpose reconfigurable architectures [135] that are user-programmed to implement a user-defined function.

The simplest restructuring approach, called bridging-bond links, is proposed by Jesshope and Ben tley [136]. The link consists of a piece of metal layer of area comparable to a conventional bond pad with an additional gap of a few micron bisecting the link. Programming is performed by depositing a gold ball across the gap using a standard wire bonding machine. The size of the link limits this approach to the switching of large area cells only. The authors propose to use the method in a hierarchical fashion with other restructuring techniques used at lower levels and the bridging-bond link at the highest level.

Laser and electrically-programmable fuses and anti-fuses are extensively used in commercial high density memories [137, 138, 139]. Typically, a few spare rows and/or columns are included in the design and they can be switched in to replace defective rows and/or columns. Reference [140] gives a list of published designs of defect-tolerant memories, up to 1986, together with their organisation, the restructuring technique, and the amount and distribution of the redundancy.

Moore et. al. [141] attribute the success of defect-tolerance in memories to their repetitive architecture and their unique feature of not requiring any logical connections between cells. Memories have another unique feature: they are the only mass produced chips to have a regular architecture. If memories were to be made fault-tolerant using soft reconfiguration techniques they would not be as attractive. Therefore, the success of defect-tolerance in memories is due not only to their unique architecture, but also to the use of physical restructuring and their highly competitive market.

The successful application of defect-tolerance in memories led many researchers to investigate its extension to other architectures. The first candidates for such investigations were regular architectures, mainly one and two-dimensional arrays of identical processing elements. At this stage, there was a shift away from the techniques used in memories because rows and/or columns replacement techniques proved to be too wasteful when the PEs have any greater functionality than just a storage cell. When researchers investigated other, less wasteful techniques, they found that physical restructuring, which is acceptable for a small number of replacements, becomes impractical for a large number because of its inflexibility. Note that the spare PEs are still provided in the form of spare rows and/or columns, since this is the only way to preserve the regularity of an array, but it is the PEs that are discarded if faulty, instead of whole rows or columns. Almost all the proposed flexible schemes rely on soft-reconfiguration, where the switching mechanism consists of ordinary MOS transistors. This is discussed in the next section.

2.5.2 Failure-Tolerance Strategies

Achieving failure-tolerance is harder than defect-tolerance for several reasons. The most obvious is that external physical restructuring cannot be used against failures. Repairing manufacturing defects is a one-time process, at the end of the manufacturing stage. Tackling operational failures has to be a continual process, since failures may occur any time during operation. In addition, a manufacturing defect is harmless in the sense that, it is either repaired or, if not possible, the chip is thrown away, increasing the chip cost. However, failure can be very harmless indeed, even in non-critical applications, and the cost involved in dealing with a failure may be orders of magnitude higher than the increased chip cost due to manufacturing defects. It has been estimated that the cost of detecting and locating faults increases exponentially with the level at which it is performed (chip level, board level, system level) [102], and since operational failures occur when the chip is part of a system, they are the most costly to detect. In safety-critical applications, the costs may be incurred in terms of human life, loss of a complete space mission, or dangers to the environment and whole communities.

As stated earlier, the methods developed over the years to deal with system reliability (an extensive list of which can be found in [142] and [55]) have constituted a pool out of which on-chip fault-tolerant schemes have been selected.

Error-correcting codes can be used for failure-tolerance. They have the advantage of not requiring special testing, reconfiguration and control circuitry. However, the area and timing overheads due to error encoders/decoders, which must be fault-free, tend to become prohibitive. Error-correcting codes also require very large redundancies in order to allow recovery from more than one signal error. In addition, as was stated for error-detection, the lack of a uniform coding methodology that could be used throughout a digital system, is another drawback of error codes. Error-correcting codes are however routinely used in random access memories, where they are mainly aimed at correcting soft errors rather than permanent failures. The use of error codes is only efficient for large memory chips because of the need to offset the overhead of the encoding and decoding circuitry. In situations where a chip contains only a small amount of memory, such as in an ASIC, other approaches have to be used, as in [143], where the addresses of faulty cells are stored, and whenever they are accessed, the control circuitry re-routes the access to a spare bank of memory cells.

Fault-tolerance techniques that rely on replication and vote-taking can be found in any safety-critical application in the form of Triple Modular Redundancy (TMR), its generalisation N-Modular Redundancy (NMR), or one of its derivatives or extensions. The advantage of these methods include simplicity, hence ease of design, and the fact that they do not require special testing and reconfiguration circuitry. However, the straight implementation of these methods onto chips has not been an attractive option because of the high redundancies, and also because of common-mode failures [144], i.e., the simultaneous failure of replicated units, which is aggravated by the clustering of manufacturing defects. One approach to overcome the common-mode failures is to use non-identical copies, such as a unit and its complementary, as described in [72, 144]. In principle, the problem of defect clustering can be avoided through the 'scattering' of the different copies over a wide area, but this creates other problems with the placement of the voter and the routing of signals [145]. When replication is carried out in conjunction with partitioning, it has some potential for yield improvement [140]. Other approaches, that avoid explicit voting and replication have also been investigated, mainly in the early 1960's [146], but they have never been implemented because of their massive redundancies, although they keep reappearing in recent publications [147, 148].

The most recent approaches to fault-tolerance, targeted specifically at VLSI/WSI implementations, are those resulting from emulating the development of fault-tolerant memories, i.e., those involved with networks of identical PEs. Two-dimensional arrays are the most popular since it is an all-purpose structure in which arbitrary networks can be embedded [149]. A wide range of computations can be effectively implemented on 2-D arrays. Some application, such as real-time signal and image processing, mandate the use of highly parallel computation engines in the form of 2-D arrays. Such applications usually require large numbers of PEs, i.e., they require silicon areas that are beyond the capabilities of current manufacturing technology. But, structures consisting of identical PEs have a distinctive advantage for fault-tolerance since redundant elements can be used to replace any faulty element.

The MOS transistor used in soft-reconfiguration requires a data storage cell to control its state (on or off) in addition to the interconnections to set the desired state at each switching site. The overhead associated with the storage cell and the interconnections for its control should be modest relative to the size of the PE replaced by reconfiguration. This degrades the advantage of very small, such as bit serial, replaceable PEs [145]. The high ON-resistance of the MOS transistor is a disadvantage, compared to fuses and anti-fuses, since it degrades speed. Furthermore, MOS transistors are unsuitable for some repair problems, such as defective clock or power distribution networks. Despite these problems, soft-reconfiguration has major advantages over hard-reconfiguration (physical restructuring):

- the switches are ordinary MOS transistors and do not require any special processing,
- no special setup is required for reconfiguration, and
- it can be used for both defect-tolerance and failure-tolerance,

the last point being probably the most important. Besides, the problems associated with soft-reconfiguration can be lessened, at least. Speed degradation can be tackled by deriving reconfiguration schemes that limit the number of switches on any path. The problems of faults in the clock and power distribution networks are addressed in [150] and [151], respectively. When small PEs, such as bit-serial, have to be used, then the approach of Chen et. al. [152] might be considered.

Configuring an array of identical PEs, some of which are faulty, into an array consisting of fault-free PEs is not an easy problem. Even the reconfiguration algorithms for the simple spare rows/columns in DRAMs are NP-complete [153]. Practical reconfiguration procedures for defect-tolerant DRAMs use heuristics to speed-up reconfiguration, at the expense of not repairing all repairable chips for example. The problem of reconfiguring a one-dimensional array using fault-free elements of an N-element array is NP-complete if the length of the longest wire is to be minimised. Furthermore, there are some arrangement of good and bad elements for which even the optimal linear array has unacceptably long wires. Thus, optimal solutions, even if they could be obtained quickly, are not always practical [149]. Reconfiguration algorithms for 2-D arrays are clearly very complex.

Most reconfiguration algorithms proposed in the literature assume that the interconnections between PEs are fault-free. This is justified by the fact that the interconnections are much simpler, and hence more reliable, than the PEs. Another assumption is the independence of faults in different PEs which is justified if the faults affect areas that are substantially smaller than the area of a PE. The least justifiable assumption concerns the existence of a perfect fault-detection and location mechanism. The task of testing VLSI circuits is already recognised as very complex. Testing an array of PEs, where each PE may be as complex as present day VLSI chips, is bound to be a difficult problem.

True failure-tolerance implies the provision of on-chip mechanisms for fault-detection (self-test) and fault-correction (self-repair). Many reconfiguration algorithms require that each PE be able to report its status (good/bad). The self-test problem will have to be solved at the VLSI level before moving to higher levels of integration. Self-repair, on the other hand, seems unlikely given the complexity of current reconfiguration algorithms. However, neat solutions are always possible, as exemplified by the self-organising 2-D array proposed by Evans and McCanny [154].

2.6 YIELD AND RELIABILITY

It is clear from the discussion in the previous two sections that, whatever techniques are adopted for fault-detection and/or fault-correction, their on-chip implementation requires additional area, which itself increases the susceptibility to manufacturing defects and operational failures. Fault-tolerance would be beneficial only if the faults that can be tolerated outnumber the faults that may occur as a result of the additional circuitry. This way of assessing the effectiveness of fault-tolerance is impractical, because of the difficulties involved in enumerating the possible faults and establishing whether they can be tolerated. The usual approach is to compare the yield and/or reliability of the fault-tolerant and non-fault-tolerant designs. Chapter 5 is devoted to this subject. A brief introduction of the topic follows.

Yield is defined as the fraction of chips that pass the manufacturers' final test. A yield model allows the prediction of a yield figure based on characteristics of the processing line before actually manufacturing the chips. The importance of yield modelling, even for non-fault-tolerant ICs, is illustrated by the extensive coverage it has received in the literature even since the early days of microelectronics [155].

The development of yield models for non-fault-tolerant ICs concentrated on the determination of probability distribution functions that describe the random distribution of defects across wafers. This involves, among other things, the classification of the manufacturing defects according to some criteria [2], and the determination of the frequency of occurrence of the different classes of defects. The early yield models were simply dependent on the chip area and the average defect density of the processing line. More sophisticated models were later developed to take into account other factors, such as the different types of defects, their random sizes, clustering, ... etc [156]. These developments resulted in very complex yield models which, in recent years, have prompted a shift away from analytical models towards simulation methods [157, 158].

Developing yield models for fault-tolerant ICs is somewhat different, since the main objectives are the comparison between fault-tolerance strategies and the selection of design decisions that maximise the yield improvement, such as the amount and distribution of redundancy. Chapter 5 is more appropriate for discussing these issues.

The reliability of a system determines the likelihood that the system will continue to operate satisfactorily. It is a statistical parameter derived by the methods of probability theory that evaluate success or failure [159]. The reliability of a system at time t is defined as the conditional probability that the system is working at time t given that it was working at time $t = 0$. It is measurable by observing N_0 identical systems, operating under specified conditions, and noting their times of failure. The reliability at time t is the fraction of systems still working at time t. An estimate of the failure rate at time t is given by

$$\lambda(t) = \frac{1}{t}\frac{\text{Number of failed systems at } t}{\text{Number of working systems at } t}$$

A common graphical representation of the failure rate, called the 'bathtub' curve is shown in Fig 2.9. Failures occurring in region 1 are termed *infant mortality* and they are attributed to components of poor quality resulting from variations in the manufacturing process. Region 2 represents the *useful life* of components. Failures in this region are considered random. They are caused by long term failure mechanisms or by the operating environment. Failures in region 3 are known as *wear-out* failures [15].

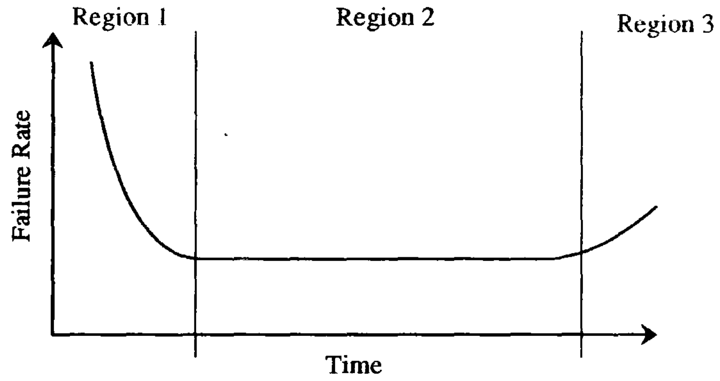


Figure 2.9 The bathtub curve.

The failure rate of an IC is usually expressed in FIT (1 FIT = 1 failure/10^9 hours). Current ICs have failure rates in the range of 100 to 1000 FIT. It is predicted that by the end of the century, ICs will have a failure rate of the order of 10 FIT [160]. The difficulties involved in measuring such low failure rates are illustrated in the following example. Assume an experiment is carried out to estimate the failure rate using N_0 identical devices. The measurement continues until time T where there are $N_f = uN_0$ failed devices, $0 < u < 1$. The failure rate is $\lambda = (N_f/T)/(N_0 - N_f)$ and the time required for such measurement is

$$T = \frac{1}{\lambda}\frac{u}{1-u}$$

It is clear from Fig. 2.10 that such measurements are virtually impossible. In practice, *accelerated life testing*, where the devices are operated at high temperatures, is used to shorten the observation times. This will have the effect of lowering the curves in Fig 2.10. However, with a 10 FIT failure rate, even accelerated testing becomes obsolete [160]. For this reason, there is currently a change in the methodologies used to ensure product reliability. As Crook put it [160], "In the future, there will be less focus on developing precise measurements of failure rate by sampling the output of a manufacturing line, and more focus on understanding and controlling the many input variables of a manufacturing line which ultimately affect product reliability". This concept is called *building-in-reliability* [161]. It is based on the fact that failure patterns in ICs are determined uniquely by the process used to create them and by their basic physical structure.

An interesting point about the concept of building-in-reliability is that if the input parameters of a process can be controlled so that failures are eliminated or reduced then this same control can be used to eliminate or reduce manufacturing defects. Hence, there is a clear convergence of the methods used for yield improvement and reliability

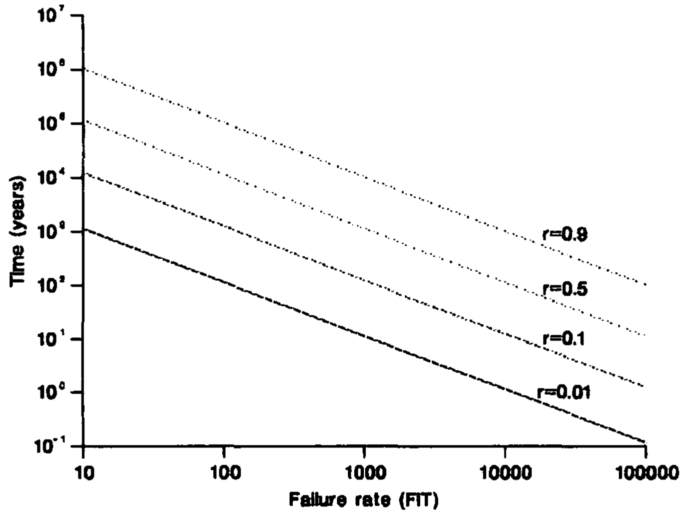


Figure 2.10 Time required for measuring the failure rate of an IC.

impression that the concept of building-in-reliability aims at a perfect manufacturing process. When Shaft was asked whether he believed that this was possible, his reply was "Is it possible to reach infinity?". Hence, building-in-reliability is an ideal, but, as put by Amerasekera [15], if that ideal is aimed at then the reliability is bound to improve.

Another important application of accelerated life testing is in screening procedures. Screening is the process by which defective devices are detected and eliminated from a production batch. Screening, sometimes referred to as burn-in, consist of testing devices under environmental or electrical stress in order to accelerate their lifetime such that the devices begin operational life with a failure rate corresponding to that beyond the infant mortality region [15].

The discussion of reliability in the context of fault-tolerance is left to Chapter 5.

2.7 CHAPTER SUMMARY

This Chapter has reviewed the present extensive knowledge of faults, fault modelling, fault detection and correction and related matters, with emphasis on those parts of the subject that are particularly relevant to the original contributions described in Chapters 3-6. The main conclusions are as follows.

Faults in integrated circuits may be due to: (i) manufacturing process defects or (ii) external disturbances. Given the current status of the semiconductor industry, the prospect of a perfect manufacturing process seems extremely unlikely. Furthermore, the widespread use of electronics in a variety of environments, including hostile ones, increases the susceptibility to external disturbances. Hence, the occurrence of faults should always be considered as a likely event, and in cases where faults may have dramatic consequences, the use of fault-tolerance is mandatory.

Fault-tolerance can be described in terms of two factors: the detection of faults and the correction of faults, both of which require a thorough knowledge of the nature of faults and their effects on circuits. The effects of physical faults listed in Section 2.2 ranged from short circuits and breaks in interconnections to threshold voltage variations and excessive leakage currents. These effects are difficult to detect, especially when the detection hardware has to be implemented on chip. Higher levels of abstraction are usually required. The discussion in Section 3.3 highlights the fact that too high an abstraction level, such as at functional or gate level, may not give an accurate representation of physical faults, although it greatly simplifies the treatment of faults, whereas a very low level of abstraction may be too complex for present day VLSI chips. For CMOS technology, switch level modelling seems the most appropriate.

In Section 2.4, a clear distinction is made between on-line and off-line fault detection approaches. It is clear that on-line error detection is the 'ideal' approach if it can be implemented effectively. However, as discussed in Section 2.4, this does not seem to be the case: on-line error detection techniques suffer from high overhead requirements and low coverage of physical faults. The importance of fault detection, even outside the framework of fault-tolerance, is also stressed in Section 2.4.

Another clear distinction is made in Section 2.5 between defect-tolerance and failure-tolerance, since the former deals primarily with manufacturing defects whereas the latter has the capability of dealing with both manufacturing defects and operational failures. In Section 2.6, a brief introduction to the main criteria for assessing the effectiveness of fault-tolerance strategies, namely, yield and reliability, is presented. This subject will be addressed again in Chapter 5.

Chapter 3
Test Generation for CMOS Circuits

3.1 INTRODUCTION

Fault-detection was described as the cornerstone of fault-tolerance in Chapter 2. The knowledge of the presence/absence of faults can be as important as the continuous provision of correct output.

An off-line fault-detection approach is adopted in this research because it achieves a higher fault coverage and requires less hardware than on-line error detection. A major aspect of off-line fault-detection is the generation of test patterns, which is a computationally intensive task. The first step in generating tests is to establish the list of faults that are to be considered. The simple line stuck-at fault model is inadequate for CMOS VLSI circuits. Transistor stuck-open faults, on the other hand, make test pattern generation even harder. However, this type of fault must be taken into consideration, especially as CMOS becomes the dominant technology for high density integrated circuits. The U.S. Department of Defense already requires a 75% coverage of stuck-open faults from its suppliers [162]. The arguments given in [53, 52] for dismissing stuck-open faults are believed to be invalid. If there is a need to ascertain the importance, in terms of frequency of occurrence, of stuck-open faults then investigations as reported in [163] are more appropriate.

In this Chapter, we attempt to derive some simple but complete test generation procedures for CMOS combinational circuits. In Section 3.2 tests are derived for a small combinational circuit by considering each possible physical fault and its effect on circuit operation. This will show that, indeed, a large proportion of physical faults do result in CMOS stuck-open faults. In addition, the difficulties in detecting certain faults and the limitations of the derived tests are also discussed.

The complete test sequence obtained from the exercise of Section 3.2 has a a very useful characteristic: **The fault free response is a trivial signal that can be generated by simple on-chip circuits.**

In Section 3.3, it is shown that tests derived for stuck-open faults are also able to detect all other detectable faults. This result is used in Section 3.4 to develop efficient test generation procedures for stuck-open faults that will also detect all other detectable faults. The distinguishing features of these procedures are:

- Fault enumeration is not required.
- The input to the procedures consists of a relatively high level description of the circuit.

The main aim in deriving the test generation procedures, is that the area overhead associated with the on-chip implementation of test sequence generation and test response analysis should be kept as small as possible. On the test generation side, this requirement implies that the test sequences should be as short as possible. On the response analysis side, it implies that the fault-free response should be trivial, since we do not want to loose the benefit of having a complete test sequence throught the compression of the response into a signature.

3.2 TESTING A CMOS CELL FOR ALL FAULTS

In this section, tests are derived for all faults that may be caused by manufacturing defects and operational failures in a circuit consisting of a 1-bit full adder which is used as example. This is done by considering every fault individually and analysing its effect on the operation of the circuit. This is clearly far from being an efficient method for generating tests, especially as in practice more complex circuits have to be considered. This exercise is carried out solely in order to get some understanding of the requirements for the detection of faults, and to highlight some general observations and conclusions which can be used subsequently to derive more efficient test procedures.

3.2.1 The Fault List

Manufacturing defects can be described, in general terms, as consisting of missing and extra patterns on all layers of an IC. An examination of Tables 2.1 and 2.2 shows that wherever the missing or extra pattern occurs, the finished chip will exhibit either a break in a planned continuous pattern, or an unwanted short between patterns. Breaks can happen only on the same layer, although a missing contact window can be considered as a break between two different layers. Shorts, on the other hand, can happen in a same layer between close patterns, or between different layers when these layers overlap. Many operational failures also result in open and short circuits.

Another type of fault, called *parametric faults*, may also occur as a result of variations in process parameters or operational failures. Threshold voltage shifts, excessive leakage currents, variations in the resistance of different conductors, are examples of parametric faults. These faults are difficult to consider individually, since the threshold voltage of a transistor, or any of the other parameters, may vary by any amount. In practice, parametric faults are tested by exercising the chip with a subset of the complete test sequence at or near the maximum operating frequency. In a BIST environment, the whole test sequence is applied at the operating frequency. Hence, there is no need for the explicit consideration of these faults. The faults of interest are therefore:

1. Open circuits in diffusion layer.
2. Shorts between neighbouring diffusion regions.
3. Open circuits in polysilicon.
4. Shorts between physically close polysilicon tracks.
5. Open circuits in the metal layer(s).
6. Shorts between close metal tracks.
7. Shorts between overlapping metal and polysilicon.
8. Shorts between overlapping metal and diffusion.
9. Shorts between overlapping metal 1 and 2.

A short between poly and diffusion is ~~impossible~~ unlikely in a standard CMOS process because polysilicon and diffusion never overlap, except in the gate regions where the polysilicon layer is used as a mask when diffusing the active regions. Pinholes in the gate oxide, which result in shorts between the gate polysilicon and the channel regions, have been shown to be a reliability hazard [164], and will be discussed later.

In order to derive tests for any of the above faults, it is necessary to consider their effect on circuit operation. The 1-bit full adder is implemented as two complex gates, one for generating the sum output and the other for generating the carry output, together with inverters at the inputs, as shown in Fig. 3.1.

A circuit schematic, as in Fig. 3.1, does not allow the determination of the effects of the physical faults listed previously, because it does not indicate what conducting layers are used for the different interconnections, and it does not show the spatial distribution of the different regions. Hence, a layout of the circuit is required. It should

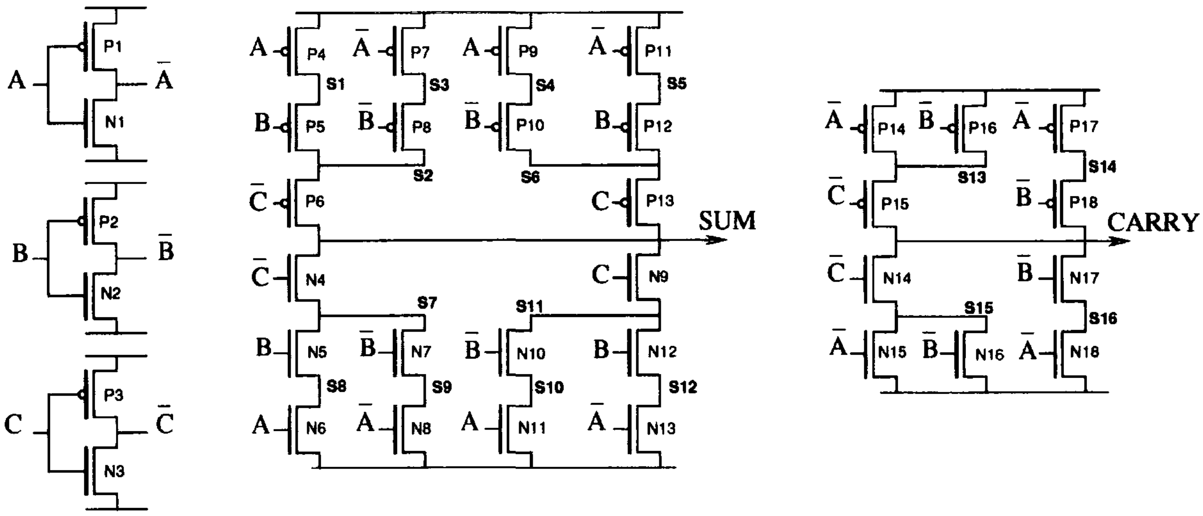


Figure 3.1 A one bit full adder cell.

be pointed out that the number, location and nature of the faults to be considered are strongly dependent on the layout style adopted. Figure 3.2 shows a possible layout of the circuit in Fig. 3.1. This is not the most area-efficient layout, but is nevertheless used here primarily to get some insight into the effects of the physical faults upon circuit operation.

3.2.2 Test Generation for Specific Faults

The complete test derivation for the full-adder cell, although a small circuit, is a very lengthy and repetitive process. It is described in Appendix A. In this section, we consider only a sample of the possible faults in order to explain the test derivation process, and also to state some problems and limitations of the derived test vectors. Subsequent sections elaborate on these problems and limitations. The faults to be considered are illustrated in Fig. 3.3.

Open bdp1: Break 1 in the diffusion layer of the pull-up network causes the source of transistor P1 in the input inverter to be disconnected from VDD, which makes it impossible to drive node $\overline{A}$ to logic 1. Therefore, the output of the inverter is effectively stuck-at zero. The only input combinations that may detect this fault are those that set input A to zero, and an analysis of the circuit response to these inputs gives the following:

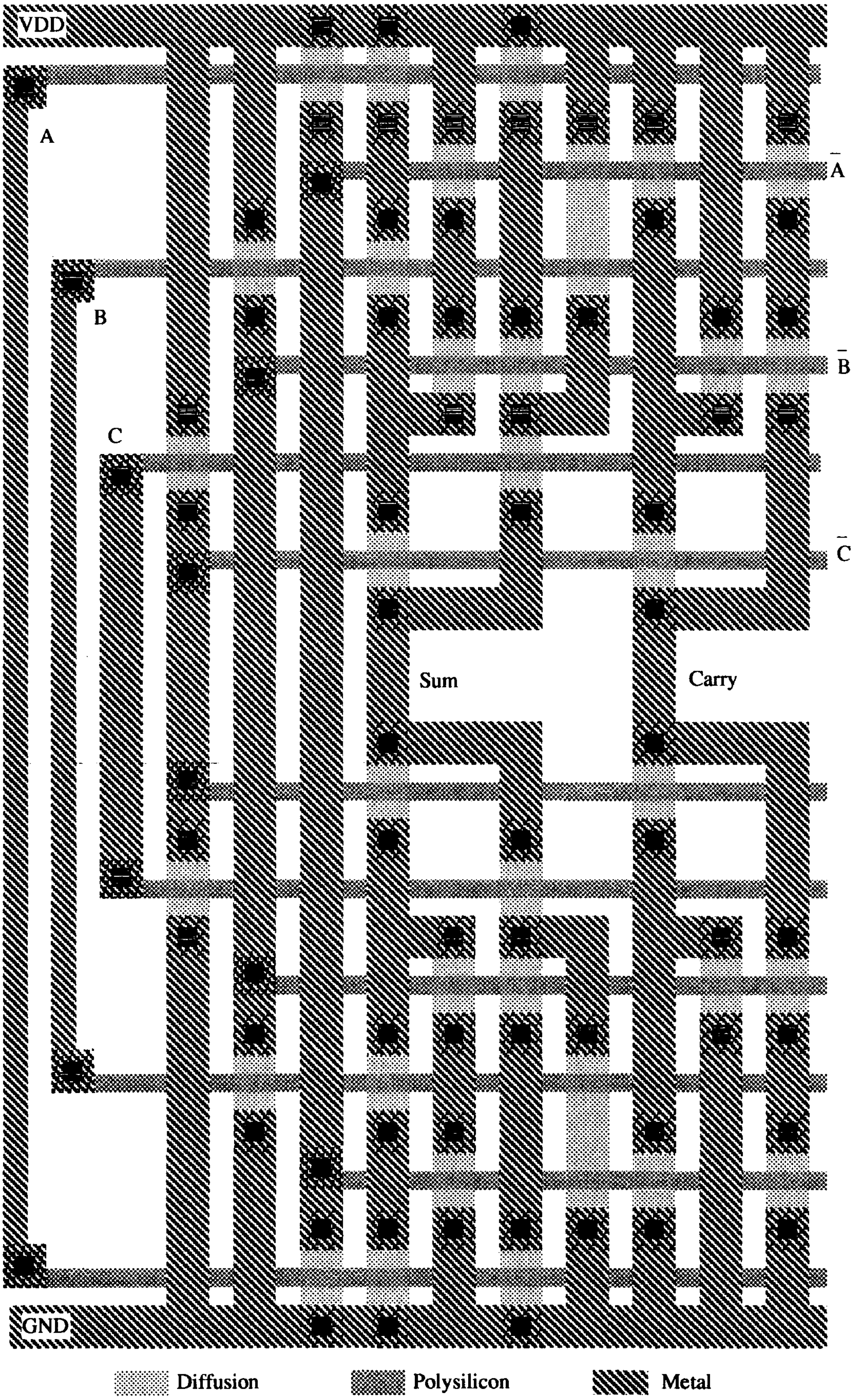


Figure 3.2 Layout of a one bit full adder cell.

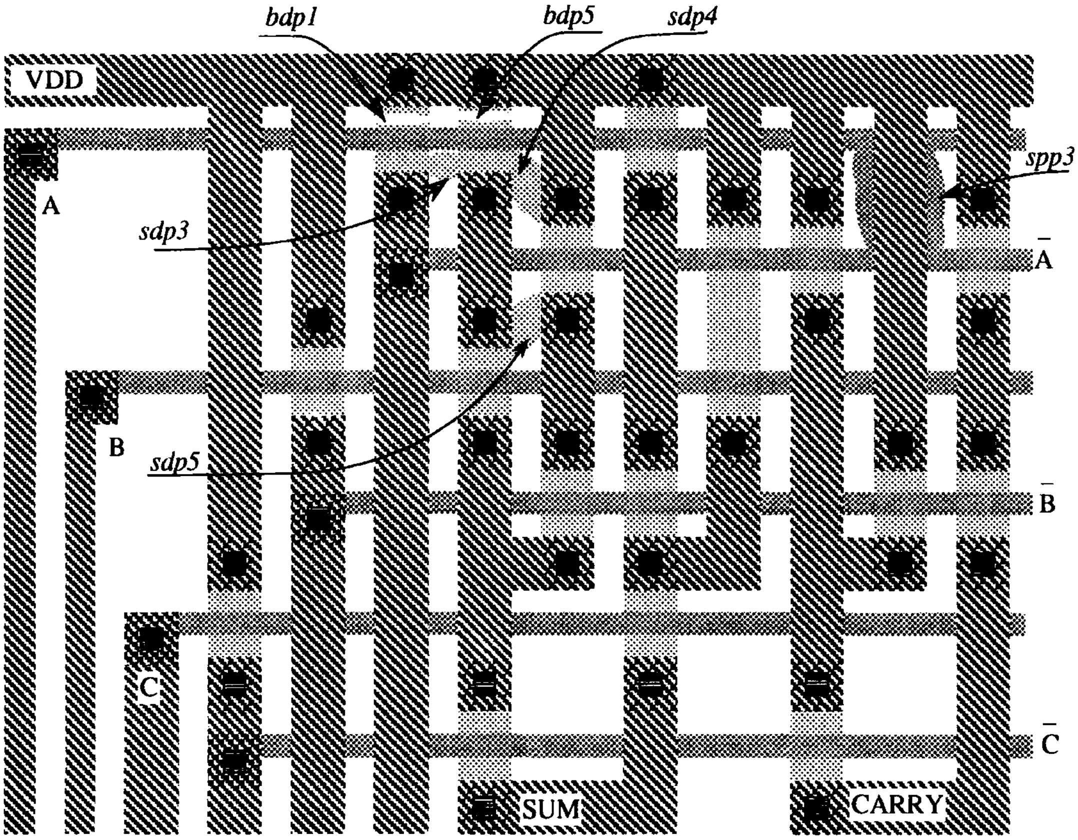


Figure 3.3 Locations and nature of the sample faults.

A	B	C	S		Co	
0	0	0	1	*	0	
0	0	1	1		1	*
0	1	0	1		1	*
0	1	1	1	*	1	

(* indicates a faulty output)

An examination of this table reveals that input vectors ABC = 001 and ABC = 011 produce an incorrect value at the sum output, and that input vectors ABC = 001 and ABC = 010 produce an incorrect value at the carry output, when the fault is present. Therefore, any of the four input combinations can be used as a test vector. The four possible test vectors are denoted as $ABC = 000(S), ABC = 001(C), ABC = 010(C), ABC = 011(S)$ where the symbol in brackets refers to the output at which the fault is detected.

Open bdp5: Break 5 in the diffusion layer of the pull-up network causes the source of transistor P4 to be disconnected from VDD, making it impossible to charge the sum output through the path {P4 P5 P6}. Since there are other paths from VDD to the sum output, this will not result in a stuck-at zero fault, as in the previous case. The response of the circuit under this fault is shown on the right. Z denotes a high impedance state. This occurs when both pull-up and pull-down networks are off, isolating the output node from both VDD and GND. In this case, the output holds its previous logic state, in the form of a charge on the output node capacitance.

A	B	C	S	
0	0	0	0	
0	0	1	Z	*
0	1	0	1	
0	1	1	0	
1	0	0	1	
1	0	1	0	
1	1	0	0	
1	1	1	1	

This fault prevents the output from being charged through the path consisting of transistors P4, P5 and P6, as said before. Hence, the first requirement to test for the presence of this fault is to attempt to charge the output through the path {P4 P5 P6}. However, if the output is already charged to logic 1, then when attempting to charge it again through the path {P4 P5 P6}, it will not be possible to state whether the fault is present or not, since if the fault is present the output will be high (retaining its previous value) which is the same as when the fault is not present. Therefore, the second requirement is that before we attempt to charge the output through the path containing the fault, we must make sure that the output is discharged first, so that if the fault is present, the output will be low when attempting to charge it through path {P4 P5 P6}. In other words, this type of fault requires a pair of vectors for its detection, the first vector being an initialisation one and the second, the test vector.

For the fault under consideration, the initialisation vector can be any input combination that sets the output to logic 0, and the test vector must activate the path {P4 P5 P6}. The only vector that activates this path is $ABC = 001$. The pair of vectors is denoted $ABC = (d, 001)(S)$ where the d stands for any input combination that discharges the output.

The notation $(d, 001)(S)$ represents four possible test pairs (since there are four ways to discharge the sum output). These test pairs are (000, 001), (011, 001), (101, 001), and (110, 001). Consider the last test pair. The initialisation input 110 sets S to zero, and then the test input 001 is applied to the circuit. In a fault-free circuit, the output should go to 1. However, if the fault is present, there is no way to charge the output, since input 001 activates path {P4 P5 P6} only, and hence, the output will retain its previous low value, which is different from the fault-free response.

Now suppose that when changing the inputs from ABC = 110 to ABC = 001 some circuit delays cause input signals A and B to be slow to fall. This will result in the transition ABC = 110 → 111 → 001 (assuming that A and B are delayed by the same amount). If the transient state ABC = 111 persists long enough to charge the output, then when ABC = 001 the output would already be at logic one, violating the requirement that, for the detection of the fault, the output must be low before the application of the test input. Therefore, the fault would go undetected.

This problem of test invalidation by circuit delays has received a wide attention in the literature in recent years. Similar invalidation of tests for stuck-open faults is also caused by charge sharing. This topic is discussed in more detail in a later section. Note that if any of the other possible test pair is selected, then the problem does not occur. This is primarily because for all three test pairs, only a single input is changed in the transition from the initialisation input to the test input, and hence, no other input vector can appear in the transition, regardless of circuit delays.

Short sdp3: Short 3 in the diffusion layer of the pull-up network causes nodes $\overline{A}$ and S1 of the sum circuit to be shorted. In general, to detect a short between two nodes, they must be driven to different values. If we set A to zero, $\overline{A}$ would be at logic one, and node S1 would also be at logic one. Hence, we must set A to one. Since this setting will cause transistor P4 to be off, node S1 will have the value of node $\overline{A}$, namely 0 (which it never gets in a fault-free circuit). Therefore, one way to detect this fault is to propagate this erroneous state of node S1 to the sum output. To do this, we must set inputs B and C to 0 and 1, respectively, resulting in the input vector ABC = 101. However, with this input the sum output will be low anyway, masking the fault-effect. Hence this fault is not detectable since it does not affect circuit operation.

In a gate level representation of a combinational circuit, if a stuck-at 1 or 0 fault is undetectable, the circuit is said to be redundant, i.e., some of its gates or signals can be eliminated without changing the function implemented by the circuit. In our case, the undetectability of the fault under consideration can also be related to circuit redundancy. The fault is undetectable because nodes $\overline{A}$ and S1 are 'equivalent'. By equivalent, it is meant that if $\overline{A}$ is at logic 1, then S1 is also at logic 1. On the other hand, if $\overline{A}$ is at logic 0 then the value of S1 can be anything, and more importantly, this value does not affect the circuit output. Another way to relate redundancy to the undetectability of the fault is by noticing that transistors P1 and P4 have common gate and source signals, and hence they can be merged, which means that one of the two transistors is redundant. In fact, considering all the faults of this type that are undetectable (Tables A.3 and A.4 of Appendix A) will reveal more transistors that can be merged. After merging all these transistors, the resulting circuit would be as shown in Fig. 3.4.

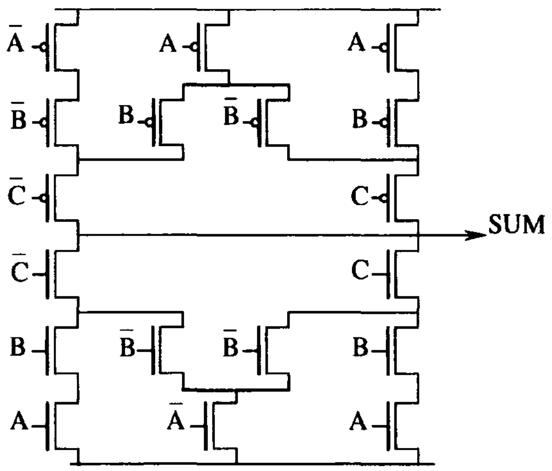


Figure 3.4 Sum circuit resulting from the merging of transistors.

This way of reducing the number of transistors, by sharing or merging, is suggested in at least one of the textbooks on CMOS circuits [165]. However, it is questionable whether such a minimisation would result in any gains in silicon area, since the interconnections are more complicated and they overlap a lot in the reduced circuit.

Short sdp4: Short 4 between the two active regions of the pull-up network connects node S1, the drain of transistor P4, to VDD making S1 stuck-at 1. If A is set to 0, node S1 will be at logic 1 regardless of the presence of the fault. We must therefore set A to 1, and in order to propagate the state of node S1 to the sum output, B and C must be set to 0 and 1 respectively, resulting in the input ABC = 101. With such an input combination, the output is discharged through path {N11 N10 N9}. However, because node S1 is at VDD and the pull-up sub-path {P5 P6} is conducting, we are in the situation where both pull-up and pull-down networks are conducting, which violates one of the principles governing the operation of static CMOS gates, that is, only one of the two networks must be conducting in steady state.

This general situation is common to many other faults. A later section considers the subject in more detail. For now, it is sufficient to say that for the fault under consideration, and with input ABC = 101, the sum output will have a voltage that is somewhere between 5V and 0V, which will be referred to as an *intermediate voltage.* Since the fault-free response to input ABC = 101 is zero, if the intermediate voltage is high enough to be interpreted as logic one, the fault is detected. If the intermediate voltage happens to be so close to the fault-free output voltage such that any circuit seeing it as its input would consider it as the same as the fault-free response, then the fault is undetectable.

Note that inducing an intermediate voltage at the gate output is the only way to detect a fault that causes the pull-up and pull-down networks to be conducting at the same time for some input combinations, and that these input combinations are the only possible test vectors for the fault. The detection, or otherwise, of the fault is dependent on many other parameters, as discussed in a later section.

Short sdp5: Short 5 between the two active regions of the pull-up connects node S3, the drain of P7, and node S1, the drain of P4. An easy way to analyse this type of fault is to consider that the additional connection between nodes S1 and S3 adds extra paths from VDD to the sum output. In this case, there are two extra paths created by this fault: {P7 P5 P6} and {P4 P8 P6}. These paths do not exist in the fault-free circuit, and if they are activated (by setting the gate input of each transistor to zero) the pull-up will be ON. Therefore, the test vectors for such a fault are searched among those input combinations that turn one or more of these extra paths ON while also creating a legal path through the pull-down. In this case, activating either of the two extraneous paths would induce an intermediate voltage at the sum output. This achieved by either $ABC = 101(S)$ or $ABC = 011(S)$.

Short spp3: Short 3 between the two polysilicon tracks of the pull-up causes nodes A and $\overline{A}$ to be shorted. Input A is a primary input in our case. However, in a practical situation, input A would be the output of another CMOS gate. Therefore, any input combination would result in nodes A and $\overline{A}$ having an intermediate voltage. Whether this intermediate voltage can be propagated through the sum and/or carry circuit is impossible to say by a simple circuit or layout inspection. Even a circuit simulation (SPICE) can not be relied upon to state whether or not an intermediate voltage at a gate input is propagated to the circuit output, since the outcome is dependent on so many parameters (device size, parasitic capacitances and resistances, ...) that it is not wise to rely on them.

In Appendix A, similar test derivations are carried out for all possible physical faults. A summary is presented in tabular form at the end of the Appendix. Among all physical faults considered, 30% result in stuck-open faults.

3.2.3 Test Invalidation by Circuit Delays

In the previous section, we saw that two-pattern tests may be invalidated by circuit delays, as well as charge sharing. In this section we will consider this problem in more detail, first by studying the example given in Section 3.2.1 and then by reviewing the main publications on the subject.

In the case presented in Section 3.2.1, it was said that if the test pair ABC=(110, 001)(S) was chosen for the detection of break *bdp5*, and if some circuit delays caused input signals A and B to be delayed (by the same amount of time), then the fault will not be detected. This was clearly a very special case. Table 3.1 presents a complete analysis of applying the test-pair ABC=(110, 001)(S) to the sum circuit under all possible circuit delay conditions. The symbols d_A, d_B and d_C represent the delays associated with signals A, B and C, respectively.

Table 3.1 Circuit analysis under all possible delay conditions.

1 delayed signal:			
A delayed	–	110→101→001	No invalidation
B delayed	–	110→011→001	No invalidation
C delayed	–	110→000→001	No invalidation
2 delayed signals:			
A and B	$d_A = d_B$	110→101→001	**Invalidation**
	$d_A < d_B$	110→111→011→001	No invalidation
	$d_A > d_B$	110→111→101→001	No invalidation
A and C	$d_A = d_C$	110→100→001	**Invalidation**
	$d_A < d_C$	110→100→000→001	No invalidation
	$d_A > d_C$	110→100→101→001	No invalidation
B and C	$d_B = d_C$	110→010→001	**Invalidation**
	$d_B < d_C$	110→010→000→001	No invalidation
	$d_B > d_C$	110→010→011→001	No invalidation
3 delayed signals			
	$d_A = b_B = d_C$	110→001	No invalidation
	$d_A > d_B > d_C$	110→111→101→001	”
	$d_A > d_C > d_B$	110→100→101→001	”
	$d_B > d_A > d_C$	110→111→011→001	”
	$d_B > d_C > d_A$	110→010→011→001	”
	$d_C > d_A > d_B$	110→100→000→001	”
	$d_C > d_B > d_A$	110→010→000→001	”

Althought the two-pattern test (110, 001) is invalidated for onl y three out of the

19 possible delay conditions, this problem cannot be dismissed since circuit delays are generally difficult to avoid. An examination of Table 3.1 indicates that even a single gate delay on any two of the three inputs will cause test invalidation. Ensuring that all inputs change values at nearly the same time is difficult to achieve, particularly when the test stimuli is applied from outside the chip.

A two-pattern test that remains valid under arbitrary circuit delays is called a *robust test* [166]. It is shown in [166] that for a stuck-open fault to have a robust test, there must exist two vectors T_0 and T_1 that differ in a single bit position and such that the pair (T_0, T_1) detects the fault. Note that this does not imply that all robust tests must have two adjacent vectors. For example, considering the carry circuit, the pair (111, 001) is a robust test for the faults N17 and N18 stuck-open even though 111 and 001 differ in two positions. The only vectors that may appear in the transition 111 → 001, as a result of circuit delays, are 011 and 101 which are both valid initialisations for the faults considered. A simple way to check for the robustness of a two-pattern test (T_0, T_1) is to replace the bit positions where T_0 and T_1 differ by dashes. If the two vectors differ in n positions, then there are $2^n - 2$ vectors that may appear in the transition $T_0 \rightarrow T_1$. The transition cube $\langle T_0, T_1 \rangle$ is defined as the set of vectors that may appear in the transition $T_0 \rightarrow T_1$ [167]. If any of these $2^n - 2$ vectors is a vertex of the same type as T_1, then under certain delay conditions, the two-pattern test (T_0, T_1) is invalidated.

It was shown in [166, 167] that there exists boolean functions for which many CMOS implementations do not have robust tests for every stuck-open fault. A procedure is presented in [167] to transform a two-level NAND-NAND realisation of such functions into an implementation that has robust tests. The procedure relies on the addition of a control input and some FETs to the original circuit. Figure 3.5(a) shows a circuit where the stuck-open fault on the pFET driven by X3 does not have a robust test. Reddy et. al. [167] transform this circuit into the one shown in Fig. 3.5(b). A more economical circuit transformation is suggested in [168] and is illustrated in Fig. 3.5(c). It is more economical in that the number of extra transistors is fixed at two, whereas in the approach of Reddy et. al. [167], the number of extra transistors depends on the number of faults that do not have robust tests.

A new circuit transformation, that actually reduces the number of transistors of the original circuit, and for which robust tests exist is proposed in Fig. 3.5(d). The pFET driven by X3 is simply eliminated by combining the first level NAND gate that generates X3 with another first level NAND gate resulting in a complex gate that produces X13.

For single level circuits, i.e., complex gates, Jha and Abraham [169] use the fact that a Boolean function can have four different implementations as a complex gate, to

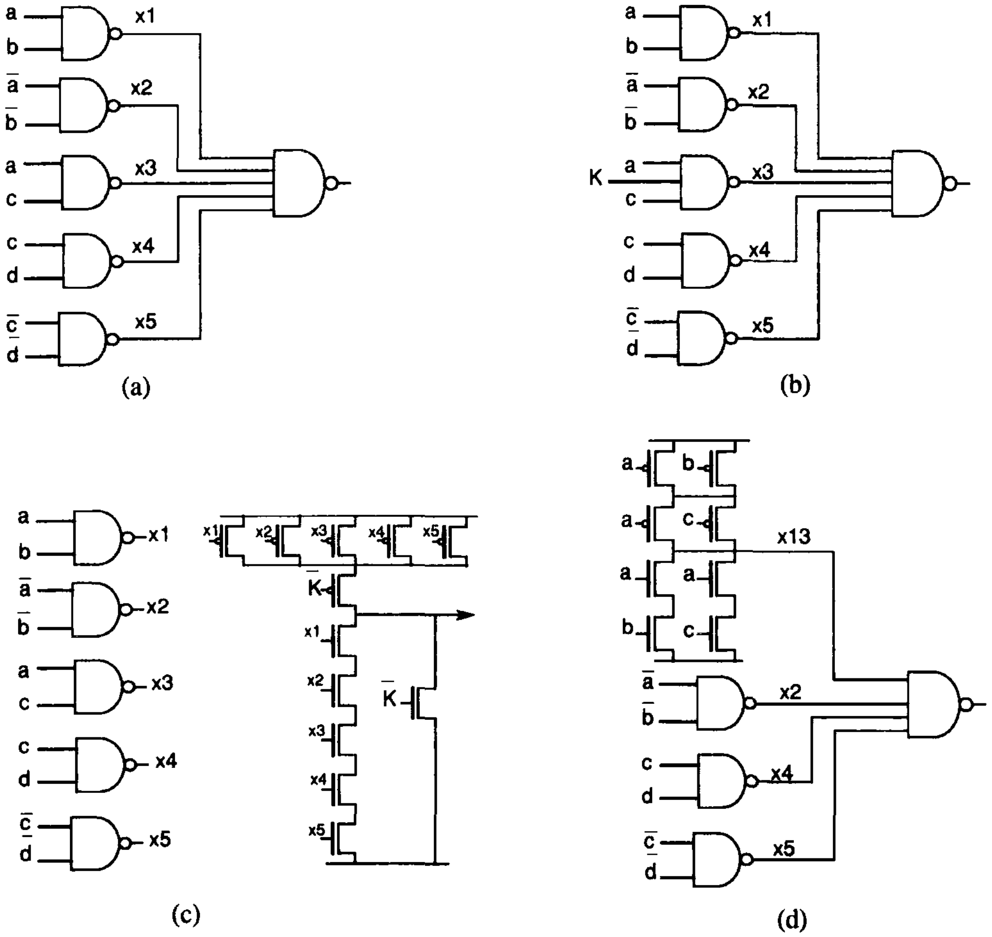


Figure 3.5 Circuit transformations for robust testability. (a) Original circuit, (b) Reddy et. al., (c) Gupta et. al., (d) New approach.

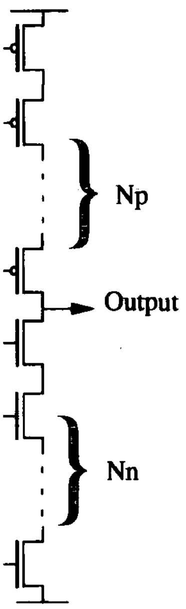
show that for one of the four implementations, called *hybrid* realisation, there always exist robust tests for all stuck-open faults.

In recent publications, another problem associated with circuit delays, namely the appearance of a glitch on the inputs **while test input** T_1 **is applied**, has been introduced [170, 171]. The problem raised in these publications is different from the previous one, and it cannot be solved by appropriate choices of the initialisation vectors. It is a direct consequence of extreme logic sharing and intuitive circuit design. The solution presented in [171] is to convert the static CMOS circuit into a pseudo-nMOS/pMOS circuit during testing. This requires the addition of two transistors to every gate in the circuit. One bonus in doing so is that stuck-open faults are now detected by a single test pattern, as in nMOS circuits. However, contrary to the claims of the authors, this will not result in a 50% reduction in test time: the fact that a single stuck-open fault is detected by a pair of vectors does not imply that the detection of n stuck-open faults requires $2n$ test vectors as an appropriate sequence of $n + 1$ vectors can detect n

stuck-open faults in many cases.

3.2.4 Faults that Induce Intermediate Voltages

In this section we look in more detail at those faults that give rise to intermediate voltages. Most short circuits fall in this category. Intermediate voltages are induced at the output of a gate when both the pull-up and the pull-down networks are conducting. The magnitude of the intermediate voltage is a function of the ratio of the impedances of the networks. This ratio is dependent on the structure of the network, the device dimensions and the voltage on the gates of the transistors. In order to get some insight onto how these parameters determine the magnitude of the intermediate voltage, SPICE simulations were run on circuits where the pull-up and the pull-down networks consist of a series connection of a variable number of transistors with the input conditions set so that both networks are conducting, as shown in Fig. 3.6.



Np	Nn	Output Voltage (V)	
		$W_p = W_n$	$W_p = 2.5W_n$
3	3	0.76	2.28
3	2	0.48	1.36
3	1	< 0.5	0.59
2	3	1.27	3.3
2	2	0.77	2.24
2	1	0.35	0.94
1	1	0.80	2.14
1	2	1.87	3.76
1	3	2.90	4.27

Figure 3.6 Simulation of intermediate voltages.

By inspection of the above results, we may conclude that the network containing the smaller number of transistors dominates the other network. In this case, a stuck-on fault in the network with fewer transistors would induce an intermediate voltage that is far enough from the fault-free values so that subsequent circuitry will view it as a faulty output. But the intermediate voltage induced as a result of a stuck-on fault in the network containing the larger number of transistors will be too close to the fault-free value, and hence the fault is not detectable. An intermediate voltage cannot be propagated through a CMOS gate because the sharp, inverter-like, transfer characteristic restores a strong logic value.

An important consequence of having both pFET and nFET networks conducting is that it violates one of the most sought after characteristic of CMOS circuits: no static current in steady state. This led many researchers to suggest monitoring the steady state current as a way of testing stuck-on faults [172, 173, 174, 175, 176, 177]. The method consists of applying input vectors that induce intermediate voltages in a faulty chip and measuring the current through the chip's VDD or GND pads. The approach is also effective in detecting bridging faults as well as other manufacturing defects, such as pinholes in the gate oxide [164].

The major problem in current monitoring is that the increased quiescent current due to a fault is very small in comparison to the transient currents. The transient currents can be orders of magnitude higher than the steady state current, even in the presence of faults, and it is difficult to supply such high currents through a circuit that can measure very low currents [173, 175]. In addition, most current ATEs do not have the ability to perform current monitoring at clock rates higher than 1MHz [173].

Built-In Current-Testing [176, 177] is a promising approach to alleviate the above problems. In this approach, current sensors are inserted between functional units and their power supplies to monitor the current consumption and provide a pass/fail flag when the current exceeds a certain limit. The current sensors presented in [176] requires a BiCMOS process and the circuitry is of analog type (external voltage reference and a sense amplifier) making the approach slightly ahead of its time (BiCMOS and the mixing of analog and digital circuitry is not yet as widespread as it should be).

Gupta et. al. [168] are the only researchers to propose a method for detecting stuck-on faults by observing logic levels, instead of monitoring current or voltage levels. They propose to augment every gate in the circuit with two control inputs and two transistors as shown in Fig. 3.7. A two-pattern test is required to detect a stuck-on fault in the circuit of Fig. 3.7. Assuming that the fault is in the pull-down network, the first vector sets the gate output to logic 1 by applying an input combination that would induce an intermediate voltage at the output of the original circuit, but sets C1 and C2 to 0, to prevent the appearance of the intermediate voltage in the augmented circuit. The second vector simply sets C1 and C2 to 1. In this case, if the fault is present the output would be low, whereas in the fault-free circuit, the output assumes a high impedance state,

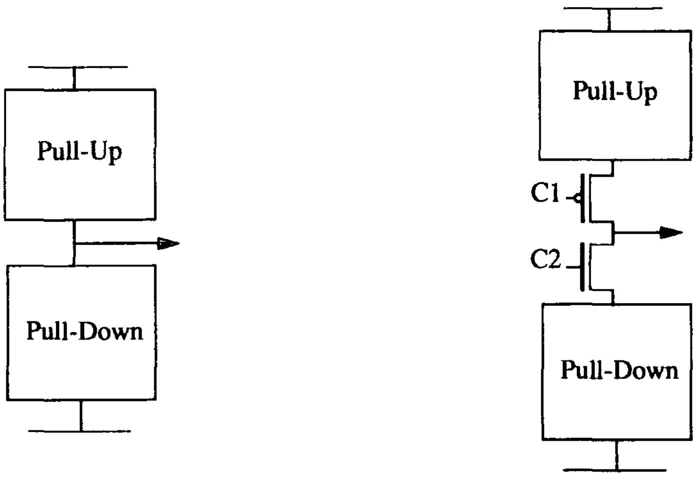


Figure 3.7 Circuit transformation for the detection of stuck-on faults as logical faults.

retaining its previous value, i.e., a logic 1. In a two level circuit, the detection of a stuck-on fault may require as many as four test vectors.

3.2.5 Complete Test Sequence

The test vectors derived in the previous section, and summarised at the end of Appendix A, are now concatenated into a single sequence that covers all the detectable faults and is as short as possible. Because of the large number of faults, and the fact that many of the faults have more than one possible test vector(s), we will take some short-cuts in constructing the test sequence.

First, let us consider the set of faults that require one vector for their detection, as opposed to those requiring a pair of vectors. Among these faults, consider the ones that have only a single possible test vector, as opposed to the those having a number of possible test vectors. The test vectors for these faults must appear in the test sequence since there is no other way to detect these faults. In addition, these vectors do not have to appear in any particular order; their presence in the test sequence is sufficient for the detection of the faults. If we construct the set of such vectors, we will notice that it comprises all possible input combinations for the circuit under consideration, therefore, we can ignore all faults that do not require a sequence of two vectors in this case, as long as the final test sequence contains all input combinations.

Next consider the faults that require a pair of vectors for their detection. Among these faults, consider the ones for which there is only one possible pair of vectors necessary for their detection. These test vectors must appear in the test sequence preceded by the proper initialisation. The sum circuit requires that all possible input combinations appear in the test sequence preceded by the appropriate initialisation vector, whereas the carry circuit requires only six, out of the eight possible, input vectors (vectors 111 and 000 are not necessary for detecting faults in the carry wircuit). These vectors must be preceded by the appropriate initialisation, that is, if the test input vector discharges the output then it must be preceded by an input vector that charges the output, and vice versa.

In order to make the test sequence as short as possible, it would be beneficial if we could use the initialisation vector of a particular fault as the test vector for another fault. That is, if fault f_i and f_j require the test pairs (T_{0i}, T_{1i}) and (T_{0j}, T_{1j}), respectively, for their detection, and if $T_{1i} = T_{0j}$ then the sequence (T_{0i}, T_{1i}, T_{1j}) detects both faults. If circuit delays are ignored, the initialisation vector is only constrained by the fact that it must charge or discharge the particular output. Hence, this optimisation is used for all faults. Under these requirements, we obtain the test sequence shown below.

	A	B	C	SUM	$CARRY$
1	0	0	1	1	0
2	0	1	1	0	1
3	0	1	0	1	0
4	1	1	0	0	1
5	1	0	0	1	0
6	1	0	1	0	1
7	0	0	1	1	0
8	0	0	0	0	0
9	1	1	1	1	1

Note that input vector 001 appears twice in the test sequence since we assume that the outputs at the start of the test sequence can have any value. If it was not repeated, and it happens that at the start of the test sequence $SUM = 1$ and $CARRY = 0$, then all faults that require input $ABC = 001$ as the second vector of a pair would go undetected.

If we want the test sequence to remain valid under any circuit delay condition, then the choice of the initialisation input is more constrained than in the above case. A single bit change between consecutive vectors is necessary for the robustness of tests in the case of the sum circuit. However, for the carry circuit, the single bit change is not always necessary. Taking this into account yields the test sequence shown ~~on the~~ below, Vectors 001 and 011 are repeated to satisfy the single bit change condition in the transition from 111 to 000. The repetition of vector 001 is also necessary for the same reason as in the previous test sequence.

	A	B	C	SUM	$CARRY$
1	0	0	1	1	0
2	0	1	1	0	1
3	0	1	0	1	0
4	1	1	0	0	1
5	1	0	0	1	0
6	1	0	1	0	1
7	1	1	1	1	1
8	0	1	1	0	1
9	0	0	1	1	0
10	0	0	0	0	0

An important attribute of the previous complete test sequences is that the fault-free response of the sum circuit is an alternating signal. Any physical fault would result in at least one of the 1 → 0 or 0 → 1 transitions at the output being absent. This is a direct consequence of minimising the length of the test sequence.

This suggests an interesting new solution to the problem of test response analysis in a BIST environment, namely, the complete test sequence should be derived such that the response of the fault-free circuit is trivial, i.e., the fault-free response can be generated by simple on-chip circuitry. The suggested approach does not suffer from any loss in fault coverage, unlike signature analysis, and all other methods that rely on data compression. Furthermore, toggling circuit nodes has been suggested in a number of publications as a way to increase the coverage of many types of faults [33, 27]. The test sequences presented toggle circuit nodes as often as it is possible.

3.3 STUCK-OPEN FAULTS AS A REPRESENTATIVE OF ALL OTHER FAULTS

In this section, it is shown that a complete test set for stuck-open faults detects all other detectable faults and that the converse is not true. This will be used in Section 3.4 to derive test sequences for stuck-open faults that will also detect all other faults.

3.3.1 Stuck-at Faults

Only a fraction of stuck-open and stuck-on faults manifest themselves as stuck-at faults. Specifically, only some of the stuck-open faults in the class of CMOS gates shown in Fig. 3.8 result in the gate output being stuck-at 1 or 0. In Fig. 3.8(a) a stuck-open fault in any of $P_1, P_2, \ldots, P_n$ would result in the gate output being disconnected from VDD, therefore, the gate output will never be high. It is stuck-at zero, assuming that the gate inputs are sufficiently exercised. Similarly, any of $N_1, N_2, \ldots, N_n$ stuck-open faults in Fig. 3.8(b) would result in a stuck-at 1 fault on the gate output. Transistor stuck-on faults on $N_1, N_2, \ldots, N_n$ in Fig. 3.8(a) are likely to result in a stuck-at 0 fault, whereas $P_1, P_2, \ldots, P_n$ stuck-on, in Fig. 3.8(b), are likely to result in a stuck-at 1 fault.

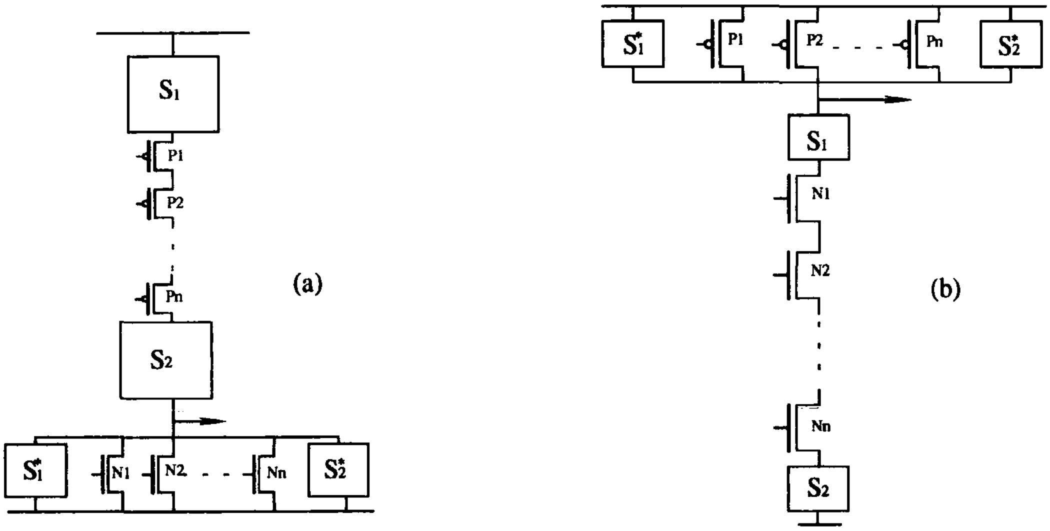


Figure 3.8 CMOS gates where stuck-open faults result in stuck-at faults.

However, the generic CMOS gate presented in Fig. 3.8 is only a special case. A more general CMOS gate is shown in Fig. 3.9, where blocks n_1, n_2 and n_3 can be any network of zero or more transistors, and n_1^*, n_2^* and n_3^* are their respective duals. In the case where a network contains no transistors then it represents either a connection or no connection. For example, if n_3 in Fig. 3.9 is an empty network, then this means that there is nothing in parallel with transistors $P_1, P_2, \ldots, P_n$, whereas if n_1 is empty, then the source of P_1 is connected to VDD. Note that Fig. 3.9 reduces to Fig. 3.8 when network n_3 is empty.

Figure 3.9 shows that, unless n_3 is empty, a stuck-open fault on any of the transistors of interest does not result in a stuck-at fault. Therefore, a complete test set for stuck-at faults may detect only a fraction of the stuck-open faults. A complete test set for stuck-open faults, on the other hand, detects all stuck-at faults, as discussed below.

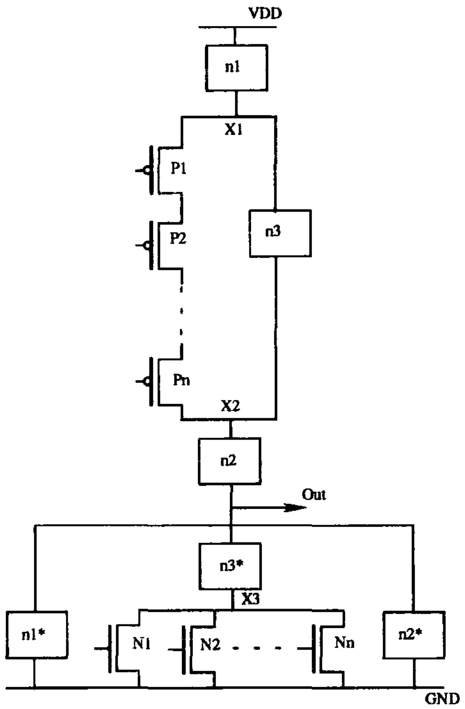


Figure 3.9 General CMOS gate when the focus is on P_i, $i = 1 \ldots n$.

A stuck-at 1 fault on the output of a gate is detected by a test for a stuck-open fault on any of the nFET transistors of the gate. Similarly, a stuck-at 0 on the gate output is detected by a test for a stuck-open fault on any of the pFET transistors of the gate. A stuck-at 1 on an input to a gate is detected by a test for a stuck-open fault on a pFET driven by that input. A stuck-at zero fault on an input is detected by a test for a stuck-open on a nFET transistor driven by that input.

3.3.2 Stuck-on Faults

In this section, it is shown that stuck-on faults are detected by tests for stuck-open faults. The second vector of a pair that detects a stuck-open fault is referred to as the test vector in what follows.

Considering Fig. 3.9, the test vector for P_i stuck-open, $1 \leq i \leq n$, requires that networks n_1 and n_2 be on while n_3 must be off and the gate of P_i must be at 0. A test for $N_i, 1 \leq i \leq n$, stuck-on requires that networks n_1^* and n_2^* be off while n_3^* must be on and the gate of N_i must be at 0. Since P_i and N_i have the same gate signal and n_i on $\Longleftrightarrow$ n_i^* off, n_i off $\Longleftrightarrow$ n_i^* on, it is clear that the test vector for P_i stuck-open is the same as the test for N_i stuck-on. Similarly, the test vector for N_i stuck-open is the same as the test for P_i stuck-on.

3.3.3 Bridging Faults

When considering CMOS circuits at the transistor level, there are two possible types of bridging fault: intra-gate and inter-gate shorts. First, let us consider intra-gate bridging faults (shorts between the internal nodes of a gate) using the general CMOS gate of Fig. 3.9. Table 3.2 is a list of these faults and the corresponding stuck-on faults which have the same tests. Since stuck-on faults are detected by tests for stuck-open faults, it follows that intra-gate bridging faults are also detected by tests for stuck-open faults.

Table 3.2 Intra-gate bridging faults and their corresponding stuck-on faults.

Node 1	Node 2	Corresponding stuck-on faults
VDD	X1	stuck-on faults in n_1
VDD	X2	stuck-on faults in $n_1, n_3, P_1, \ldots, P_n$
X1	X2	stuck-on faults in $n_3, P_1, \ldots, P_n$
X1	Out	stuck-on faults in $n_2, n_3, P_1, \ldots, P_n$
X1	Source of P_i $1 < i \leq n$	stuck-on faults in P_j $1 \leq j < i$
GND	X3	stuck-on faults in $N_1, N_2, \ldots, N_n$
X3	Out	stuck-on faults in n_3^*

Note that the bridging faults are more likely to be detected than their corresponding stuck-on faults, since they tend to reduce the effective number of transistors in the affected network, making it the dominant one, as discussed in Section 3.2.3.

Inter-gate bridging faults are more difficult to consider in a general way. However, the nature of the test sequences generated by the procedures to be described in the following sections responds to the usual requirement for the detection of this type of fault, which is to toggle the circuit nodes as often as possible.

3.4 TEST SEQUENCE GENERATION PROCEDURES

The aim in this section is the derivation of procedures that yield test sequences capable of detecting all detectable faults. It was shown in the previous section that a complete test set for stuck-open faults detects all other faults. Therefore, the procedures proposed in this section are derived by considering stuck-open faults only. In addition, it was observed in Section 3.2 that many stuck-open faults are detected by the same two-pattern tests. Hence, a further aim is to avoid the enumeration of faults, since their number may be very large.

3.4.1 Exhaustive Stuck-Open Fault Testing

The test pattern generation procedure presented in this section is based on the observation, in 3.2.4, that all possible input combinations appear in the test sequence, but in a particular order.

Consider a complex gate implementing a Boolean function $F = f(x_1, x_2, \ldots, x_n)$ where F is the gate output and $x_1, x_2, \ldots, x_n$ its inputs. Let C and D be the sets of true and false vertices of function f, and n_c and n_d their respective number of elements.

If we apply every 1-vertex of f to the complex gate, preceded by a 0-vertex, then all stuck-open faults in the pull-up network would be detected. In the same way, applying every 0-vertex of f, preceded by a 1-vertex, would detect all stuck-open faults in the pull-down network. We can obtain a single test sequence that detects all faults in the complex gate by combining the sub-sequences for pull-up and pull-down, in such a way that every 1-vertex (0-vertex) is used as a test input for some stuck-open faults in the pull-up (pull-down) and also as an initialisation input for some other faults in the pull-down (pull-up). The following procedure constructs such a test sequence, when $n_c \leq n_d$.

Procedure 1:

1) Add an element of C to the test sequence

2) Add an element of D to the test sequence and delete it from D

3) Add an element of C to the test sequence and, if it is not the last one, delete it from C

4) Add an element of D to the test sequence and delete from D

5) Repeat steps 3) and 4) until $D = \emptyset$.

The procedure for the case $n_c > n_d$ is the same as above except that every instance of C is replaced by D, and every instance of D is replaced by C. The length of a test sequence produced by procedure 1 is $2\max(n_c, n_d)$ when $n_c \neq n_d$ and $2\max(n_c, n_d) + 1$ otherwise. For and n-input function, the test sequence length L is such that $2^n \leq L \leq 2^{n+1} - 2$. The procedure presented in the next section generates shorter test sequences.

3.4.2 Minimum Length Test Sequences

An n-variable NAND function has a single false-vertex. Among the $2^n - 1$ true-vertices, only a set of n special true-vertices are useful for the detection of stuck-open faults in the pull-up network, namely, the ones consisting of a single 0 and $n-1$ 1's. The other $2^n - 1 - n$ true-vertices should not be used because they activate more than one path in the pull-up. This fact is used in this section to derive test sequences that are shorter than in the previous section.

The procedure described in this section is the same as procedure 1 except that sets C and D are replaced by C_s and D_s, the sets of *special* true and false vertices of a function. The reduction in test sequence length is due to the fact that sets C_s and D_s have fewer elements than C and D in most cases. The remaining of this section describes how to obtain sets C_s and D_s.

Sets C_s and D_s depend on the particular implementation of a function and, usually, they are not unique. Starting with the minimum sum-of-product form of a function, there are four possible implementations as a single level complex gate and two possible implementations as a two-level NAND-NAND or NOR-NOR gate. The four implementations as a complex gate are described below [169, 167].

1) SP-PS: The transfer function of the pull-up network is the sum-of-product (SP) form of the function whereas the pull-down network implements the product-of-sum (PS) form of $\bar{f}$.

2) PS-SP: The pull-up network implements the PS form of f and the pull-down network is the SP form of $\bar{f}$.

3) SP-SP: The SP forms of f and $\bar{f}$ are used for the pull-up and pull-down networks, respectively.

4) PS-PS: The PS forms of f and $\bar{f}$ are used in the pull-up and pull-down networks, respectively.

If a network (pull-up or pull-down) is implemented according to the SP form of a function (f or $\bar{f}$) then the corresponding set of special vertices (C_s or D_s) can be easily identified from the Karnaugh map of f or $\bar{f}$. Every grouping of 1's in the K-map, or a loop, corresponds to a prime implicant of the function. The essential prime implicants are those that correspond to 1-groupings which contain at least one minterm that is not contained in any other prime implicant loop. The special vertices of the network under consideration are obtained by considering each loop used in the SP form and selecting one minterm that is not included in any other loop. For example, if the K-map shown in Fig. 3.10 is that of f, then the set of special 1-vertices in the SP implementation of the pull-up network is derived as follows.

f wx yz	00	01	11	10
00	1	1	0	1
01	0	0	1	0
11	0	0	0	0
10	1	1	0	1

$\bar{f}$ wx yz	00	01	11	10
00	0	0	1	0
01	1	1	0	1
11	1	1	1	1
10	0	0	1	0

Figure 3.10 Karnaugh map of the example function.

The function has 3 product terms, and hence, C_s will contain three elements. The first element is clearly $wxyz$ = 1101. In the loop containing the four corner 1's, we can select either $wxyz$ = 1000 or 1010, but not 0000 or 0010, since these are contained in another loop. In the remaining loop, we can select either $wxyz$ = 0100 or 0110. Therefore, we have four possible ways to choose the elements of C_s. These are listed below.

$$C_{s_1} = \{1101, 1000, 0100\} \qquad C_{s_2} = \{1101, 1010, 0100\}$$

$$C_{s_3} = \{1101, 1000, 0110\} \qquad C_{s_4} = \{1101, 1010, 0101\}$$

When a network is implemented according to the PS form of a function, the construction of the set of special vertices is not as simple. The PS implementation of a network consists of a series of parallel branches of transistors. An input combination that makes the network conducting is a special vertex if, and only if, for at least one of the parallel branches, a single transistor is on and all other transistors of the parallel branch are off. The procedure to select a set of special vertices in this case is illustrated with the example function of Fig. 3.10. The pull-down network is assumed to implement the PS form of $\bar{f}$. The set of special 0-vertices D_s is determined by constructing Table 3.3.

The first column of Table 3.3 is a list of all 0-vertices of the function. Each of the remaining major columns corresponds to a sum term in the PS form of $\bar{f}$. The subcolumns of a major column correspond to the literals of the sum term. The table entries are marked by considering each 0-vertex and applying the following rules:

Table 3.3 Procedure to select the special vertices of a PS network.

$wxyz$	$\bar{w}$	$\bar{x}$	y	$\bar{z}$	w	z	x	z
1 0001	√	√				√		√
2 0011	√	√	√			√		√
3 0101	√					√	√	√
4 0111	√		√			√	√	√
5 1100				√	√		√	
6 1110			√	√	√		√	
7 1111			√		√	√	√	√
8 1001		√			√	√		√
9 1011		√	√		√	√		√

Rule1 If input variable a is 1 then all entries at the intersection of a subcolumn labelled a and the row of the current 0-vertex are marked.

Rule2 If input variable a is 0 then all entries at the intersection of a subcolumn labelled $\bar{a}$ and the current 0-vertex are marked.

Rule3 If the network under consideration is a pull-up, then the 1's are changed into 0's and the 0's into 1's in rules 1 and 2.

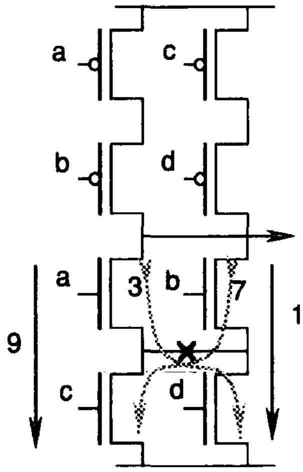
Once the table entries have been marked, the special vertices are obtained by considering each subcolumn and selecting a 0-vertex that marks a single entry in the corresponding major column. If the selected 0-vertex also marks a single entry in another major subcolumn, then the corresponding subcolumn is marked as covered. This will reduce the number of elements in D_s.

The selection of special 0-vertices from Table 3.3 is now illustrated. Considering subcolumn $\bar{w}$, vector 3 is selected since it marks no other subcolumn in the first major column. Subcolumn z in the second major column is also covered by vector 3, so it will not be considered later.

Considering subcolumn $\bar{x}$, vector 8 is selected. Subcolumn z of the third major column is also covered by vector 8. Considering subcolumn y, vector 7 is selected. No other subcolumn is covered by vector 7. Considering subcolumn $\bar{z}$, vector 5 is selected. The remaining subcolumns, w and x are also covered by vector 5, and hence the selection process stops. The resulting minimum set of special 0-vertices is

$$D_s = \{3, 8, 7, 5\} = \{0101, 1100, 1111, 1001\}$$

In general, the above selection procedure yields more than one possible set. The number of elements is an obvious criterion to choose between different sets. A less obvious criterion is illustrated in Fig. 3.11. The two possible sets of special zero vertices are $D_{s_1} = \{7, 3\} = \{1001, 0110\}$ $\quad D_{s_2} = \{9, 1\} = \{1010, 0101\}$



abcd	a	b	c	d
1 0101		√		√
2 0111		√	√	√
3 0110		√	√	
4 1101	√	√		√
5 1111	√	√	√	√
6 1110	√	√	√	
7 1001	√			√
8 1011	√		√	√
9 1010	√		√	

Figure 3.11 Selection of special vertices for the PS form.

The paths activated by the different vectors are indicated on the pull-down of Fig. 3.11. It can be seen that set D_{s_2} does not detect the break indicated by **x**, whereas D_{s_1} does. Note that a change in the labelling of the gates of the transistors reverses the situation. In a large circuit, taking this effect into account will complicate the selection process. It may even be necessary to choose a non-minimal set of special vertices in order to detect breaks similar to the one considered above.

The above procedure for selecting special vertices can also be used with the SP form. The literals in the subcolumns are replaced by product terms and the rules are modified so that a product term is marked only if all its literals are marked. This extension may be necessary when the number of input variables exceeds five or six.

Once the sets of special vertices C_s and D_s have been selected, procedure 1 can be used with C_s and D_s as input, instead of C and D, to yield a shorter test sequence.

Using the example function of Fig. 3.10 and selecting set C_{s_1}, procedure 1 yields the test sequence shown below.

	w	x	y	z
1	1	1	0	1
2	0	1	0	1
3	1	0	0	0
4	1	0	0	1
5	0	1	0	0
6	1	1	1	1
7	1	1	0	1
8	1	1	0	0

Test sequences obtained from the procedure presented in this section may be invalidated by circuit delays. In the next section, a procedure that yields robust test sequences is presented.

3.4.3 Robust Test Sequences

The procedure described in this section yields a complete test sequence that remains valid under all delay conditions. The definitions of special vertices and sets C_s and D_s are as in the previous section. The procedure is as follows:

Procedure 2

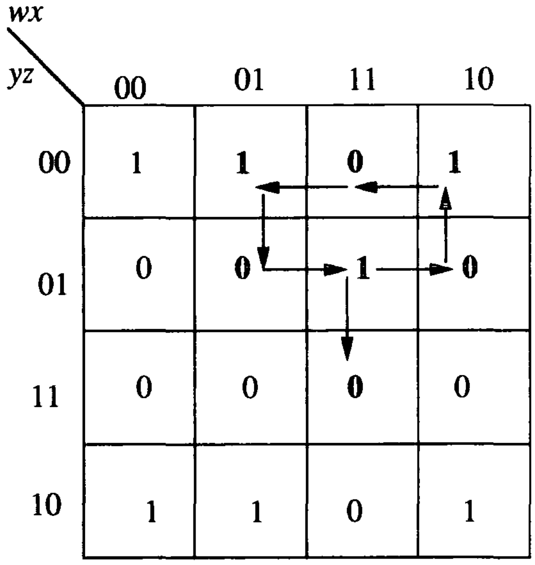
1) Select any special a-vertex ($a \in \{0, 1\}$).

2) Move to an $\bar{a}$-vertex that is adjacent to the previous one.

3) Repeat step 2 until all special vertices are traversed at least once.

An additional requirement is that the starting vector must appear twice in the sequence. Figure 3.12 illustrates procedure 2. The special true and false vertices are highlighted. The directed lines indicate the traversal process. The resulting test sequence is shown on the right.

Another way to state the procedure is as follows: To obtain a robust complete test sequence, construct a path that passes through all selected true and false special vertices at least once, and such that the path is alternatively on 0 and 1 vertices of the function. Ideally, to get the shortest possible test sequence, the path should traverse every vertex



yz \ wx	00	01	11	10
00	1	1	0	1
01	0	0	1	0
11	0	0	0	0
10	1	1	0	1

	w	x	y	z
1	1	1	0	1
2	1	0	0	1
3	1	0	0	0
4	1	1	0	0
5	0	1	0	0
6	0	1	0	1
7	1	1	0	1
8	1	1	1	1

Figure 3.12 Illustration of procedure 2.

no more than once. However, this is not always possible (as in the case where the first vector of the sequence has to be reapplied, for example).

The problem of constructing such paths is very similar to the notoriously complex problem of the Travelling Salesman. This agrees with [178] where it was shown that the problem of detecting stuck-open faults is NP-complete.

The underlying assumption in procedure 2 is that there should be adjacency between consecutive vectors to guarantee test sequence robustness. As mentioned earlier, this is not always necessary. A better procedure, which is more amenable to automation then the previous one, and that does not require adjacency between consecutive vectors, is now presented.

Definition: Two vectors T_0 and T_1 are said to be *compatible* if $f(T_0) = \overline{f(T_1)}$, and the transition cube $\langle T_0, T_1 \rangle$ does not contain any vertex of the same type as T_1.

In other words, T_0 and T_1 are called compatible if they form a robust two-pattern test for some stuck-open fault.

Procedure 3

1) For each element $\mathbf{c}_i$ of C_s, construct the set $D(\mathbf{c}_i)$ consisting of the elements of D_s that are compatible with $\mathbf{c}_i$.

2) For each element $\mathbf{d}_i$ of D_s, construct the set $C(\mathbf{d}_i)$ consisting of the elements of C_s that are compatible with $\mathbf{d}_i$.

3) Add element $\mathbf{c}_i$ to the test sequence,

4) Add an element $\mathbf{d}_i$ from $D(\mathbf{c}_i)$ to the test sequence,

5) Add an element $\mathbf{c}_{i+1}$ from $C(\mathbf{d}_i)$ to the test sequence,

6) Add an element $\mathbf{d}_{i+1}$ from $D(\mathbf{c}_{i+1})$ to the test sequence,

$$\vdots$$

until every special vertex has been used at least once.

The procedure is now illustrated using the example function of Fig. 3.10. Sets C_s and D_s are as follows: $C_s = \{1101, 1000, 0100\}$, $D_s = \{0101, 1001, 1111, 1100\}$

First, we construct the sets of compatible vectors for every special vertex.

$D(1101) = \{0101, 1001, 1111, 1100\}$
$D(1000) = \{1001, 1100\}$
$D(0100) = \{0101, 1100\}$

$C(0101) = \{1101, 0100\}$
$C(1001) = \{1101, 1000\}$
$C(1111) = \{1101\}$
$C(1100) = \{1101, 1000, 0100\}$

Next, the sequence is constructed.

Select an element from C_s: →1101
Select an element from $D(1101)$: →0101
Select an element from $C(0101)$: →0100
Select an element from $D(0100)$: →1100
Select an element from $C(1100)$: →1000
Select an element from $D(1000)$: →1001
Select an element from $C(1001)$: →1101
Select an element from $D(1101)$: →1111

Note that at every step, if the choices were arbitrary then the resulting test sequence may not be the shortest possible one. The following longer test sequence is obtained using the same procedure, but making different choices after step1.

Select an element from C_s:→1101
Select an element from $D(1101)$: →1111
Select an element from $C(1111)$: →1101
Select an element from $D(1101)$: →1100
Select an element from $C(1100)$: →1000
Select an element from $D(1000)$: →1001
Select an element from $C(1001)$: →1101
Select an element from $D(1101)$: →0101
Select an element from $C(0101)$: →0100

No algorithm has been found so far to direct the choice to obtain test sequence of minimal length. However, some helpful observations follow:

Observation 1: The shortest possible test sequence, obtained from procedure 1 with sets C_s and D_S as input has a length of $2\max(n_c, n_d)$. This can be regarded as a lower bound of the length of robust test sequences, i.e., if the length of a test sequence obtained from the procedure of this section is $2\max(n_c, n_d)$, then it is the optimal robust and complete test sequence.

Observation 2: If every special vertex has more than one compatible vector, than it is usually possible to obtain an optimal robust test sequence.

Observation 3: If there exists a special vertex that has no compatible vector, then the implementation of the function cannot be robustly tested. Section 3.2.2 discussed this case.

An alternative, graph-oriented, approach to obtain the complete robust test sequence is illustrated in Fig. 3.13. The vertices of the graph are the elements of C_s and D_s. There is an edge between two vertices if they form a compatible pair. The test sequence is obtained by constructing the shortest path that traverses all vertices following the edges of the graph. The bold line in Fig. 3.13 represents the resulting path.

3.4.4 Test Generation for Multi-Level Circuits

Complex gate implementations of combinational circuits become impractical when the number of input variables exceeds five or six. In such cases multi-level circuits have to be used. However, with the exception of two-level NAND-NAND and two-level

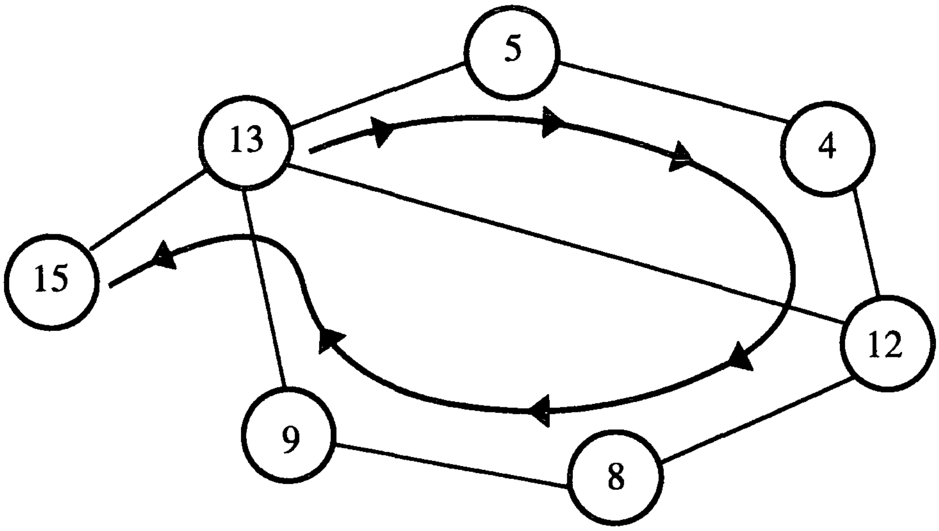


Figure 3.13 Graph-oriented approach for the obtention of a complete, robust test sequence.

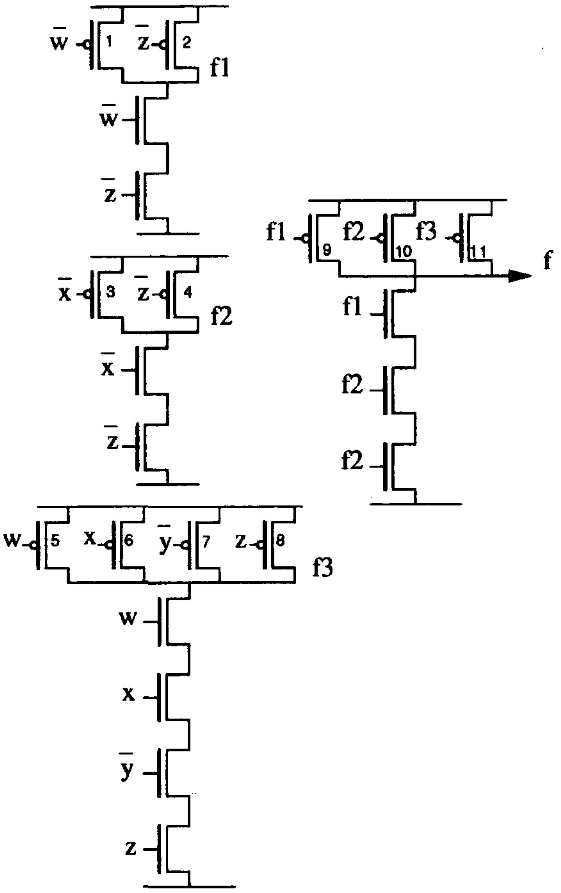
NOR-NOR, there is no systematic method for the implementation of a combinational circuit as a multilevel network of primitive and/or complex gates. Furthermore, if a complex gate implementation exceeds some fan-in limit then the corresponding two-level implementation also exceeds this limit. However, it is much easier, and more systematic, to accomodate the fan-in limit in a two-level circuit (by simply relacing some gates with trees of NAND and/or NOR gates) than in a complex gate.

In the following, we will consider the extension of the test generation procedures to two-level NAND-NAND circuits. The discussion of two-level NOR-NOR circuits is omitted since it is analogous to two-level NAND-NAND. Unstructured multi-level circuits will then be discuused.

The simplest way to extend the procedures described previously is to apply them to each individual gate and then concatenate the resulting test sequences. In cases where the inputs to the gate under consideration are a subset of the inputs to the circuit, the remaining inputs are left unspecified until the concatenation of the sequence where they are set in such a way that the fault effect is propagated to the circuit output.

Indeed, applying this simple procedure to a two-level NAND-NAND circuit reveals that only the first level NAND gates need considering and that the resulting complete test sequence is of minimal length in some cases, whereas in other cases it is possible to further reduce the length of the test sequence, as illustrated in the following example (Fig. 3.14).

Setting the unspecified inputs so that the fault is propagated to the output is straightforward. Note that the initialisation vector for stuck-open faults in the pFETs of first



	w x y z	f1	f2	f3	f	Faults
1	0 - - 0	0	-	1	1	
2	1 1 - 0	1	1	1	0	P1
3	0 1 - 0	0	1	1	1	P9
4	0 - - 1	1	1	1	0	P2
5	- 0 - 0	-	0	-	1	
6	1 1 - 0	1	1	1	0	P3
7	1 0 - 0	1	0	1	1	P10
8	- 0 - 1	1	1	1	0	P4
9	1 1 0 1	1	1	0	1	
10	0 1 0 1	1	1	1	0	P5
11	1 1 0 1	1	1	0	1	P11
12	1 0 0 1	1	1	1	0	P6
13	1 1 0 1	1	1	0	1	
14	1 1 1 1	1	1	1	0	P7
15	1 1 0 1	1	1	0	1	
16	1 1 0 0	1	1	1	0	P8

Figure 3.14 Two-level NAND-NAND implementation of the example circuit.

level gate i is also a test vector for the stuck-open fault on the pFET of the output gate that is driven by the output of gate i. Hence, there is no need for considering the output NAND gate. If gate i has n_i inputs then the initialisation vector is applied n_i times.

The reduction in test sequence length comes from the fact that it is possible to simultaneously test two or more pFETs belonging to different first level gates. This is possible if

- The prime implicant loops corresponding to these gates have a non-empty intersection (the common initialisation vector belongs to the intersection of the loops), and
- the specified inputs of the 0-vertices that detect these faults do not conflict.

Considering the previous example, an examination of the test sequence reveals that vectors 2 and 6 can be combined to test for P1 and P3 simultaneously, and vectors

4 and 8 can be combined to test for stuck-open faults on P2 and P4 simultaneously. However, because the initialisation vector for P2 stuck-open is the test vector for P9 stuck-open, and the initialisation vector for P4 stuck-open is the test vector for P10 stuck-open, testing stuck-open faults on P2 and P4 simultaneously will result in either P9 or P10 not being tested. Therefore, only vectors 2 an 6 can be combined. (The initial test sequence can be reordered so that vectors 4 and 8 can be combined, but then combining vectors 2 and 6 will result in either P9 or P10 not being tested). The resulting test sequence is two vectors shorter. In a larger circuit, with more prime implicants and more literals per prime implicant, the reduction in the length of the test sequence through the simultaneous testing of different pFETs belonging to different first level gates may be more substantial than in this case.

An important attribute of the previous test sequences is that if a two-level circuit is transformed into a multi-level circuit, to accomodate some fan-in limit throught the replacement of single gates with trees of gates, then the resulting multi-level circuit is also completely tested by the same test sequence. Furthermore, and as shown in [167], if the test sequence is robust in the two-level realisation then it remains robust in the multilevel circuit. As an illustration, consider the circuit of the previous example where the 4-input first level NAND gate is replaced by a tree of two 2-input NOR gates and a NAND gate, as shown in Fig. 3.15.

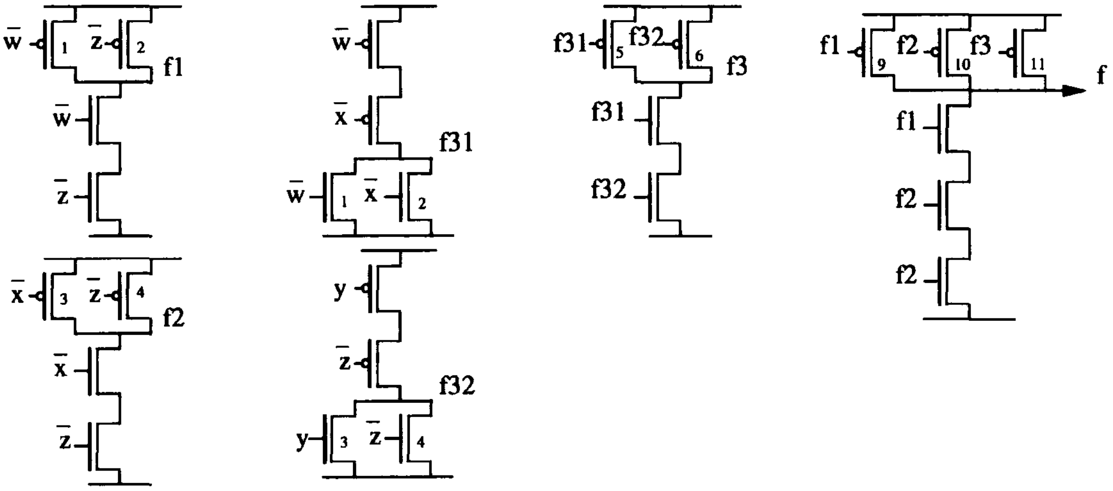


Figure 3.15 Transformation of a two-level NAND-NAND circuit into a multilevel circuit.

Stuck-open faults on transistors N1, N2, N3 and N4 of the two NOR gates are detected by vectors 10, 12, 14 and 16, respectively, of the test sequence for the corresponding two-level NAND-NAND realisation.

For unstructured multi-level circuits, obtained throught factorisation and/or logic sharing, the test sequence derived for the corresponding two-level realisation is generally not complete. The following procedure generates a complete test sequence for multi-level circuits.

Let n be the number of gates in the circuit. Each gate i has its associated sets C_{si} and D_{si}, the sets of special 1 and 0 vertices, as defined in Section 3.4.2. The procedure is as follows:

Procedure 4

1. Derive a test sequence for gate i by combining the elements of C_{si} and D_{si}.
2. Sensitise path(s) from the gate output to the circuit output.
3. Set all the node values implied by the settings in **1.** and **2.**
4. Optimisation 1:

   For each gate $j, j \not= i$,

   - if an element of C_{sj} follows an element of D_{sj} in the test sub-sequence for gate i, then mark this element and sensitise path(s) from the output of gate j to the ciruit output.
   - if an element of D_{sj} follows an element of C_{sj} in the test sub-sequence for gate i, then mark this element and sensitise path(s) from the output of gate j to the ciruit output.
5. Repeat steps **1. 2. 3.** and **4.** until all elements of all C_{si}'s and D_{si}'s are marked.
6. Optimisation 2: Reorder the test subsequences so that:
   - the output is alternating, and
   - if the last input vector of a subsequence is also the first vector of another subsequence, then these vectors can be merged.

Unspecified values of some nodes may be set at step **4.** in order to further optimise the test sequence.

The procedure is illustrated with the circuit of Fig. 3.16 which is a multi-level realisation of the circuit of the previous example. Table 3.4 shows the test derivation process. The node values in bold are obtained at step **1.**. The other node values are obtained by application of steps **2.** to **5.** The complete test sequence obtained from step **6.** consists of the subsequences for gate 4, gate 3, gate 2 and then gate 1, noting that vectors 1 and 8 of Table 3.4 can be merged. This results in a 11-vector long test sequence.

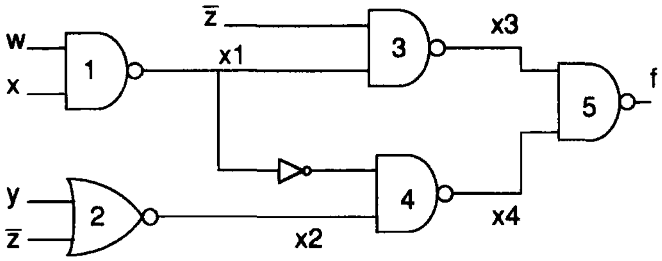


Figure 3.16 Unstructured multi-level realisation of the previous example ciruit.

Table 3.4 Illustration of Procedure 4.

	w x y z	x1	x2	x3	x4	f	Special Vertices
1	**1 1** - -	**0**	0	1	1	0	
2	**0 1** - 0	**1**	0	0	1	1	$(01)_1,(11)_3,(01)_5$
3	**1 1** - 0	**0**	0	1	1	0	$(11)_1,(10)_3,(11)_5$
4	**1 0** - 0	**1**	0	0	1	1	$(10)_1$
5	1 1 **0 1**	0	**1**	0	0	1	
6	1 1 **1 1**	0	**0**	1	1	0	$(10)_2,(10)_4$
7	1 1 **0 1**	0	**1**	1	0	1	$(00)_2,(10)_5,(11)_4$
8	1 1 **0 0**	0	**0**	1	1	0	$(01)_2$
9	- - - **0**	**1**	0	**0**	1	1	
10	- - 0 **1**	**1**	1	**1**	1	0	$(01)_3$
11	1 1 0 1	**0**	**1**	1	**0**	1	
12	- - 0 1	**1**	**1**	1	**1**	0	$(01)_4$

The test sequences for multi-level circuits are always longer than the sequences for the corresponding complex gates. For example, a two-level NAND-NAND implementation of the SUM circuit requires a 24-vector long test sequence, compared with just 9 vectors in the test sequence of the complex gate.

The circuit transformation suggested for robust testability, Fig. 3.5(d) yields shorter test sequences than other multi-level realisations.

3.4.5 Testing Multi-Output Circuits

The major aspect of the test sequence generation procedures presented so far is that they produce a trivial alternating output, thus greatly simplifying the test response analysis, while detecting all detectable faults. The notion of 'alternating output' has no meaning when the circuit has more than one output. It is possible to have an input sequence that toggles all outputs at every cycle but such a sequence will detect only a small fraction of the faults, if any.

On the other hand, an input sequence that detects all possible faults but produces an output response that does not follow any regular pattern is of little use in a BIST environment, since this leaves only two alternatives for response analysis: (i) store the circuit response on-chip, or (ii) compress the test response.

Storing the test response on-chip would require a large area overhead, while the compression of the test response may give rise to alia sing errors, negating the benefits of having a complete test sequence.

One way to restrict the circuit response to a regular pattern would be to test one output at a time. All the faults that can be tested at output i are tested with an input sequence, derived by one of the procedures of the previous sections, i.e., i is an alternating output, then all faults that can be detected at output $i + 1$ are tested in a similar manner, and so on. The circuitry required for analysing such a test response is simpler than in the previous two alternatives. The drawback is a much longer testing time.

The testing time can be reduced if the test sequences corresponding to each output are dervied such that they overlap, as illustrated in Fig. 3.17 for a three output circuit.

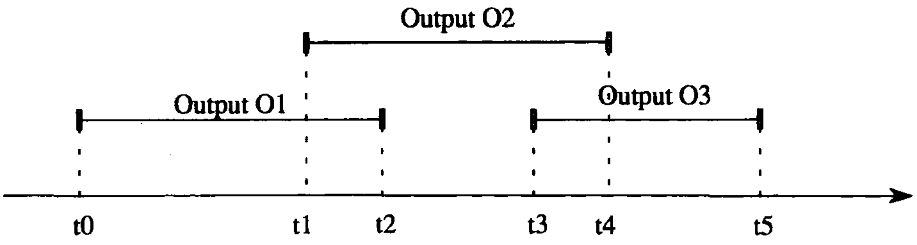


Figure 3.17 Reduction of test sequence length through overlapping.

O1 is an alternating signal between t_0 and t_2. O2 is alternating between t_1 and t_4, while O3 is alternating from t_3 to t_5. The input vectors that detect faults at output O1 only are applied between t_0 and t_1. Similarly, the input vectors that detect faults at output O2 only appear between t_2 and t_3. Input vectors that detect faults at both O1 and O2 appear between t_1 and t_2, where both outputs are alternating. Examples illustrating this approach are presented in the poster section of [179].

The circuitry for test response analysis will have to perform the following functions:

- Enable the monitoring of output O1 at t_0
- Enable the monitoring of output O2 at t_1
- Disable the monitoring of output O1 at t_2
- Enable the monitoring of output O3 at t_3
- Disable the monitoring of output O2 at t_4

Therefore, the circuitry required for test response analysis is more complex than when no overlapping occurs, but the reduction in the length of the test sequence may offset the increased complexity.

Another way to test multi-output circuits is to generalise the idea of a trivial output to the case when there is more than output, i.e., devise a multi-bit signal that can be generated on-chip by simple hardware, and then derive test sequences that would:

1. detect all faults in the circuit, and
2. give a fault-free response equal to this multi-bit signal.

Such a multi-bit signal must have many 0 → 1 and 1 → 0 transitions since the detection of CMOS stuck-open faults requires inducing transitions at certain nodes of the circuit. The output of a binary counter may be appropriate. A further alternative approach for testing multi-output circuits is presented in Chapter 4, where the outputs are combined into a single output.

3.5 CHAPTER SUMMARY

The aim of the work presented in this chapter was to derive simple test generation procedures for the detection of all detectable faults. The laborious task of generating tests by considering every possible physical fault and its effect on circuit operation was undertaken in order to get insights into the requirements for the detection of the different faults. An important result of this task was the derivation of a complete test sequence that produces a toggling fault-free response.

The idea of generating complete test sequences in such a way that the fault-free response of the circuit under test is easily generated by simple on-chip circuitry is proposed in this thesis as a novel and effective solution to the problem of test response analysis, especially for BIST implementations. Currently, signature analysis is the most widely used technique. The proposed approach is superior to signature analysis because it does not suffer from any error aliasing or fault masking effects. The only requirement is the deterministic derivation of test sequences, as opposed to pseudo-random or pseudo-exhaustive sequences. However, this requirement is necessary for achieving high levels of coverage of stuck-at and stuck-open faults. This approach is used in the next chapter to implement built-in self-test with a hardware overhead comparable with current BIST approaches, while achieving a much higher fault coverage.

Tests for stuck-open faults were shown to detect all other detectable faults, thereby obviating the need to consider them for test generation. Simple test sequence generation procedures were derived. Test sequence generation, as opposed to test pattern generation, does not require fault enumeration. This is another advantage of the procedures presented in this chapter. An additional advantage offered by these procedures is that test sequences are derived from a truth table specification of the circuit.

Chapter 4
Built-In Self-Test for CMOS Circuits

4.1 INTRODUCTION

This chapter addresses the problem of designing chips that are able to provide information concerning their internal status (operational/faulty) using an off-line fault detection approach. This is commonly termed Built-In Self-Test (BIST).

The benefits of BIST, even outside the framework of fault-tolerance, are numerous. Most of these benefits stem from the following two factors:

- Unlimited access to internal nodes, at least in theory, and
- the ability to test circuits at their operating frequency.

The limited access to internal nodes of a chip, in external testing, is proving to be a major problem in exploiting the advantages offered by VLSI. The ever decreasing pin-to-gate ratios are limiting the ability of an external tester to control and observe the internal nodes. It was in anticipation of these forthcoming difficulties that the concept of Design For Testability (DFT) appeared in the early seventies, and subsequently evolved to the concept of BIST. BIST is considered as the ultimate DFT technique [109]. It is also seen as the only solution, in conjunction with boundary scan, for the testing of state of the art PCBs populated with surface mounted ICs [78].

The increasing clock frequencies of today's VLSI circuits are also causing problems in testing. The connections between the test fixture and the pins of an IC start behaving as transmission lines above certain frequencies. Taking these effects into account increases the complexity, and therefore the cost, of the Automatic Test Equipment (ATE).

BIST would make it possible to use simpler and less expensive ATE and still allows "at-speed" testing which is not only necessary for checking the chip performance at high clock rates, but also uncovers more faults. System testing and maintenance are also greatly simplified with the use of BIST chips.

In Section 4.2, the design of a built-in self-testing circuit is presented. The principal objective is the detection of all possible faults in the functional circuit. This requires that the tests be derived in a deterministic manner. Furthermore, this requirement excludes signature analysis for test response analysis. These last three points represent significant departures from current BIST techniques for the following reasons:

- The idea behind pseudo-random testing is to use a sequence of pseudo-random test patterns and then evaluate the stuck-at fault coverage. There is no attempt to detect all faults. If the stuck-at fault coverage proves too low, then additional steps are required to bring it up to an acceptable level, such as increasing the number of vectors, modulation of the pseudo-random patterns, or re-design of the circuit under test.

- In pseudo-exhaustive testing all stuck-at faults are detected but there is no provision for the detection of other faults. Furthermore, the reduction of test lengths relies on 'structural' rather than 'functional' partitioning of the circuit.

- For both pseudo-random and pseudo-exhaustive testing, signature analysis is the only option for processing the test response. Alia sing errors effectively reduce the fault coverage.

- One of the acclaimed advantages of pseudo-random and pseudo-exhaustive testing is the elimination of the task of test pattern derivation. However, the tasks of partitioning the CUT, simulating it for obtaining the fault-free signature, fault simulation for assessing the fault coverage, and the measures undertaken for improving it, are all nearly as complex as the task of deriving tests, which pseudo-random and pseudo-exhaustive testing were supposed to eliminate.

The hardware simplicity of the Linear Feedback Shift Registers, LFSRs, used as test pattern generators and signature analyzers, is therefore the only real advantage of current BIST approaches. However, even with the requirement of detecting all faults, it is shown in Section 4.2 that the resulting hardware overhead is comparable to that associated with pseudo-random and pseudo-exhaustive testing. Other design issues, with an effect on the hardware overhead in particular, are also discussed in Section 4.2. These issues are not specific to the chosen example and they can be of relevance in any BIST design.

In Section 4.3, the problem of testing the test circuitry is addressed. Faults in the added hardware may render the whole chip unusable, hence, it is necessary for it to be tested as rigorously as the functional parts. Testing the extra hardware requires the addition of circuits which themselves need to be tested requiring a further addition of circuits that also need testing, . . ., and so on. This seemingly endless process is similar to the situation encountered in self-checking circuits using coding techniques, where the only way to end this infinite loop is to assume that some parts of the chip must be working for checking the working status of the rest of the chip.

For some parts of the test circuitry, it is shown in Section 4.3 that it is possible to avoid the above endless process. For the other parts, examples are given to illustrate the difficulties in achieving *complete test coverage.* These examples suggest that improved fault coverage may be achieved if separate chips were to cooperate in testing each others untested parts.

It was mentioned in Chapter 1 that there might exist an optimum level for introducing fault-tolerance. It is shown in Section 4.4 that the hardware overhead for fault-detection achieves a minimum for a particular partitioning of the circuit under test. This suggest that the size of the added circuitry can be further minimised by selecting an appropriate partitioning scheme.

The results presented in Sections 4.2 and 4.4 assume that the test sequence generator is implemented as a ROM array and a ring counter. This is the simplest possible implementation. In Section 4.5, other implementations are investigated for lower overhead and for better testing of the test circuitry.

In Section 4.6, a time redundancy method is investigated as a *pseudo-on-line* fault-detection approach for CMOS circuits. Section 4.7 concludes the chapter.

4.2 DETECTION OF ALL FAULTS IN A BUILT-IN SELF-TEST IMPLEMENTATION

The implementation of a self-testing circuit is considered in this section. The main goal of detecting all faults is achieved through the use of the test sequence generation procedures of Chapter 3. A ripple carry adder is selected as an example for this implementation. This is a circuit of manageable complexity and its modularity allows for the study of the effect of partitioning on the hardware overhead. The regularity of the adder also allows some insights into the implementation of BIST in a chip consisting of several units.

An n-bit ripple carry adder consists of n full adder cells. The one-bit full adder cell is the same as the one used for the fault analysis of Chapter 3. Each cell is completely tested with a sequence consisting of nine vectors. The simplest method for testing an n-bit adder takes advantage of the modularity of the circuit: The same 9-vector long test sequence is applied to every adder cell and the outputs are monitored. The extra hardware required to implement BIST is listed below.

- Test pattern generator.
- Input multiplexers.
- Analyzer circuit.

As described in this section, the test pattern generator is implemented as a ring counter and a ROM array. In Section 4.5, alternative implementations are investigated. The input multiplexers are used to select between the normal inputs and the test inputs. The analyzer circuit monitors the output response and is responsible for reporting the faulty/operational status of the circuit.

Considering the placement of the analyzer circuit, two strategies emerge. They are discussed in the next subsection. The subsequent sections give the implementation details of all the extra hardware and its placement relative to the adder circuit.

4.2.1 Distributed *vs* Centralised Analyzer

Since the ripple adder is an iterative circuit, we may either test all cells in parallel, or sequentially test one cell at a time. In the first case, every adder cell is associated with extra circuitry to monitor its outputs, as shown in Fig. 4.1. This case will be referred to as *distributed.*

In the second case, a single analyzer circuit is provided for the whole n-bit adder, as shown in Fig. 4.2. The test sequence is applied to one adder cell at a time, the output of which is monitored by the analyzer. This case will be referred to as *centralised.*

As far as test time is concerned, the distributed approach is clearly superior: it is n times faster than the centralised approach. However, the situation for the hardware requirements needs a more detailed analysis. The test pattern generator and the input multiplexers are the same for both cases. We can, therefore, make a comparison based on the response analysis circuitry alone. In the distributed case, an analyzer circuit is required for every bit of the n-bit adder. For the centralised case, only one analyzer circuit is used for the whole n-bit adder, but additional circuitry is also required to

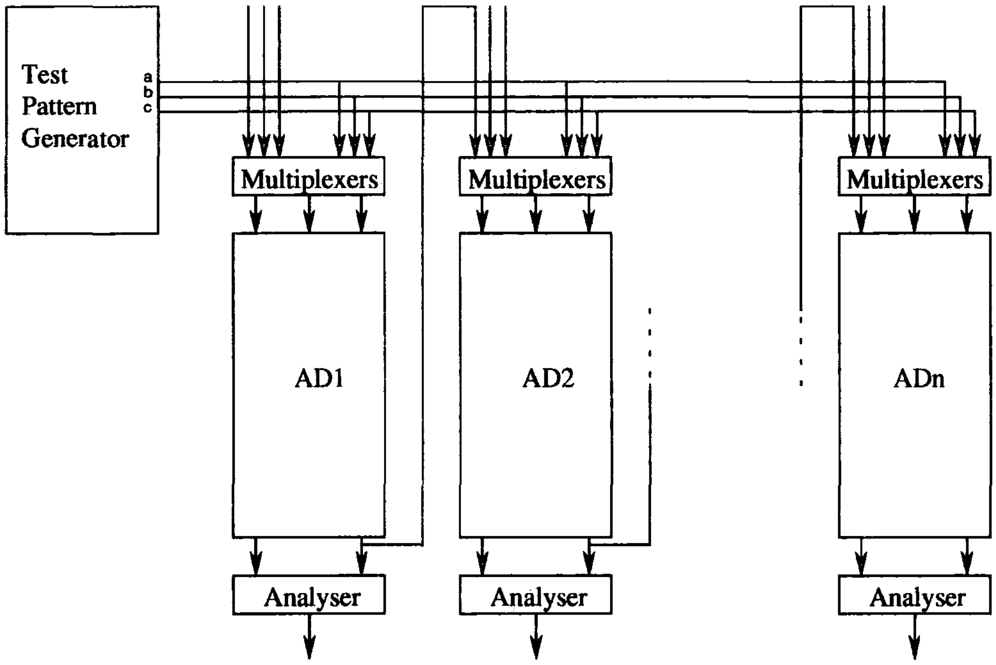


Figure 4.1 Distributed approach to test response analysis.

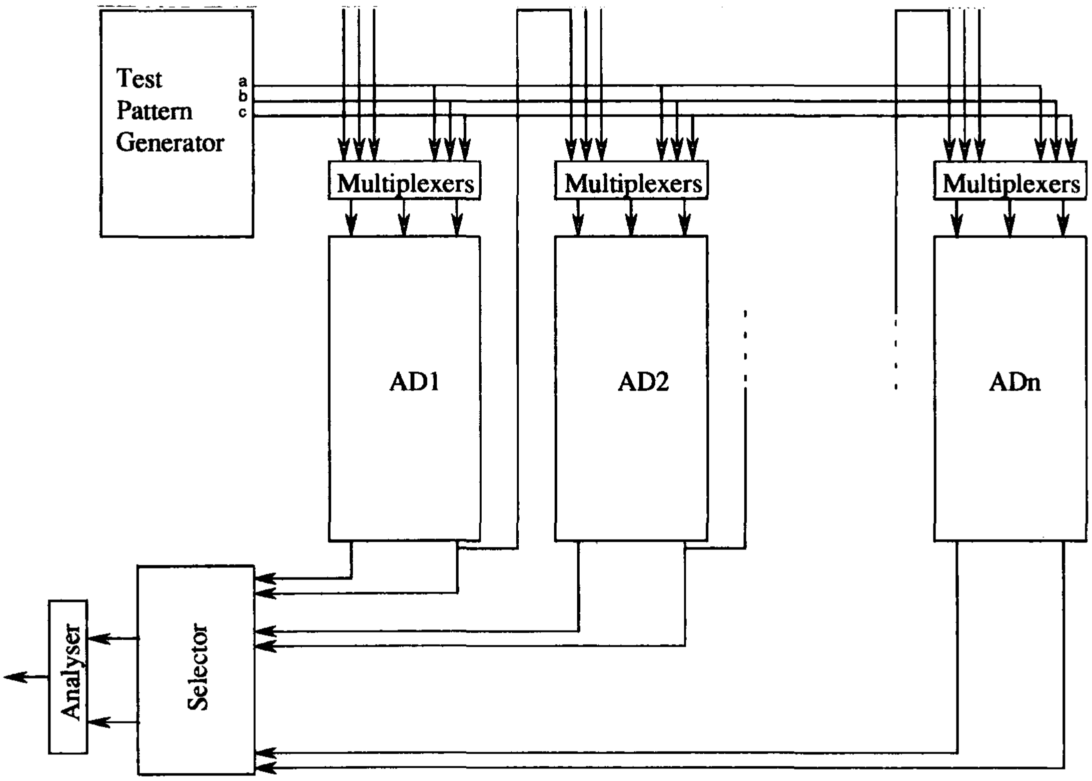


Figure 4.2 Centralised analyzer.

select one-of-n adder cells to be analyzed. The details of such circuitry are given in Fig. 4.3, for $n = 4$. For an n-bit adder, we require:

- A $\log_2(n)$-bit binary counter.
- A $\log_2(n)$-to-n decoder, and n inverters.
- $2n$ transmission gates.

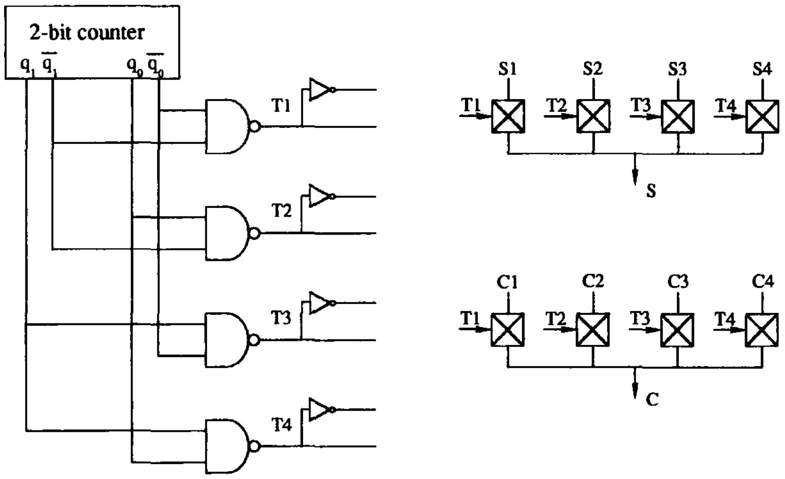


Figure 4.3 The selector circuits for the centralised analyzer.

Table 4.1 gives the size of the response analysis circuitry, as a number of transistors, for both cases and for different analyzer and adder sizes.

Table 4.1 Comparison of the sizes of response analysis circuitry for distributed and centralised approaches.

Analyzer	16 FETs		24		32	
Adder	Distributed	Centralised	Distributed	Centralised	Distributed	Centralised
1	16	16	24	24	32	32
2	32	34	48	42	64	50
4	64	84	96	92	128	100
8	128	164	192	172	256	180
16	256	318	384	326	512	334
32	512	646	768	654	1024	662

If the analyzer circuit contains fewer than 24 transistors, the hardware overhead figure makes the distributed case preferable to the centralised case. Even when the analyzer has 24 transistors or more, the difference between the two cases does not justify the use of the centralised analyzer. The distributed case has another important advantage over centralised testing in that it preserves the regularity of the n-bit adder circuit. The placement of the analyzer circuit is unambiguous, whereas the centralised nature of the analyzer in the alternative approach is bound to cause problems because of the necessity to route many signals through long distances. The strongest argument for preferring a distributed analyzer however is that a faulty analyzer circuit does not necessarily render the whole chip faulty.

4.2.2 Design and Placement of the Test Pattern Generator

As mentioned earlier, the test pattern generator is realised as a ROM array and a ring counter. For an m-vector long test sequence, an m-bit ring counter is preferred to the combination of a $\log_2(m)$-bit counter and a $\log_2(m)$-to-m decoder circuit, for two reasons: (1) Faults in the decoder circuit would be difficult to test, and (2) the size of the binary counter + decoder combination increases rapidly with the test sequence length.

Figure 4.4 gives a block diagram of the test sequence generator. Details of the ring counter cells are given in Fig. 4.5. Dynamic latches are used since there they are simpler; the function of the ring counter being simply to circulate a single logic one through the m cells. Furthermore, testing is performed at the normal operating speed, so that none of the usual problems associated with dynamic circuits are of concern here.

The first cell of the ring counter is designed such that, at the start of the test sequence, the RESET signal sets its output to logic one and sets all other cells to logic zero. In this way, each clock cycle will shift the one to the next stage, and the end of the test sequence is indicated by the return to logic one of the first stage. The output of every stage drives a word line in the ROM array.

The cells of the ring counter should be designed so that they are as narrow as possible in the direction of the propagation of the logic one. The separation between the word lines of the ROM array can be very small. Therefore, in order to reduce the wasted area, the width of the cells should match the separation between the word lines as closely as possible. It is clearly impossible to fit the ring counter cell of Fig. 4.5 into the smallest separation between the word lines allowed by the design rules, so there will always be some wasted area that we can only try to minimise. A possible layout

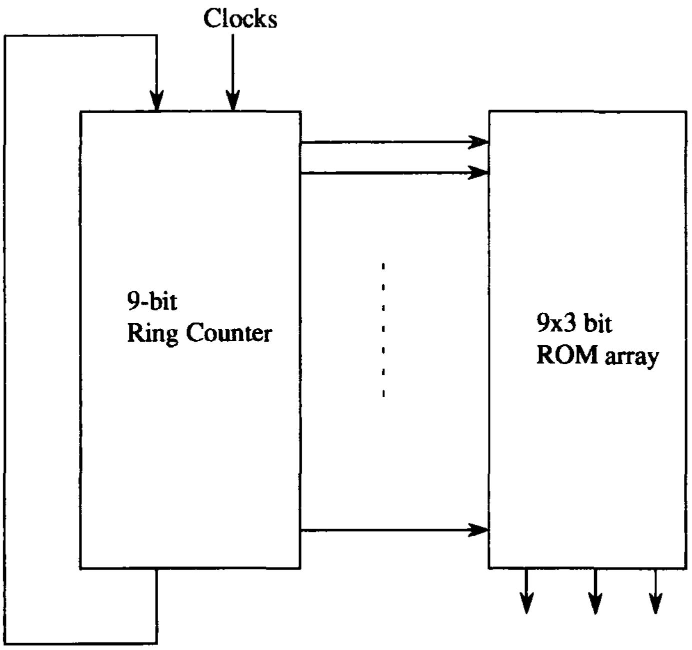


Figure 4.4 Block diagram of the test sequence generator.

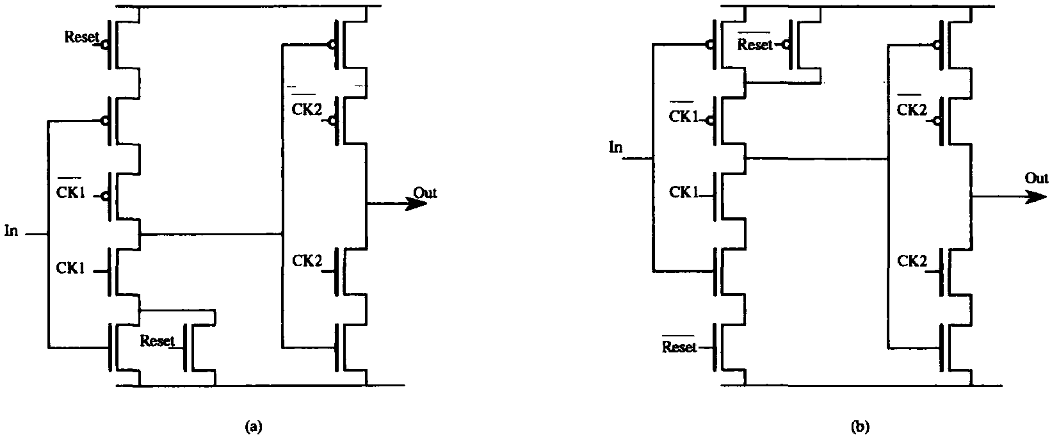


Figure 4.5 Implementation of the cells of the ring counter.

for the ring counter cell and the ROM array is shown in Fig. 4.6. The order of the different layers is modified in Fig. 4.6 for clarity.

If nine cells of the ring counter are stacked to form the test sequence generator for the full adder, then the resulting circuit will have an area of 432 × $160\mu m^2$, for $3\mu m$ design rules. A floor plan of the n-bit adder and the test sequence generator is shown in Fig. 4.7. Even if the area marked A0 can be used for the input multiplexers and the

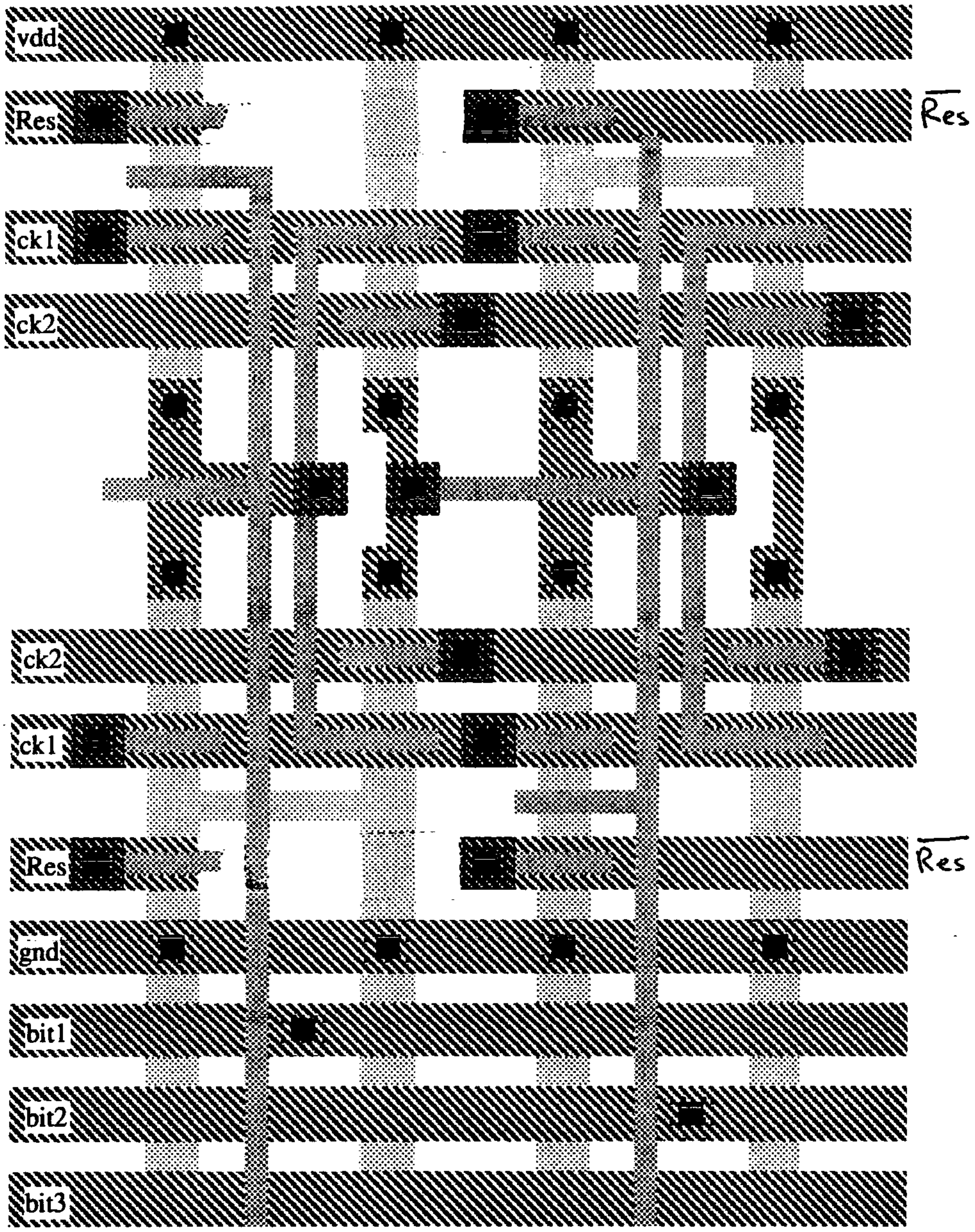


Figure 4.6 Layout of the test pattern generator.

output analyzers, the area overhead associated with BIST will be

$$O_v = \frac{\text{Test area}}{\text{Total area}} = \frac{432(160 + 136n) - 221 \times 136n}{432(160 + 136n)}$$

For $n = 8, O_v = 55.4\%$, whereas for $n = 32, O_v = 50.6\%$.

A better solution may be to lay down the ring counter and the ROM array as shown in Fig. 4.8, where the word lines are in the order $1, m, 2, m - 1, \ldots m/2, m/2 + 1$. In this case the area of the test sequence generator is approximately $240 \times 287 \mu\text{m}^2$. This not only reduces the total area of the test sequence generator, but it also offers better placement options of the test sequence generator in relation to the adder circuit.

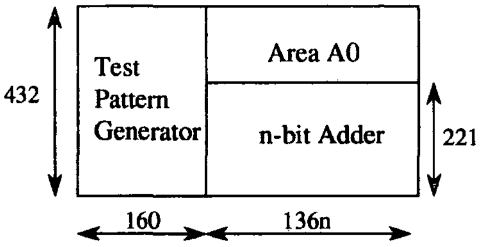


Figure 4.7 Floor plan of the n-bit adder and the test sequence generator.

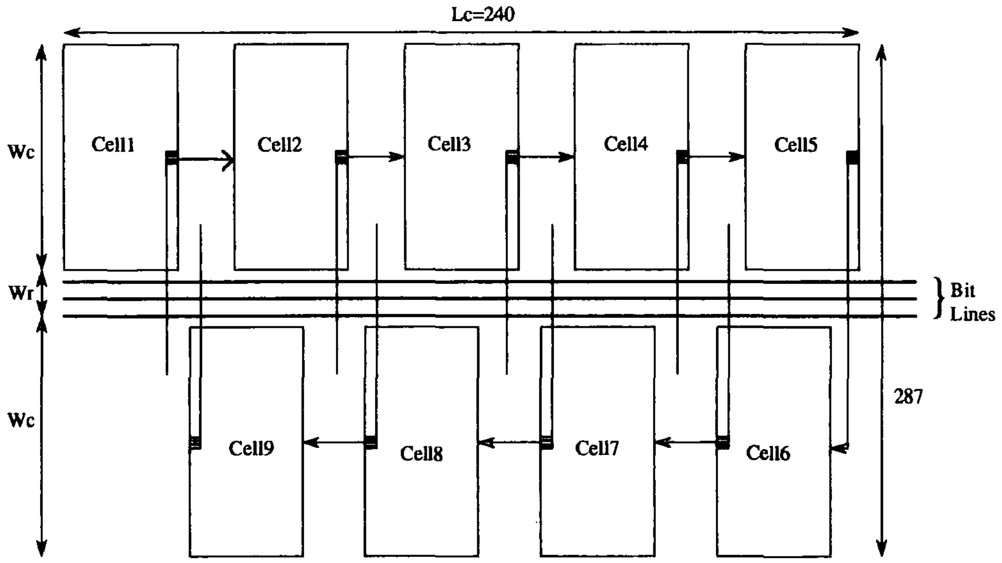


Figure 4.8 An alternative placement of the ring counter relative to the ROM array.

In Figs. 4.6 to 4.8, the ROM array is considered to have three bit lines corresponding to the three inputs of a full-adder cell. The sequence used to test this circuit is derived using the procedures of the previous chapter. Such test sequences produce a toggling output, obviating the need to provide the fault-free circuit response. However, because the full-adder has two outputs, SUM and CARRY, the definition of the toggling property needs to be considered. The approach suggested in Chapter 3, in such cases, was to test the outputs one at a time. The complete test sequence for the SUM circuit is 9-vectors long, and the CARRY circuit is completely tested by a 6-vectors long sequence (the last vector in the SUM sequence is used as an initialisation for the CARRY circuit). This would result in a 15-vector long test sequence for the full adder cell, shown in Table 4.2.

If this concatenated test sequence is used to test the full adder cell, then we would require a 15-bit ring counter. The output from the 10th stage, when set to logic one, triggers a flip-flop controlling the analyzer to switch the monitoring process from the

Table 4.2 Test sequence of a 1-bit adder.

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

SUM output to the CARRY output.

It was also said in the previous chapter that it is possible to overlap the testing of more than one output in order to reduce the test sequence length and the testing time. Looking at the previous test sequences, it can be seen that if it was not for input vectors 000 and 111, the test sequences for the SUM and CARRY circuits can overlap completely. To make use of this observation, we should try to construct a test sequence such that these two vectors appear late in the list, as in Table 4.3.

With such a test sequence, the output of the 8th stage of the ring counter will be used to switch off the monitoring of the CARRY output. Vectors 9 and 10 have been inserted between 000 and 111 to maintain the robustness of the test sequence, in case of circuit delays. In the built-in self-test environment, there is a better control over the circuit delays than when the test inputs are applied from the periphery of the chip, i.e., the robustness of the test sequence can be ensured by proper circuit design rather than by a proper choice of test vectors. Therefore, vectors 9 and 10 can be safely omitted

Table 4.3 Shorter test sequence for a 1-bit adder using overlapping.

	A B C	SUM	CARRY
1	0 0 1	1	0
2	0 1 1	0	1
3	0 1 0	1	0
4	1 1 0	0	1
5	1 0 0	1	0
6	1 0 1	0	1
7	0 0 1	1	0
8	0 0 0	0	0
9	0 0 1	1	0
10	0 1 1	0	1
11	1 1 1	1	1

from the previous test sequence to reduce the length to 9-vectors.

4.2.3 Design and Placement of the Analyzer Circuit

The function of the analyzer circuit is to monitor the SUM and CARRY outputs. According to the previous section, the SUM output should be an alternating signal throughout the test sequence, but the CARRY output is toggled only up to the seventh vector of the sequence. Therefore, the inputs to analyzer are as follows.

- The SUM and CARRY outputs of a full-adder cell.
- A toggling signal, T, which is simply a clock signal of half the frequency of the main clock.
- A signal that switches OFF the monitoring of the CARRY output.

The resulting logic circuit is shown in Fig. 4.9(a), where Sw=1 disables the monitoring of the CARRY output, and Oa=0 indicates the presence of a fault in the SUM or CARRY circuit.

A static CMOS implementation of the above circuit consists of 24 transistors. This figure makes the distributed analyzer approach less attractive than the centralised analyzer approach, in terms of hardware requirements. Therefore, the above circuit is

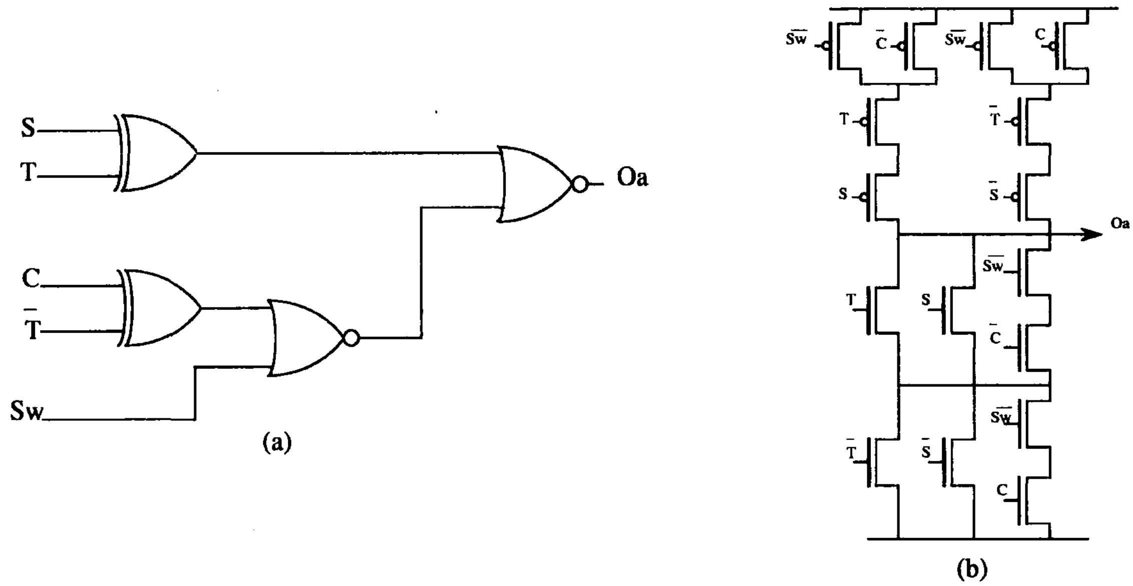


Figure 4.9 Analyzer circuit (a) and its complex gate implementation (b).

designed as a single CMOS complex gate, shown in Fig. 4.9(b), to reduce the number of transistors to 16, which is the minimum possible numbers. Both the SUM and CARRY inputs need complementing.

The layout of the full-adder cell was given in Chapter 3. To form an n-bit adder, n cells are cascaded horizontally. The inputs and outputs of every cell run vertically, in metal 2, either from the top or bottom of the cell. This fact is used to split the input multiplexers into two halves, each consisting of three transmission gates. The first half, at the top, selects the data inputs. The second half is used to select the test inputs and can be shared by all n adder cells if it is moved to the output of the ROM array. The output inverters are placed close to the top half of the input multiplexers and the analyzer circuits are at the bottom of every full adder cell as shown in Fig. 4.10.

4.2.4 Area Overhead Figures

The floor plan of the n-bit adder and the test circuitry is shown in Fig. 4.11, together with the dimensions of the different blocks. The area overhead O_v is given by

$$O_v = \frac{\text{Test Area}}{\text{Total Area}} = \frac{\text{Total Area} - \text{Adder Area}}{\text{Total Area}} = 1 - \frac{\text{Adder Area}}{\text{Total Area}}$$

which is

$$O_v = 1 - \frac{221 \times 136n}{(136n + 287)318}$$

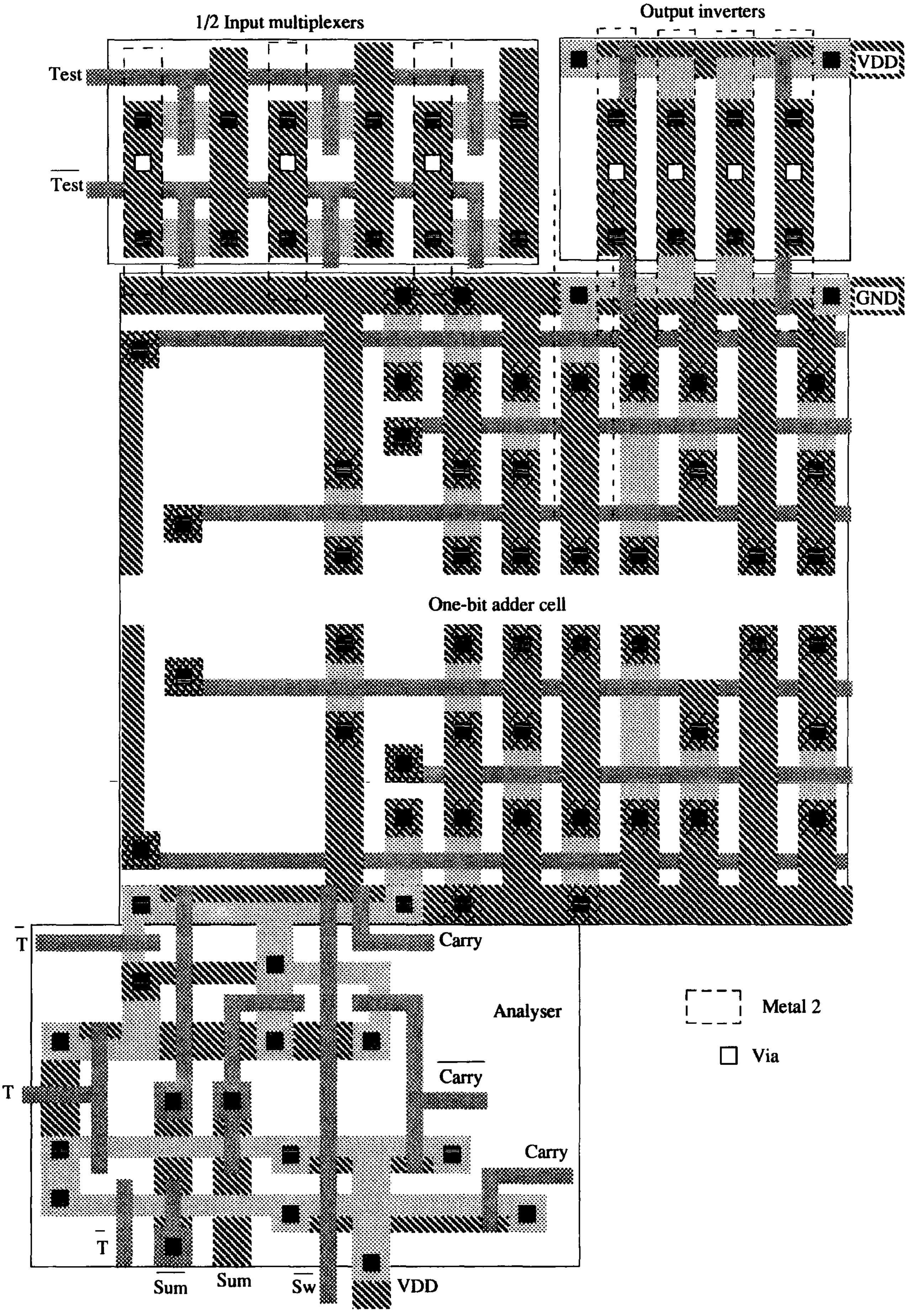


Figure 4.10 Layout of the input multiplexers, output inverters, and analyzer circuit.

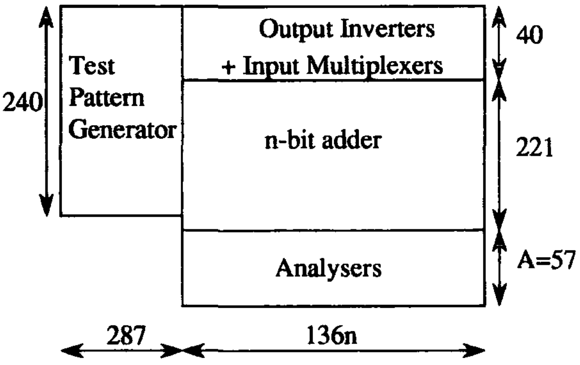


Figure 4.11 Floor plan of the adder circuit and the test circuitry.

The area overhead is plotted against the adder size in Fig. 4.12. Note that with a floor plan as shown in Fig. 4.11, the depth of the analyzer circuit has a crucial influence on the area overhead. For a 32-bit adder, the area overhead varies from 21% to 43% as the depth of the analyzer circuit is varied from 0 to 100μm. The layout of Fig. 4.10 corresponds to an analyzer depth of 57μm.

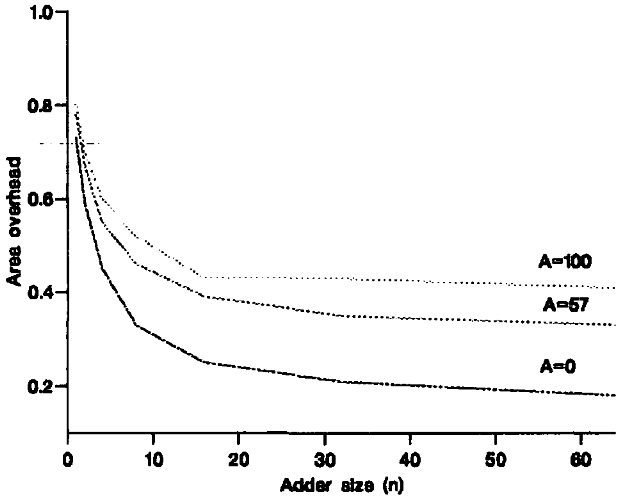


Figure 4.12 Area Overhead for different adder sizes.

4.3 TESTING THE EXTRA HARDWARE: IS COMPLETE TEST COVERAGE POSSIBLE?

Faults in the extra hardware introduced for BIST may result in the situation where a faulty unit is flagged as fault-free or vise versa. Therefore, it is important to test the extra hardware as rigorously as the functional circuits.

In this section, the testing of the added test circuitry is investigated. This circuitry falls into three categories: response analysis circuitry, test pattern generator, and the input multiplexers and output inverters.

4.3.1 Testing the Analyzer Circuit

The output of the analyzer considered in the previous section assumes a constant logic 1 value if the circuit under test is fault-free. Hence, a stuck-at 1 fault on this output would not be detected, in addition to most of transistor stuck faults. The detection of the stuck-at 1 fault requires that the output of the analyzer be driven to logic 0 at least once during the test sequence. Furthermore, the detection of stuck-open faults requires that $0 \rightarrow 1$ and $1 \rightarrow 0$ transitions be induced at the analyzer output.

Considering Fig. 4.9(b), it is clear that if SUM is an alternating signal and if when Sw=0, CARRY is also an alternating signal, then none of the paths in the pull-down network of the analyzer are activated. Therefore, the test sequence for the 1-bit adder needs to be augmented by some test vectors such that the resulting sequence does not always toggle SUM and CARRY (when Sw=0), in order to activate these paths.

Let us derive a complete test sequence and see how it can be applied to the circuit of Fig. 4.9(b). The set of special 1-vertices is C_s = Sw T S C = $\{0001, 0110, 1000, 1111\}$ and one possible set of special 0-vertices is $D_s = \{0000, 0111, 1010, 1100\}$. Combining C_s and D_s yields the test sequence shown in Table 4.4.

A major problem with this test sequence is that signals T and Sw are considered as controllable inputs, i.e, they need to be produced by the test pattern generator. Furthermore, the role of signal Sw, as defined in Section 4.2, was to switch off the monitoring of the CARRY output, once all faults in the CARRY circuit have been tested. With the above test sequence this definition becomes ambiguous: some of the faults in the CARRY circuit are tested by vector 2, while the remaining are tested by the sequence following vector 9.

An alternative approach that does not assume that T is a controllable input is now presented. Set C_s is divided into two sets C_{s0} and C_{s1} according to whether input T is 0 or 1, respectively. This gives $C_{s0} = \{1000, 0001\}$, $C_{s1} = \{0110, 1111\}$. A similar operation is performed on set D_s resulting in $D_{s0} = \{0000, 1010\}$, $D_{s1} = \{0111, 1100\}$.

The complete test sequence consists of two parts. The first part is obtained by combining C_{s0} and D_{s1} and the second part is obtained by combining sets C_{s1} and D_{s0}, as shown in Table 4.5.

Table 4.4 Complete test sequence for the analyzer circuit.

	A	B	C	Sw	T	S	C_0	Oa
1	1	1	1	0	1	1	1	0
2	0	0	1	0	1	1	0	1
3	0	0	1	1	0	1	0	0
4	0	0	0	1	0	0	0	1
5	0	0	0	1	1	0	0	0
6	1	1	1	1	1	1	1	1
7	1	1	1	0	1	1	1	0
8	0	1	1	0	0	0	1	1
9	0	0	0	0	0	0	0	0
10	0	0	1	0	1	1	0	1
11	1	0	1	1	1	0	1	0
12	0	1	0	0	1	1	0	1
13	1	1	0	1	1	0	1	0
14	1	0	0	0	1	1	0	1

The disadvantage of the above test sequence is that the analyzer output is not toggled throughout the test sequence. In fact, it is not possible to toggle signal T and the output of the analyzer at the same time, since this would require that Oa=T or $\overline{\text{Oa}}$=T for all special vertices. Although the initial definition of signal T is preserved in the test sequence of Table 4.5, signal Sw is still considered as a controllable input, making its function unclear. Furthermore, the fault-free response of the analyzer also needs to be provided by the test pattern generator.

In both of the above test sequences for the analyzer, two extra signals need to be produced by the test pattern generator, in addition to the three inputs of the 1-bit adder (T and Sw in the first case, the fault-free analyzer's response and Sw, in the second case). It would have been simpler to generate the fault-free response of the 1-bit adder in the first place without taking advantage of the toggling property. A better approach, which makes an effective use of the toggling property is described below.

In the new method, the SUM and CARRY outputs are combined into a single output by a combinational circuit that is designed so that all the faults can be detected

Table 4.5 Alternative complete test sequence for the analyzer.

	Sw	T	S	C	Oa
1	1	0	0	0	1
2	0	1	1	1	0
3	0	0	1	0	1
4	1	1	0	0	0
5	1	0	0	0	1
6	0	1	1	0	1
7	0	0	0	0	0
8	1	1	1	1	1
9	1	0	1	0	0
10	0	1	1	0	1

by observing the combined output only. This two-input combinational circuit is chosen from the ten non-trivial functions of two-variables. Among these functions, only the XOR and XNOR functions can be used. For all other functions, there are some faults that cannot be detected. The implementations of these functions require only half the number of transistors of the initial analyzer circuit. The drawback, however, is that a longer test sequence is required to test for all faults and toggle the combined output. The test sequence, when a XOR gate is selected, is shown in Table 4.6.

A 24-bit ring counter is required in this method, making the floor-plan of Fig. 4.11 unsuitable, since a large area would be wasted. The problem is solved by simply rotating the test pattern generator by 90^o relative to its orientation in Fig. 4.11. The area overhead becomes

$$O_v = 1 - \frac{\text{Adder Area}}{\text{Total Area}} = 1 - \frac{221 \times 136n}{287 \times (576 + 136n)}$$

which is slightly less than the figures of Section 4.2 for $n \geq 16$.

It should be noted that after vector 5, all faults in the XOR gate have been tested, and it is not necessary to continue toggling the XOR gate output in order to test the remaining faults in the SUM and CARRY circuits. It is therefore possible to maintain the output of the XOR gate at a constant logic value while testing the SUM and CARRY circuits, as shown in Table 4.7. However, the hardware required to monitor the output of the XOR gate in this case may offset any gains from a shorter test sequence.

Table 4.6 Test sequence of the SUM, CARRY and XOR gate.

	A B C	S Co	Out	Detected stuck-open faults
1	1 1 1	1 1	0	
2	0 1 1	0 1	1	P19, P22, N9, N12 , N13
3	0 0 0	0 0	0	N21, N22
4	0 0 1	1 0	1	P20, P21, P4, P5, P6
5	1 1 1	1 1	0	N19, N21
6	1 0 1	0 1	1	N9, N10, N11
7	1 1 1	1 1	0	P6, P7, P8
8	1 1 0	0 1	1	N4, N5, N6
9	0 0 0	0 0	0	
10	1 0 0	1 0	1	P11, P12, P13
11	0 0 0	0 0	0	N4, N7, N8
12	0 1 0	1 0	1	P9, P10, P13
13	0 0 0	0 0	0	
14	0 1 1	0 1	1	P15, P16
15	0 0 0	0 0	0	
16	1 0 1	0 1	1	P14, P15
17	0 0 0	0 0	0	
18	1 1 0	0 1	1	P17, P18
19	1 1 1	1 1	0	
20	0 0 1	1 0	1	N17, N18
21	1 1 1	1 1	0	
22	0 1 0	1 0	1	N14, N15
23	1 1 1	1 1	0	
24	1 0 0	1 0	1	N14, N16

The idea of combining the outputs of a circuit into a single signal for response analysis purposes can be generalised as an alternative solution to the problem of testing multi-output circuits, instead of sequentially testing one output at a time.

Even when using the test sequence of Table 4.6, we still need a toggle detector to monitor the output of the XOR gate. This can be a simple circuit that compares the XOR-gate output with signal T, as defined in Section 4.2. However, this simple circuit would also need testing, and therefore, another circuit is required to monitor its output,

Table 4.7 A shorter test sequence for the SUM, CARRY and XOR gate.

	A	B	C	S	Co	Out
1	1	1	1	1	1	0
2	0	0	1	1	0	1
3	0	0	0	0	0	0
4	0	1	1	0	1	1
5	1	1	1	1	1	0
6	0	1	1	0	1	1
7	0	0	1	1	0	1
8	1	0	1	0	1	1
9	0	1	0	1	0	1
10	1	1	0	0	1	1
11	1	0	0	1	0	1

which also needs testing, and so on. This is similar to the problem of self-checking circuits discussed in Chapter 2. However, in the present case, there is a simple method to stop this endless process.

Analyzing a signal by comparing it to a reference signal requires a continuous monitoring of the outcome of the comparison. This is the reason for the above endless process. When analyzing an alternating signal, the focus should be on the 0 → 1 and 1 → 0 transitions. The absence of any transition ends the monitoring process, regardless of subsequent values of the signal. A possible circuit implementation that achieves this goal is shown in Fig. 4.13.

Input T is as defined previously. Input A is the alternating signal to be monitored. Input SET sets the FLAG output to logic 1 at the start of the test sequence. When A follows T, a logic 1 is stored in the latch. Whenever, A is different from T, a logic 0 is stored in the latch. This will turn transistor N off, trapping the faulty state until the next SET.

In a fault-tolerant design, each 'unit' would have its own toggle detector. The FLAG signals control the switches that enable/disable the corresponding unit. Ideally, all toggle detectors should be included in a scan-chain so that they are accessible from outside the chip. This would also help their testing.

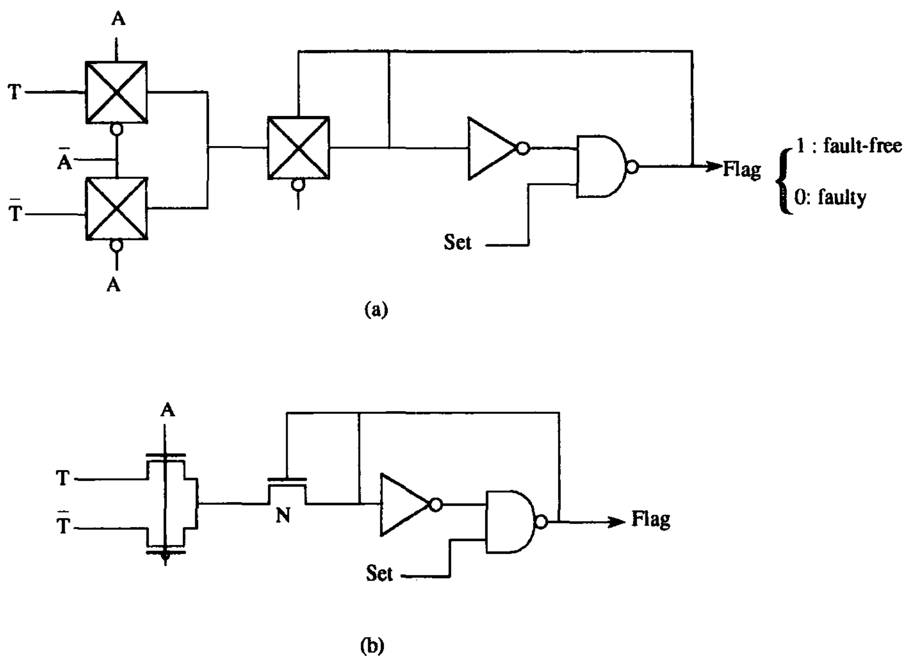


Figure 4.13 Toggle detector.

Assuming that we have access to the FLAG signals, the following procedure tests for faults in the toggle detector. All flags are set to logic 1 initially, and all units receive their normal test sequences. However, input T is complemented during the first two vectors. At the end of the test sequence, all fault-free toggle detectors should have their output at logic 0. This will detect stuck-open faults in all transistors of the circuit with the exception of the pFETs of the NAND gate. A stuck-on fault on nFET N is also detected. Asserting the SET input at this stage will test for the stuck-open on the pFET driven by SET. A stuck-open fault on the other pFET of the NAND gate is undetectable. It turns the static latch into a dynamic one.

4.3.2 Testing the Test Pattern Generator

The test pattern generator consists of a ring counter and a ROM array. The problem of testing the ring counter is similar to the problem of testing the latches in a scan-path arrangement. A complete structural test of an m-bit counter should be constructed by considering the circuit as having 3 inputs (CK1, CK2, and the input to the first stage (after breaking the feedback)), and m outputs. However, because of the simplicity of the adopted implementation, a better approach is proposed, The feedback line from the last to the first stage is broken with a multiplexer, and an alternating signal is applied to the first stage. After m clock cycles, the alternating signal should appear at the output of the last stage, if no faults are present. In this way, the number of outputs to be monitored is reduced from m to just one. This simple approach can be shown to detect all detectable faults in the ring counter, with the exception of stuck-open fault on the transistors driven by RESET. Stuck-open faults on these transistors are detected by setting the first stage to 0 an the remaining stages to 1, and then asserting RESET. After m clock cycles, a logic 1 should then appear at the output of the last stage if there are no faults.

The ROM array is similar to the OR-plane of a PLA. The cross-point fault model is the most appropriate for testing such structures [45, 180]. In this fault model, all physical faults are assumed to result in a missing or an extra device.

Considering the ROM as having m inputs (word lines), it is possible to convert the pull-ups into a shift register in order to have access to every cross-point of the ROM. However, given the small size of the ROM used so far, a testing procedure that addresses every cross-point would require even more hardware than simple duplication of the ROM array.

It was not possible to find a testing method for the ROM that requires less hardware overhead than duplication. However, since the test sequence stored in the ROM has a very precise ordering, any fault in the ROM is bound to disturb this ordering. For example, any single cross-point fault would result in a test sequence that does not toggle the SUM output. But a double cross-point fault results in a test sequence that still toggles the SUM output, while some of the transitions at the CARRY output may be missing. Therefore, if at the end of the test sequence all adder cells are reported as faulty, it is very likely that the ROM itself is faulty. A faulty adder cannot be reported as fault-free, unless there is a double cross-point fault. In a fault-tolerant design, it is not relevant to distinguish between the case where all or most adder cells are faulty and the case when the ROM array is faulty: in both cases the chip is discarded.

Note that duplicating the ROM would result in a only a small increase in the area overhead (from 34.8% to 35.2% for a 32-bit adder) because of the small size of the ROM.

4.3.3 Testing the Remaining Test Circuitry

For the input multiplexers, Fig. 4.14 presents a design approach that would make stuck-open faults detectable. Stuck-on faults are also detectable if the intermediate voltages that they induce are far enough from their fault-free values. The output inverters are tested without any extra effort, since a transition at the input of the inverter is all what is needed to induce an erroneous output if a fault is present.

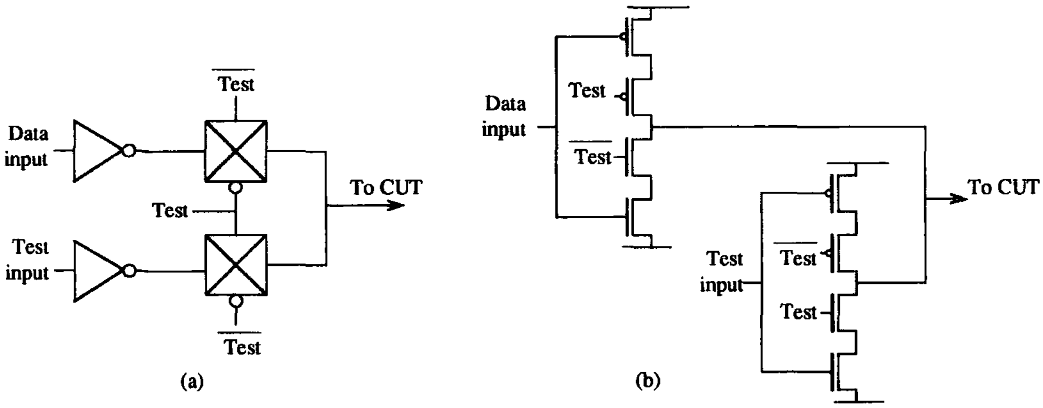


Figure 4.14 Input multiplexers (a) and a testable realisation (b).

4.3.4 Is Complete Test Coverage Possible?

The proposed BIST approach aims at detecting all detectable faults in the functional circuit. This was shown to be achievable at a moderate hardware overhead. However, to achieve complete test coverage, the test circuitry should also be tested for all detectable faults. This requires more test circuitry that also needs testing. This may be conceptually represented as in Fig. 4.15, where tester0 tests the functional circuit, tester1 tests tester0, and so on.

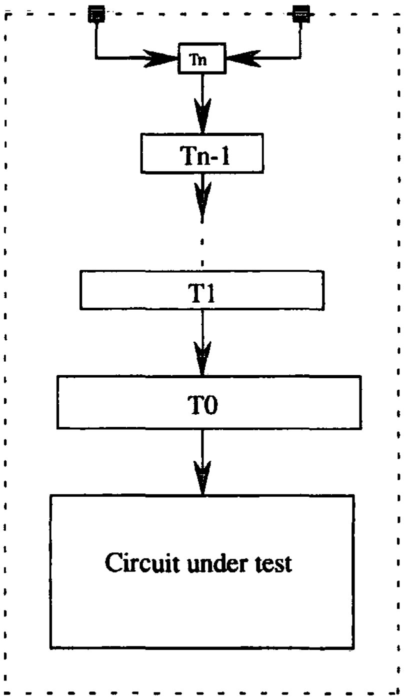


Figure 4.15 Hierarchy of testers.

For this to be effective, tester i needs to be smaller than tester $i-1$ so that the untested part is reduced to tester n, and it can be made small and simple enough to be easily tested from outside the chip, or incorporated in the boundary scan test. Note that even if this was realisable, it would not strictly give complete test coverage, since there is still a need for some external testing, but it would be an acceptable option. However, as seen in the previous subsections (especially the testing of the ROM array), it does not seem possible to have tester i smaller than tester $i - 1$.

Figure 4.16 depicts another problematic situation. Tester T1 fully tests module M1 but does not test the interconnections between modules M1 and M2. It is conceivable to have a third tester T3 that tests both the interconnections and the testers T1 and T2. However, there is still some other interconnections that are not tested. In a multi-chip system, we may have a single extra chip that tests the untested parts of the other chips. For complete test coverage, this extra chip also needs testing. However, in practice it should be simple enough for it to be tested externally. Alternatively, it is possible for the chips of a system to 'cooperate' in testing each others untested parts.

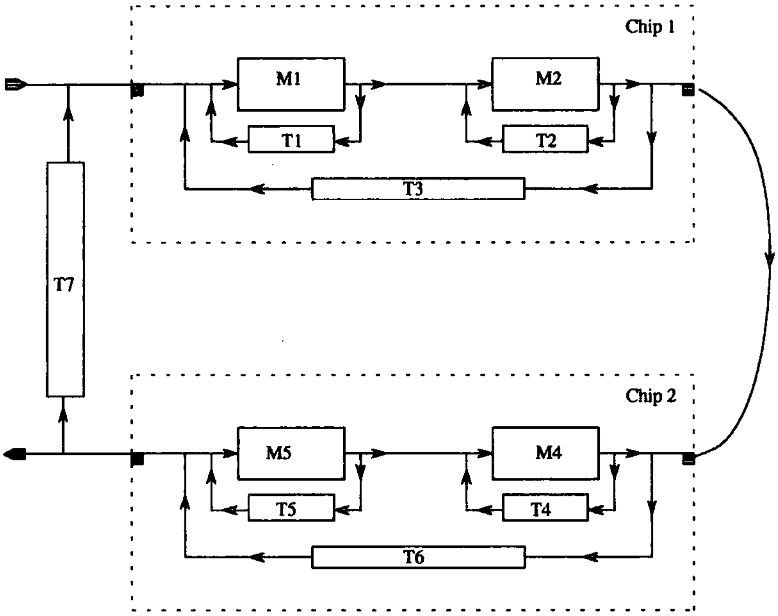


Figure 4.16 Chip cooperation to test each others untested parts.

We may conclude that a chip cannot completely test itself: it needs external assistance. For the same reasons, a single-chip system cannot be completely self-testing. In a multi-chip system, it seems feasible to achieve complete test coverage by making chip i test the untested parts of chip $i+1$. Merlino et. al. [181] arrive at a similar conclusion by showing that, under certain conditions, the package-related faults become dominant and therefore, BIST implementations must exercise the chip through its package pins.

An effective approach towards complete test coverage would be to detect all faults in the test circuitry while it is performing its function. For example, the XOR gate used in Section 4.3.1 to combine the outputs into a single signal is completely tested while performing its function. No further extra hardware is needed if all test circuits are tested in this manner. However, faults in the test hardware will be indistinguishable from faults in the functional hardware. This does not present any problems if only BIST is considered. In a fault-tolerant chip, a further requirement is that the test circuitry be as distributed as possible, i.e., each functional unit should have its own test hardware, and there should be very little or, preferably, no global test circuits that are shared by many units. In this case, there is no need to distinguish faults in the test hardware from faults in the functional hardware.

Finally, and as noted in Chapter 2, the benefits of BIST are recognised in many publications but it is still only timidly used in commercial products. The reluctance towards the use of BIST is due to two reasons:

- The extra design effort required.
- The extra hardware introduced by BIST.

The second reason is probably the most relevant, especially in mass-produced, state of the art chips. BIST can be achieved with a hardware overhead ranging from 20% to 50% [104, 182, 183], depending on the fault coverage aimed at. Complete test coverage, as discussed in this section, would require higher hardware overheads. Therefore, it will be a long time before it is considered for real-life applications.

4.4 EFFECT OF PARTITIONING

The area overhead figures of Section 4.2 show that the implementation of BIST in a 1-bit adder results in a very large overhead ($\approx$ 80%). The overhead for a 32-bit adder was 35%. However, this overhead would have been much larger if the 32-bit adder was considered as a single unit having 65 inputs and 33 outputs. The lower overhead figure was obtained only because the 32-bit adder was considered as 32 separate 1-bit adder units.

In this section, we investigate the effect of using different unit sizes. An n-bit adder is considered as a set of n/n_0 units of n_0-bit adders. The hardware overhead associated with the introduction of self-test is evaluated for different adder sizes (n) and different unit sizes (n_0). However, since it is not practical to do a layout for all possible combinations, the hardware overhead figures are expressed in terms of the number of transistors.

In an n_0-bit adder, we have access to the two input words A and B, the carry input to the first stage the carry output from the last stage, and the sum word. It is assumed that there is no access to the internal carry signals. The size of the test pattern generator depends on the test sequence length. This is the subject of the next subsection.

4.4.1 Test Sequence Derivation for an n_0-Bit Adder Block

The modularity of the adder circuit allows for a simple test generation procedure. All n_0 sum outputs must be toggled at the same time so that only a single T signal is used to analyze them. The test generation procedure is as follows:

1- Apply a complete test sequence to the first adder cell. This will result in the sum output being toggled and the carry output having certain values.

2- Append to the resulting carry the inputs A and B of the next stage so that a complete test sequence is applied to this next stage.

3- Repeat step 2 for the remaining adder cells.

The test sequences applied to each cell do not need to be exactly the same. The fact that the adder cell can be completely tested by many different test sequences is used in step 2 in order to obtain the shortest possible test sequence for an n_0-bit adder. The procedure is illustrated in Table 4.8.

The sequence length is $m = 7 + 2n_0$. This test sequence has the advantage that, for any n_0-bit adder, the output of the $(m-2)^{\text{th}}$ stage of the ring counter is used to stop the monitoring of the carry output. The disadvantage is that, as n_0 increases, the number of redundant vectors (marked with a star), that are used just to maintain a toggling output, also increases.

4.4.2 Hardware Requirements

The length of the test sequence determines the 'length' of the ROM array and the ring counter. The width of the ROM is determined by the number of test inputs to be generated. For an n_0-bit adder, this number is simply $2n_0 + 1$, corresponding to the two n_0-bit input words and the carry-in of the first stage.

Expressing the 'size' of the ROM array in terms of its number of transistors presents some problems, since the physical size of a ROM is in no way related to its transistor content. Let A_c and A_r be the areas of the ring counter and the ROM, respectively. The transistor count of the ring counter is simply $N_c = 10(7 + 2n_0)$. Assuming that

Table 4.8 Illustration of the test procedure for an n_0-bit adder.

C1 A1 B1	S1	C2 A2 B2	S2	C3 A3 B3	S3	C4 A4 B4	S4	C5
0 0 1	1	0 0 1	1	0 0 1	1	0 0 1	1	0
0 1 1	0	1 0 1	0	1 0 1	0	1 0 1	0	1
0 1 0	1	0 1 0	1	0 1 0	1	0 1 0	1	0
1 1 0	0	1 1 0	0	1 1 0	0	1 1 0	0	1
1 0 0	1	0 0 1	1	0 0 1	1	0 0 1	1	0
1 0 1	0	1 0 1 *	0	1 0 1 *	0	1 0 1 *	0	1
0 0 1	1	0 0 1 *	1	0 0 1 *	1	0 0 1 *	1	0
0 0 0	0	0 1 1	0	1 0 1 *	0	1 0 1 *	0	1
1 1 1	1	1 0 0	1	0 0 1 *	1	0 0 1 *	1	0
	0	0 0 0	0	0 1 1	0	1 0 1 *	0	1
	1	1 1 1	1	1 0 0	1	0 0 1 *	1	0
	0		0	0 0 0	0	0 1 1	0	1
	1		1	1 1 1	1	1 0 0	1	0
	0		0		0	0 0 0	0	0

the transistor count and the area are such that $A = \alpha N, \alpha = \text{const}$, we can define an equivalent transistor count N_r for the ROM array as follows (refer to Fig. 4.8).

$A_c = 2L_c \times W_c$, $A_r = L_c \times W_r = \frac{A_c}{2W_c} W_r = \alpha N_c \frac{W_r}{2W_c} = \alpha N_r$. Therefore, $N_r = \frac{W_r}{2W_c} N_c$, and since the width of the ROM is $W_r = (2n_0 + 1)W_0$, where W_0 is the width required for a single bit, then $N_r = \frac{W_0}{W_c}\frac{(2n_0+1)}{2} N_c$. The constant ration W_0/W_c is determined from Fig. 4.6.

The size of the test pattern generator, expressed as a number of transistors, is

$$\text{TPG} = 10(7 + 2n_0)\left(1 + \frac{W_0}{W_c}\frac{2n_0 + 1}{2}\right) \tag{4.1}$$

Figure 4.17 is a block diagram of the n_0-bit adder together with the remaining test circuitry.

The carry-out analyser circuit implements the function $f = \text{Sw} + \text{Cout}\,\overline{\text{T}} + \overline{\text{Cout}}\,\text{T}$

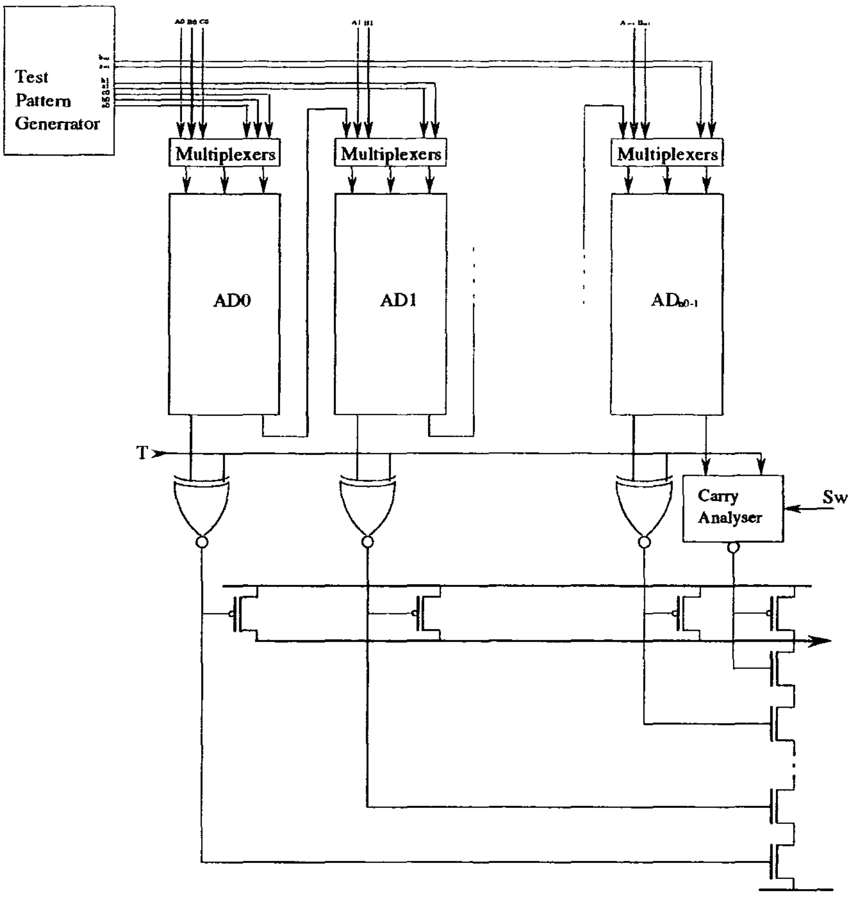


Figure 4.17 Block diagram of an n_0-bit adder and test circuitry.

where signals Sw and T are as defined in section 4.2. It is implemented as a complex gate consisting of 10 transistors.

4.4.3 Hardware Overhead Figures

Table 4.9 summarises the sizes of the different circuits required for BIST.

Table 4.9 Size of the different circuits required for BIST.

n_0	1	2	4	8	16	32	
Input MUXs	12	20	36	68	132	260	$N_1 = 4(2n_0 + 1)$
Output INVs	4	6	10	18	34	66	$N_2 = 2(n_0 + 1)$
EXNOR gates	8	16	32	64	128	256	$N_3 = 8n_0$
Carry analyzer	10	10	10	10	10	10	$N_4 = 10$
Output NAND	4	6	10	18	34	66	$N_5 = 2(n_0 + 1)$
TPG	101	133	208	399	947	2708	(4.1)

For simplicity, it is assumed that there are no restrictions on the fan-in of the output NAND gate.

For an n-bit adder implemented as n/n_0 units of n_0-bit adders, the total size of the test circuitry is given by

$$\text{TPG} + \frac{n}{n_0}(N_1 + N_2 + N_3 + N_4 + N_5)$$

Table 4.10 gives the overhead as a percentage of the total size for different values of n and n_0. Figure 4.18 shows that in all practical cases, there is always an optimum unit size for which the introduction of BIST yields the smallest overhead.

Table 4.10 Percentage overhead.

n_0 n	1	2	4	8	16	32
1	79	-	-	-	-	-
2	71	72	-	-	-	-
4	64	63	68	-	-	-
8	58	56	58	67	-	-
16	55	51	51	57	69	-
32	53	48	46	49	58	74
64	52	46	44	44	50	63

4.5 TEST SEQUENCE GENERATORS

The detection of transistor stuck-open faults requires a precise ordering of the input test vectors. Furthermore, if we want to take advantage of the toggling property, then the sequencing of the input test patterns becomes crucial. In the previous sections we considered the generation of such test sequences by means of a ring counter and a ROM array configuration, which is the simplest approach. In this section, we investigate other approaches in order to see whether they yield a smaller hardware overhead, and whether the resulting circuitry is easier/harder to test. First, we look at previously proposed implementations of test sequence generators for CMOS faults.

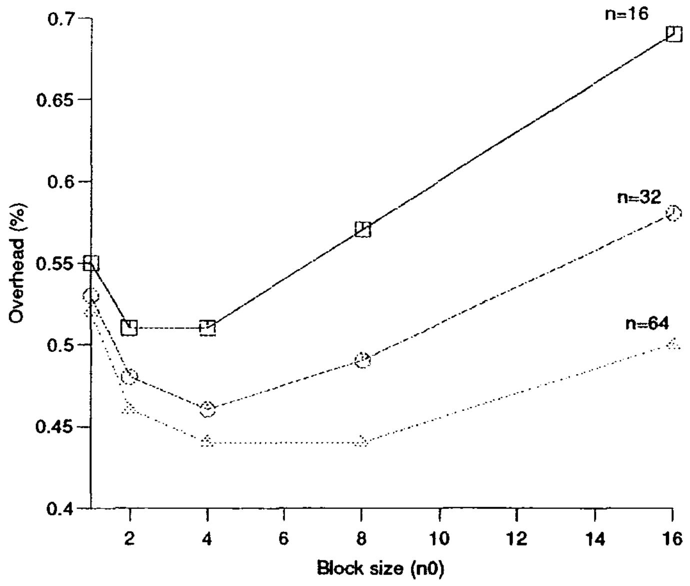


Figure 4.18 Effect of partitioning on hardware overhead.

4.5.1 Review of Proposed Techniques

The techniques to be described are analogous to the pseudo-exhaustive testing of stuck-at faults, in that they do not attempt to generate a predefined test sequence. Instead, a much longer sequence, which should completely test the circuit is generated by a Linear or Non-linear Feedback Shift Registers (LFSR or NFSR).

The first technique, called Pseudo-Exhaustive Transition Testing (PETT) [184] uses a $2n$-bit feedback shift register to test an n-input combinational circuit, where the inputs are taken from every other stage of the register. If the feedback is such that the FSR generates a maximum-length sequence, then all possible transitions are applied to the CUT, i.e., every n-bit input combination appears 2^n times preceded by every other $2^n - 1$ input combinations, and preceded once by itself.

The $(2^{2n} - 1)$-vector long sequence generated by a maximum length $2n$-bit LFSR, therefore, contains all the transitions necessary to the detection of all stuck-open faults in an n-input circuit. However, it also contains many transitions that are not needed. A procedure is presented in [184] to design the feedback circuit in such a way that the number of extra transitions is reduced. In the example given in [184], a test sequence consisting of 16 4-bit vectors is generated by an 8-bit NFSR. The design procedure yields a sequence of 28 vectors, nearly doubling the length of the initial test sequence.

The drawback with PETT is that, because the circuit response does not follow any particular pattern and the test sequence length is much longer than a deterministic one, the only option for response analysis is signature analysis. This negates the benefit of having an input test sequence that detects all possible stuck-open faults.

Pseudo-Exhaustive Adjacency Testing (PEAT) is proposed in [185] as an alternative to PETT to alleviate the problem of test sequence length. An adjacency test is defined as a two-pattern test for which the two vectors differ in a single bit position. This is inspired from the condition of robustness of a two-pattern test. In PEAT all adjacency tests are generated. It is implemented using an n-bit NFSR and an n-bit flip control register. The procedure for PEAT is as follows:

1. Clock the NFSR.
2. Sequentially flip each register bit in the NFSR in a fixed (but arbitrary) order. This produces a new adjacency test for every flip.
3. Repeat 1 and 2 for the NFSR exhaustive cycle.

The test sequence length is $(n + 1)2^n$. PEAT does not generally provide 100% detection of stuck-open faults. In addition, the problem of test sequence length is not much alleviated, and signature analysis is the only option for response evaluation. A more systematic technique to generate all adjacency tests is presented in [186]. However, the hardware required to generate such a sequence is much larger than in the more intuitive PEAT approach.

4.5.2 Test Sequence Generation Using Shift Registers

Most publicised BIST approaches use a feedback shift register, in one form or another, because of their small size, compared to an equivalent binary counter for example. Another advantage of an FSR is that it can easily be included in a scan path chain. Furthermore, in the case of LFSRs, no design effort is involved since the feedback functions are tabulated in most text books [92, 102], and even the conversion of an LFSR into an NFSR to obtain an exhaustive sequence is quite simple. In this subsection, we investigate the use of FSRs in generating test sequences for CMOS circuits.

A technique that has nearly the same fault detection capability as PEAT is proposed as follows: The test sequence generated by an FSR is repeated twice; the first time, every output from the FSR is preceded by a fixed arbitrary 1-vertex of the function under test, and the second time, it is preceded by a fixed arbitrary 0-vertex of the function.

In a single level combinational circuit, all stuck-open faults in the pull-down network are detected in the first cycle, while the second cycle is used to detect all stuck-open faults in the pull-up network. In multi-level circuits, this technique does not detect all stuck-open faults. However, if the multi-level circuit is testable with McCluskey's simplified two-pattern tests [107] then the proposed technique detects all stuck-open faults.

The length of the sequence 2^{n+1} for an n-input circuit, which is shorter than the lengths of 2^{2n} and $(n + 1)2^n$ corresponding to PETT and PEAT, respectively. Furthermore, the size of the hardware required for the sequence generator is of the same order as that required by PEAT and PETT (approximately, $2n$-bit latches) as shown in Fig. 4.19.

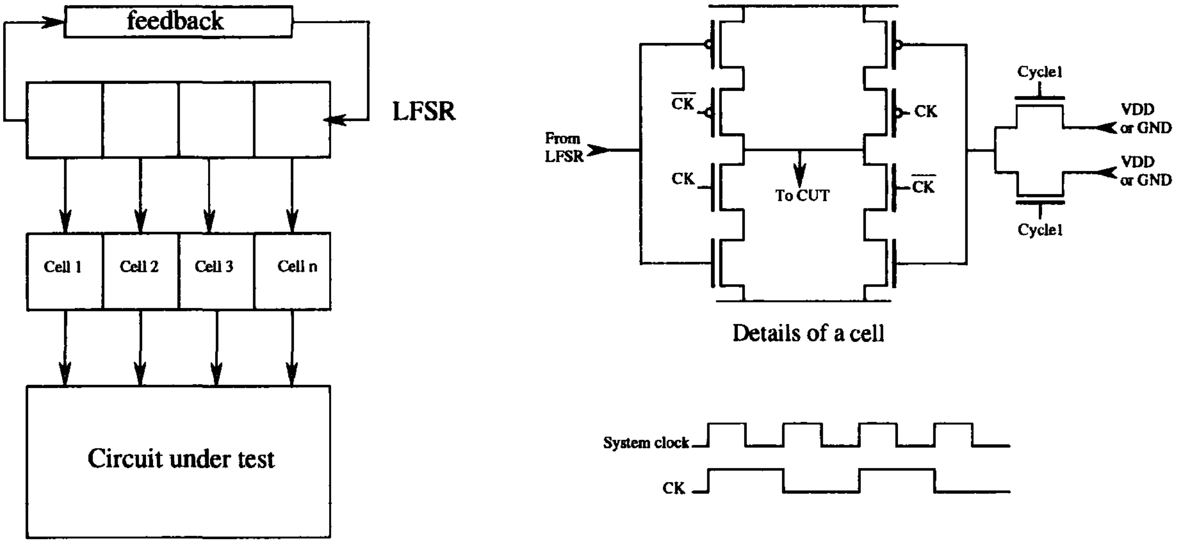


Figure 4.19 Alternative to PEAT and PETT.

All the techniques that use FSRs, discussed so far, generate a given sequence as a subset of a much longer sequence. For example, a 9-vector long, 3-bit wide test sequence for a one bit adder is generated as a subset of a 16-vector sequence in the approach of Fig. 4.19, and as a subset of a 24 and 64 vector sequences in the case of PEAT and PETT, respectively.

In the following, we consider a procedure presented in [187] that finds the shortest possible FSR that generates a given 1-bit wide sequence. Hence, for a circuit with n inputs, each input is generated by a separate FSR.

Let m be the length of the 1-bit sequence to be generated and k the length of the shortest possible FSR that generates it. Initially, the first k bits of the sequence are in the register, as shown in Fig. 4.20 (a). The feedback network must generate b_{k+1} to be fed to the last stage on the next clock cycle, as shown in Fig. 4.20 (b), then the

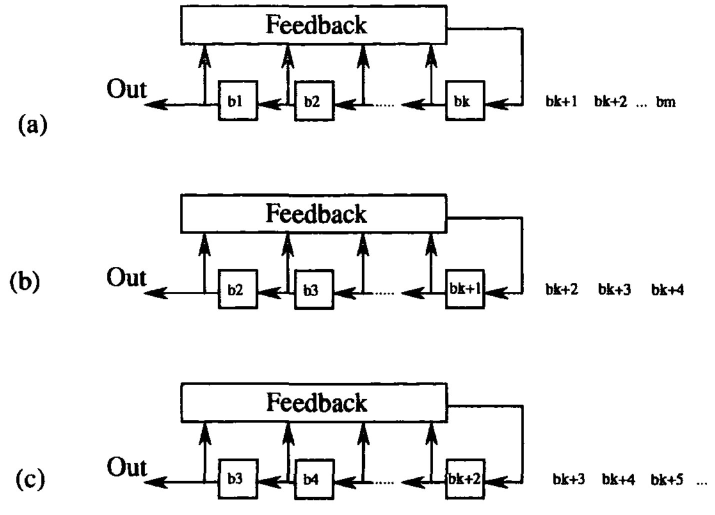


Figure 4.20 Illustration of the specification of the feedback function.

feedback must generate b_{k+2} which is fed to the last stage of the FSR as in Fig. 4.20(c), and so on.

The procedure is illustrated with the design of a test sequence generator for a 1-bit adder. The required test sequence is shown below. Inputs A, B, and C are generated by three separate shift registers. Input A cannot be generated from a single shift register since it is not an alternating or constant signal. A two-stage FSR also cannot generate input A since the truth table of the feedback function contains inconsistencies because the sub-sequence 00 is followed by 0 at vector 4 and then it is followed by a 1 at vector 9. With a three stage FSR, it is possible to generate input A. The truth table of the feedback circuit is shown below.

Applying the same procedure to the other inputs results in the test sequence generator shown in Fig. 4.21. The total number of transistors for this realisation is 126 compared with 101 for a ring counter + ROM implementation. The difference is not substantial. The drawback with this approach is that the number of latches required to generate a given sequence is dependent on the pattern of 1's and 0's. It can be shown that for arbitrary sequences of length m, the average length of the FSR tends to $m/2$. The testing of a 4-bit adder requires 15 9-bit vectors. This requires approximately 63 latches and some combinational logic, compared with 15 latches and a ROM array.

The generation of each input with a separate FSR results in a large sequence generator. Daehn et. al. [188] present a similar procedure for generating a predefined set of vectors as a subset of a larger set, but from a single FSR. The input vectors are ordered in such a way that they can be generated by a shift register. This requires the insertion

Test	Sequence		
	A	B	C
1	0	0	1
2	0	1	1
3	0	1	0
4	1	1	0
5	1	0	0
6	1	0	1
7	0	0	1
8	0	0	0
9	1	1	1

Truth table of the feedback circuit

	b1	b2	b3	Out
1	0	0	0	1
2	0	0	1	1
3	0	1	1	1
4	1	1	1	0
5	1	1	0	0
6	1	0	0	1
1	0	0	1	x

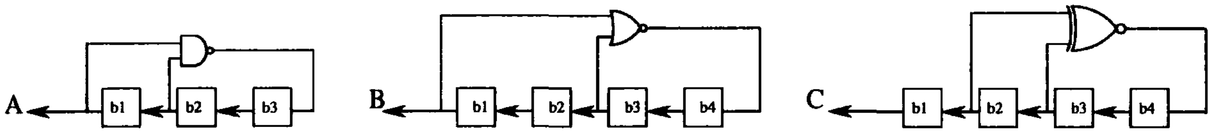


Figure 4.21 Feedback shift register implementation of the test sequence generator.

of redundant link vectors to make the shift operation possible. The feedback circuit is determined as in the previous procedure.

Note that the initial ordering of test vectors is not preserved. The initial sequence is further deteriorated with the insertion of link vectors. In addition successive states of the FSR do not always represent valid tests for stuck-open faults. For example, considering a 1-bit adder, the input vector ABC=011 can be obtained from only two possible right-shift operation:

(1) 111 → 011 or (2) 110 → 011

In (1), the carry circuit is not tested, whereas in (2) both the carry and sum circuits are not tested. If inputs A, B, and C were generated by non-contiguous stages of a shift register, the above problem does not occur. Therefore, as an adaptation to CMOS stuck-open faults, the procedure of Daehn et. al. [188] is extended by requiring that dummy stages be inserted in the shift register so that each vector has the appropriate initialisation. Table 4.11 illustrates the application of this method to the testing of a 1-bit adder cell.

Table 4.11 Illustration of the generation of sequences by a single FSR.

	d1	**A**	d0	**B**	**C**	F
1	0	**0**	1	**0**	**0**	1
2	1	**0**	0	**1**	**0**	1
3	1	**1**	0	**0**	**1**	0
4	0	**1**	1	**0**	**0**	0
5	0	**0**	1	**1**	**0**	0
6	0	**0**	0	**1**	**1**	0
7	0	**0**	0	**0**	**1**	1
8	1	**0**	0	**0**	**0**	1
9	1	**1**	0	**0**	**0**	1
10	1	**1**	1	**0**	**0**	1
11	1	**1**	1	**1**	**0**	0
12	0	**1**	1	**1**	**1**	1
13	1	**0**	1	**1**	**1**	1
14	1	**1**	0	**1**	**1**	0
15	0	**1**	1	**0**	**1**	x
16	x	**0**	1	**1**	**0**	x

d0 and d1 are the two dummy stages and F is the output of the feedback function. The size of such a test sequence generator is 78 transistors which is less than the 101 transistors required for a ring counter + ROM implementation. However, the number of redundant vectors and dummy stages is again very dependent on the pattern of 0's and 1's in the initial sequence. Furthermore, this approach suffers from the same problem as PETT and PEAT, as far as response analysis is concerned, but it is better, in terms of hardware overhead and test sequence length. The only advantage of PEAT and PETT over this method is that no design is required. However, the design procedure can easily be programmed.

4.5.3 Finite State Machine Implementation of the Test Sequence Generator

An FSM that generates a 9-vector long test sequence that would completely test a 1-bit full adder cell can be implemented as a PLA and four bistables as shown in Fig. 4.22. Without going into the details of the implementation, it can be seen that the part of the OR-plane of the PLA that generates signals A, B, and C has the same size as the ROM array in the ring counter + ROM implementation. Therefore the FSM implementation does not seem to offer any advantages over the initial implementation in terms of hardware overhead. Furthermore, the problem of testing the tester is worse than in the case of a ROM array. Testing a structure as in Fig. 4.22 was the subject of a whole PhD thesis [189].

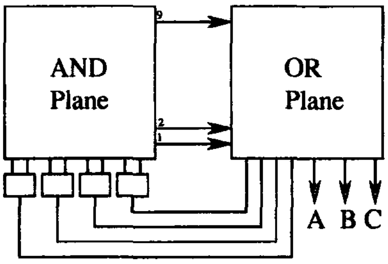


Figure 4.22 Finite state machine implementation of the test sequence generator.

A more intuitive design of a circuit that generates a predefined sequence of m vectors is shown in Fig. 4.23. It uses a $\lceil \log_2(m) \rceil$ LFSR (or NFSR if $m = 2^n$) and some combinational logic to modulate the output of the FSR in order to obtain the required sequence.

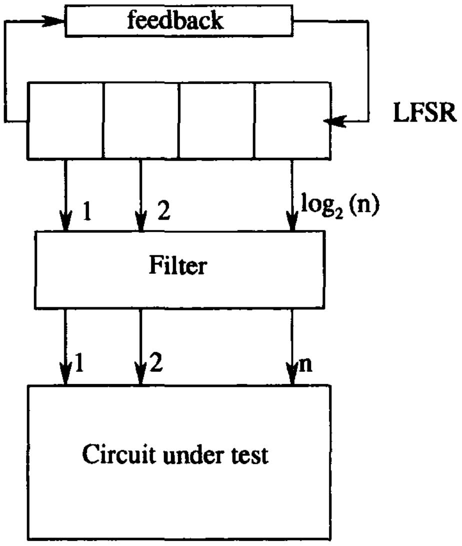


Figure 4.23 Another FSR implementation of the test sequence generator.

One of the advantages of the above structure is that the testing of multiple circuits does not require another LFSR, i.e., the same LFSR can be used to test all of them. In this case the length of the LFSR is determined by the longest test sequence required. Each circuit under test requires its own 'filter'. This is believed to be a promising approach towards the implementation of distributed BIST, especially if the filters could be designed so that they are tested while performing their function.

The idea of using a central LFSR in conjunction with distributed filters to test multiple CUTs is easily extended to the initial implementation of the test sequence generator. It is conceivable to have a single ring counter for the whole chip and a ROM array for each CUT. However, the requirement to route all the outputs of the ring counter to all CUTs is very likely to be an expensive proposition. With a centralised LFSR, the number of signals to be routed is much smaller.

4.6 TIME REDUNDANCY FOR FAULT DETECTION

The difficulty in testing CMOS stuck-open faults, compared to stuck-at faults, lies in the detection of high impedance states. The work presented in Chapter 3 and the previous sections of the present chapter is based on the standard approach of using two-pattern tests for detecting stuck-open faults. A different method, based on the work of McCluskey et. al. [81], is investigated in this section.

The C/D testing approach for detecting stuck-open faults [81] is illustrated in Fig. 4.24. The technique consists of charging and discharging the gate outputs after the application of each test pattern using inputs TEST and C/D. If the output line is not in a high impedance state, the output logic value will return to its correct value after the charging or discharging is ceased. If the line is at a high impedance state, it will remain charged or discharged.

An improved version of the C/D technique is proposed in Fig. 4.25. The number of extra transistors is doubled but the number of extra input is the same as in Fig. 4.24. Two successive and non-overlapping pulses are applied to the pFETS and nFETS after the application of any input to the circuit. The fact that a node that is in a high impedance state is charged or discharged much more quickly than a node that is driven to the same logic value may be used to set the width of the pulses so that it is large enough to charge or discharge a high impedance node and so that it is too short for perturbing the logic value of a driven node. In this way the presence of a stuck-open fault is indicated by any change in the logic value of the node under observation, but stuck-at faults are not detected.

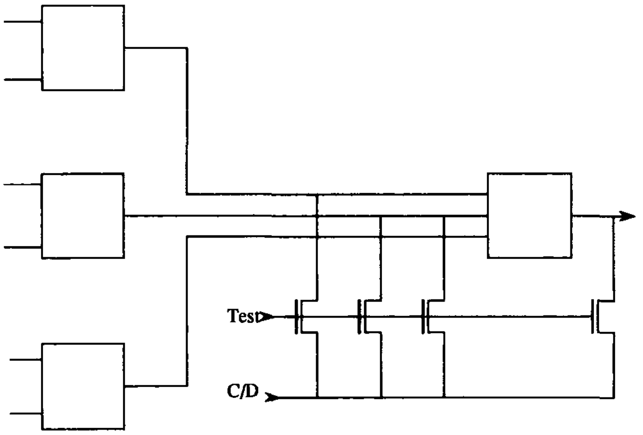


Figure 4.24 C/D testing approach.

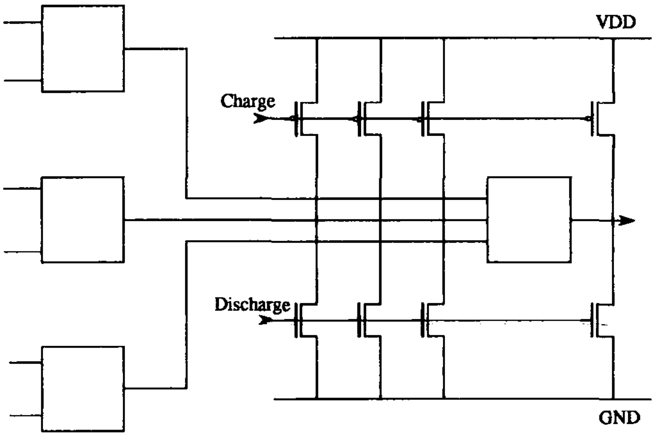


Figure 4.25 Improved version of the C/D testing approach.

If the output node is at logic 0 then it is not necessary to attempt to discharge it. Similarly, it is not necessary to charge a node that is already at logic 1. This observation leads to the new method proposed in Fig. 4.26. Depending on the value of the output at time i, a 1-vertex or a 0-vertex is applied to the CUT to charge or discharge the output at time $i + 1$. At time $i + 2$ the normal input to the CUT is re-applied. If the CUT is fault-free, its output will be $1 \rightarrow 0 \rightarrow 1$ or $0 \rightarrow 1 \rightarrow 0$. Any other output pattern indicates the presence of a fault.

In single-level combinational circuits, all stuck-open faults can be detected, whereas in multi-level circuits, the detection of all stuck-open faults requires that the circuit be testable with McCluskey simplified two-pattern tests. If the CUT has a single output that a multiplexer is used to select between the 0 or 1 vertex, but for a multi-output circuit, the multiplexer need to be replaced by a larger combinational circuit that select

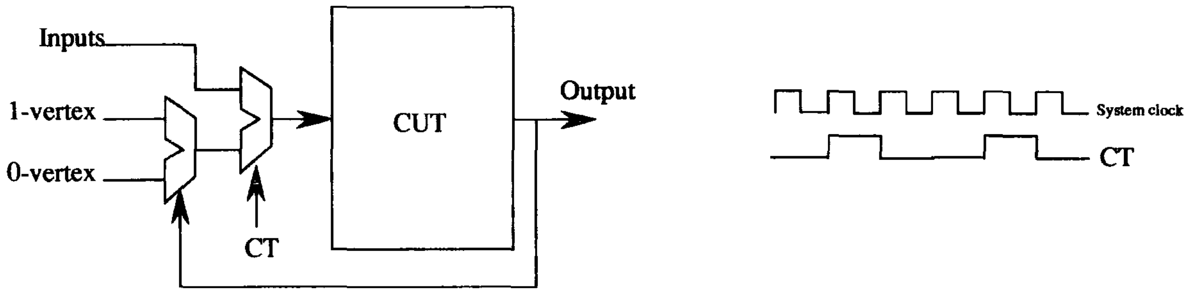


Figure 4.26 Time redundancy approach.

the appropriate vertex for time $i+1$ according to the output pattern at time i. For single-level circuits, the multiplexers can be integrated into the gate as shown in Fig. 4.27.

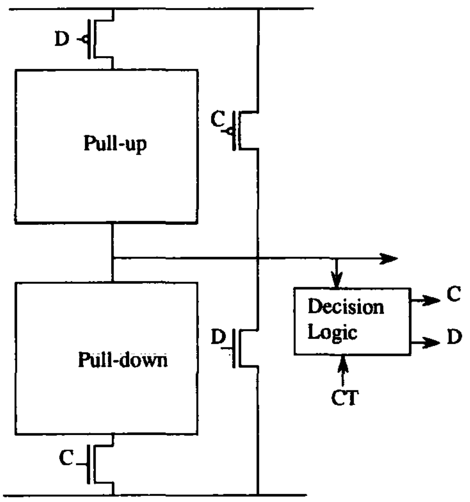


Figure 4.27 Time redundancy approach for a single level circuit.

The inputs to the CUT in Fig. 4.26 do not have to be test inputs. Therefore, the method can also be used for on-line fault-detection if the normal clock cycle is extended.

4.7 CHAPTER SUMMARY

This chapter has considered the possibility of using BIST for the detection of all faults in CMOS circuits. BIST offers many advantages, even outside the framework of fault-tolerance. BIST may also become mandatory in highly integrated circuits of the near future, as evidenced by the limited use of BIST features in current state of the art chips.

Most current BIST approaches are based on pseudo-random or pseudo-exhaustive testing. They do not aim at detecting all faults either because of the test patterns used and/or because of the compression of the test response into a signature. Furthermore, the detection of non-classical faults is not addressed. The principal objective of the BIST design presented in Section 4.2 was the detection of all faults in the functional circuit. Yet, the hardware overhead is comparable to that associated with pseudo-random and pseudo-exhaustive testing.

In Section 4.3, the major issue of testing the test circuitry was addressed. It was concluded that improved test coverage may be achieved through the cooperation of different chips in testing each other's untested parts. An effective approach towards complete test coverage would be to design the test circuitry so that it is tested while performing its function. Faults in the functional hardware will be indistinguishable from faults in the test hardware, but this does not present any problems if the test hardware is distributed so that each functional unit has its own test circuitry.

Considering the issue of partitioning the circuit into functional units, it is shown in Section 4.4 that there exists an optimum size of the units for which the introduction of BIST yields the minimum hardware overhead. This issue will be addressed again in Chapter 5.

The test sequences used for the detection of all faults in the functional circuits have been shown to require a precise ordering. They are generated by a combination of a ring counter and a ROM array. Although this is the simplest possible implementation, it creates problems in testing the ROM array. Alternative implementations are investigated in Section 4.5. Some implementations proposed in the literature are adapted for the detection of CMOS stuck-open faults. In addition, new implementations are also proposed in Section 4.5.

A completely different approach to detect stuck-open faults was presented in Section 4.6. The method relies on the use of extra time in order to exercise the circuit in such a way that the presence of faults becomes evident. This method could also be used for on-line fault detection.

Chapter 5
Yield and Reliability Modelling for Fault-Tolerant VLSI Circuits

5.1 INTRODUCTION

Fault-tolerance is achieved by including redundancy in an integrated circuit. One obvious disadvantage is that the addition of redundant units to an integrated circuit increases its area, making it more susceptible to manufacturing defects. The susceptibility to operational failures is also increased since there are more devices that might possibly fail. On the other hand, the addition of redundant units also enables the chip to survive some of the manufacturing defects and operational failures. The question of whether the introduction of redundancy can lead to the manufacture of more working integrated circuits and/or whether these ICs have a longer life time can only be answered by comparing the yield and the reliability of the fault-tolerant and non-fault-tolerant ICs. This chapter is devoted to the development of new yield and reliability models for performing such comparisons.

Yield models for fault-tolerant ICs are based on similar models for non-fault-tolerant ICs, whereas the reliability models seem to be based on those developed for large fault-tolerant systems. This will become apparent in the discussion of Section 5.2 which contains a brief review of modelling techniques.

The best method for adding redundancy into a chip depends on the architecture of the chip. Completely different redundancy schemes may be cost-effective for different architectures. Section 5.3 presents a simple fault-tolerance strategy for the detailed development of the yield and reliability models in Section 5.4. This strategy makes the least assumptions on the architecture of the initial design. Other assumptions necessary for the development of the models are also listed in Section 5.3.

In a fault-tolerant IC, the dependence of reliability on manufacturing yield is much stronger than in the non-fault-tolerant case, since some of the redundancy may have to be used to tolerate processing defects so that less is available for substituting for operational failures. The reliability model developed in Section 5.4 takes this dependence into account. It also allows for the important effect of off-line periodic testing.

Section 5.5 illustrates the application of the models and presents some interesting results concerning, among other things, the requirements for the size of the reconfiguration logic and the effect of periodic testing on reliability. Section 5.6 concludes the Chapter.

5.2 REVIEW OF MODELLING TECHNIQUES

Integrated circuit yield modelling, because of its crucial role in the profitability of a manufacturing process, has received wide attention since the early sixties. In contrast, reliability models have been confined largely to the study of system level reliability.

The numerous yield modelling techniques are best classified into models for fault-tolerant and non-fault-tolerant ICs for the following reason. Yield models for non-fault-tolerant ICs are developed using statistical methods where the main concern is the determination of probability distribution functions and their parameters to give the best fit to the observed yield data. Yield models for fault-tolerant ICs, on the other hand, are developed by chip designers whose main concern is the analysis of the effect of a particular fault-tolerance scheme on yield and the selection of the design decisions that maximise yield.

The two classes of yield models are discussed in the next two sections. This is followed by a review of reliability modelling techniques.

5.2.1 Yield Modelling for Non-Fault-Tolerant Integrated Circuits

Yield (or yield loss) is defined as the fraction of chips that pass (or fail) the final test. In many cases, it is important to have a formula to predict the expected yield before actually manufacturing the chips, as when a new product is designed. All research on yield modelling has been directed towards finding such formulas.

Yield losses are caused by randomly occurring manufacturing defects. In the earliest yield models, the random defects were assumed to be uniformly distributed over the chip or wafer areas. A uniform distribution of random defects can be described by a Poisson process, where the probability of having k defects in an area A is given by

$$P_k = e^{-AD}\frac{(AD)^k}{k!}$$

and the yield is

$$Y = P_0 = e^{-AD} \tag{5.1}$$

where D is the mean defect density per unit area and the quantity AD is the average number of defects per chip.

The early use of (5.1) revealed that the projected yields were always lower than the observed yields. Furthermore, the average defect density was found to vary from chip to chip and from wafer to wafer, suggesting that the random distribution of defects is not uniform across a wafer or a batch of wafers.

The effect of non-uniform defect distributions was introduced into the yield models by Murphy [155]. He proposed that the average defect density D in (5.1) be considered a random variable that is characterised by a probability density function $f(D)$ (The probability density function is referred to as simply distribution function in the literature and in this chapter). The yield formula becomes the average, or expected value, of the quantity e^{-AD}, given by

$$Y = \int_0^{+\infty} e^{-AD} f(D)dD$$

In the following years, most work on yield modelling was concentrated on investigating different defect distribution functions. There was a much heated debate in the seventies over which distribution is the most appropriate [190, 191, 192, 193, 194, 195, 196]. The different distribution functions and the corresponding yield expressions are listed in Table 5.1

Table 5.1 Defect distribution function and corresponding yield expressions.

Distribution Function	Yield Expression
Delta	e^{-AD_0}
Triangular	$(\frac{1-e^{-AD_0}}{AD_0})^2$
Rectangular	$\frac{1-e^{-2AD_0}}{2AD_0}$
Exponential	$\frac{1}{1+AD_0}$
Gamma	$(1+\frac{AD_0}{\alpha})^{-\alpha}$

This debate seems to have been silenced by the persistence shown by Stapper [193, 197, 195, 156] in proposing the Gamma distribution as the most appropriate; his justification being that, with a proper choice of the distribution parameters, it can fit any yield data. He illustrated the parameter fitting procedure using yield data from different IBM processing plants.

The Gamma distribution is undoubtedly the most flexible defect distribution function: by varying its two parameters, it can be made to coincide with any other distribution function used in yield models [198]. However, the assertion that it is the most physical distribution [198] is questionable if we bear in mind thatthe Gamma distribution was initially developed for the study of infectious diseases.

As a further justification of the appropriateness of the Gamma distribution, Stapper [156] also developed a physical interpretation of the distribution parameter α: it is a coefficient that reflects the severity of the defect clustering phenomenon.

It should be noted that the clustering of defects is implicitly taken into account by any non-uniform defect distribution. The non-uniformity of defect distributions means that some regions, or some wafers, receive more defects than others, i.e., the defects cluster in these areas. The Gamma distribution accounts for defect clustering in an explicit manner, through the parameter α. Other explicit methods have been proposed [199]. The quasi-universality of the Gamma distribution in accounting for the clustering of defects is even challenged by Ferris-Prabhu [200] who is also from IBM and, presumably, has access to the same yield data as Stapper.

Numerous types of defects can be introduced during the manufacturing process, as discussed in Chapter 2. This is taken into account by assuming that the defects are independent so that the yield can be expressed as a product of partial yields corresponding to different defect types. That is,

$$Y = \prod_{i=1}^{n} Y_i, \quad \text{and} \quad Y_i = \int_0^{+\infty} e^{-A_i D_i} f(D_i) dD_i \tag{5.2}$$

where D_i and $f(D_i)$ are the defect density and its distribution function, respectively, and A_i the area susceptible to defect type i. Expression (5.2) is usually multiplied by the gross yield Y_0 which represents the losses due to global defects.

The notion of susceptible area, and its relation to chip area, was not clarified until the early seventies [197]. Prior to that it had the meaning of 'the area where a manufacturing defect would cause a fault' and it was clear that it should be smaller than the chip area. For example, the area susceptible to oxide pinholes is the area where different conductors overlap. Gupta et.al. [201] assumed that susceptible area is proportional to the chip area. Stapper [197] prefers to call it *critical* area. He shows how it depends on the defect size, the circuit dimensions and the type of defects. The size of defects is also a random variable. Ferris-Prabhu [202, 203] gives analytical methods for computing the critical area.

Computing the critical area of a VLSI chip requires a complete layout of the circuits and it is a complex problem. This gave rise to the latest development in yield modelling. It is the emergence of a number of simulation tools that, given a circuit layout and information on the processing line (defect sizes, distribution, locations ...) as input, would give an estimate of the yield [204, 205, 157, 206, 158, 207]. These simulation tools can also pinpoint areas of the chip where defects are most likely to cause a fault. This move from analytical methods to simulation [207, 158] illustrates the complexity of the yield modelling problem.

5.2.2 Yield Modelling for Fault-Tolerant Integrated Circuits

As mentioned earlier, yield models for fault-tolerant ICs are usually developed by IC designers who are interested in comparing different fault-tolerance strategies, selecting the optimum amount of redundancy, and identifying the design decisions that influence yield. Given that yield modelling for non-fault-tolerant ICs is already a complex problem, systematic simplifying assumptions must be used in developing models for fault-tolerant ICs.

The fundamental expression to all yield models is the probability of having exactly M-out-of-N units defect-free. This is usually assumed to be

$$P_{M,N} = \binom{N}{M} y^M (1-y)^{N-M}$$

where y is the probability that a single unit is defect free and $\binom{N}{M} = \frac{N!}{M!(N-M)!}$. In a chip containing N identical units where M are required to be defect-free (i.e., $R = N - M$ are spare units), the yield would be

$$Y = \sum_{M=N-R}^{N} P_{M,N}$$

The problem in the above expression for $P_{M,N}$ is that it is assumed that the probability of having n defect-free units is $y_n = y^n$ which is true only if the defects affecting different units are independent. Note that, except for the Gamma distribution, there is no reason to invalidate the above assumption for any uniform or non-uniform defect distributions, except when the defects are very large. For the Gamma distribution, the underlying assumption is that a particular region of a chip or a wafer is more likely to be affected by defects if it already has defects in it. Therefore, defects affecting different units cannot be assumed to be independent when using the Gamma function.

An alternative expression for $P_{M,N}$ is given by

$$P_{M,N} = \sum_{x=0}^{+\infty} P\{x\} P_{x,R}^{N}$$

where $P\{x\}$ is the probability of having x defects in N units and $P_{x,R}^{N}$ is the probability that these defects are distributed exactly in $R = N - M$ units. This way of computing $P_{M,N}$ has been used in [208, 209, 210]. Although $P\{x\}$ is usually calculated according to some defect distribution functions to account for clustering or interdependence of defects, the probability $P_{x,R}^{N}$ is calculated using combinatorial methods that assume that the x defects are **uniformly** distributed among the N units, as pointed out in [211]. Furthermore, the question of whether manufacturing defects are distinguishable or not has to be addressed in calculating $P_{x,R}^{N}$. This was one of the debatable questions of the early seventies [190, 191]. Koren et. al. [212] show that the two methods of evaluating $P_{M,N}$ are equivalent when using the Poisson or the Gamma distributions.

The derivation of yield models for fault-tolerant ICs should not be too involved with statistical manipulations as in the case of non-fault-tolerant models (although IC designers have rarely access to yield data to do so, because of the commercial sensitivity of such data [2]) because this may distract from the main goals of these models. However, a minimum knowledge of the methods used to derive yield models for non-fault-tolerant ICs is required in order to extend them to fault-tolerant ICs. This is not always the case: the fundamental expression of $P_{M,N}$ is sometimes misused [213, 214].

The probability y of having a defect-free unit is taken to be given by one of the expressions of the second column of Table 5.1 in the expression $\binom{N}{M} y^{M} (1 - y)^{N-M}$. This assumes a non-uniform distribution of defects within a unit only, and not among different units. This might be valid when the units are very large, but not when they are very small as in [214]. Ramacher [213] used the Gamma distribution to calculate y and then goes on studying the effect of defect clustering on her fault-tolerance scheme.

One of the most important goals of yield modelling for fault-tolerant ICs is the determination of the optimum amount of redundancy. In memory chips, it was observed that most defects affect a single bit or a single column [215], therefore, the addition of a few spare columns and/or rows was enough to reclaim a large proportion of the defective chips. The situation is likely to be the same in non-memory fault-tolerant chips so that only those chips having few manufacturing defects may be reclaimed and those with a large number of random defects are likely to defeat any fault-tolerance scheme. Therefore, only a small number of redundant units should be added. More redundancy may actually decrease the yield, because more redundancy implies more hardcore. The situation where all this redundancy is required rarely occurs, and if it does occur it is unlikely that bypassing faulty units would be possible. Hence, the yield would be expected to decrease beyond a certain amount of redundancy, and not saturate at a constant value as in [208, 214].

Another important issue is the determination of the optimum size of the unit that is to be replicated. Thibeault et. al. [216] show that, when the reconfiguration logic is neglected, the optimum size of the replaceable unit is zero. They then show that the size of the reconfiguration logic can be minimised by choosing an appropriate size of the replaceable unit. This suggests that the yield can also be maximised by choosing the unit size that minimises the size of the reconfiguration logic.

5.2.3 Reliability Modelling

Reliability is defined as the conditional probability that a system is working at time t, given that it was working at time $t = 0$. It is measured by observing N identical systems and noting the times at which they fail. The reliability at time t is the fraction of systems still working at t. As mentioned in Chapter 2, such measurements are virtually impossible. Furthermore, and as for yield, it is important to have estimates of reliability at design time, especially for safety critical applications. Reliability prediction at design time is even more important than yield prediction: yield losses have an impact on the cost of chips and the profitability of a production line, whereas the costs incurred as a result of reliability problems can be orders of magnitude higher (maintenance costs, human lives, environmental disasters, ... etc).

Another reason for the importance of reliability modelling of integrated circuits is the ever increasing levels of integration. The reliability of an electronic system comprising from a few tens to few hundreds of ICs is determined by the reliability of the solder joints and the interconnections between ICs [217]. However, when such a system is re-designed so that it comprises just one or two larger and denser chips, which is the current trend in the electronic industry, the system's reliability would certainly be improved, but at the same time it becomes strongly dependent on the reliability of the integrated circuit itself. Many papers on yield modelling for fault-tolerant ICs emphasize that fault-tolerance also improves the reliability since the larger area chips, made possible by fault-tolerance, reduce the chip count. Hence, reliability improvement is seen as a by-product of yield improvement.

The time at which a system or component fails is a random variable. It is usually assumed that the time between successive failures, in a sample of identical systems, follows an exponential distribution. That is, the probability that a system will have failed by time t is given by $F(t) = 1 - e^{-\lambda t}$ where λ is the failure rate. The reliability, or the probability that the system is still working at time t is $R(t) = 1 - F(t) = e^{-\lambda t}$. The underlying assumptions of the exponential distribution of times to failures are [218]:

- The likelihood of a single failure in a small interval of time $[t, t+h]$ is proportional to the length of the interval (and is independent of t).
- The likelihood of two or more failures during a given small interval is negligible.

In a system consisting of N units where all units have to be working the reliability of the system is the probability that all units are working. If unit i has failure rate λ_i, the reliability of the system is

$$R(t) = \prod_{i=1}^{N} e^{-\lambda_i t} \qquad (5.3)$$

Such a system is called a *series* system. If the units themselves consist of sub-units then their reliabilities are also expressed as a function of the reliabilities of the sub-units. This decomposition process ends when individual components are reached. The failure rates of individual components are normally provided by the manufacturers. The MIL-HDBK-217 [219] is a widely used alternative method for estimating the failure rate of components.

A *parallel* system consists of N identical units where only a single unit is required to be working. The reliability of such a system is given by

$$R(t) = \sum_{i=1}^{N} \binom{N}{i} e^{-i\lambda t}(1 - e^{-\lambda t})^{N-i} \qquad (5.4)$$

where λ is the failure rate of a single unit. Alternatively, we may compute the probability of failure $F(t) = 1 - R(t)$ which is the probability that all units have failed

$$F(t) = \prod_{i=1}^{N}(1 - e^{-\lambda_i t}) = (1 - e^{-\lambda t})^N$$

and the reliability is $R(t) = 1 - (1 - e^{-\lambda t})^N$.

These expressions are used extensively in modelling the reliability of fault-tolerant systems [220, 218]. In large systems consisting of series and parallel subsystems that are themselves in series or in parallel, the above expressions become very complex. Markov models [221] are used in these cases.

In the case where all units are part of a single chip, there are some problems in evaluating expressions (5.3) and (5.4). The first problem is the non-existence of any method for the estimation of the failure rate of a section of a chip. A simple and practical solution to this problem is presented in Section 5.4.3.

Another problem in the evaluation of expression (5.4), when all units are on a single chip, is that some of the N units may not be working as a result of manufacturing defects, transforming an N-unit parallel system into an $(N - k)$-unit parallel system, where k is the number of defective units. This clearly establishes the dependence of the reliability of a fault-tolerant chip on the manufacturing yield. In system-level reliability modelling, this dependency does not exist since the components are assumed to be defect-free, hence all the redundancy is initially available for tolerating field failures. The reliability model developed in Section 5.4 takes this dependency into account.

Publications on reliability modelling for fault-tolerant ICs are very scarce, as compared to the abundant literature on yield modelling. The subject has been touched upon in [222] and [223]. Koren et. al. [224, 225] present the only comprehensive study of the subject using Markov modelling techniques. A chip consisting of N units where $N - R$ units are required to be working can be represented by the Markov chain of Fig. 5.1 [224].

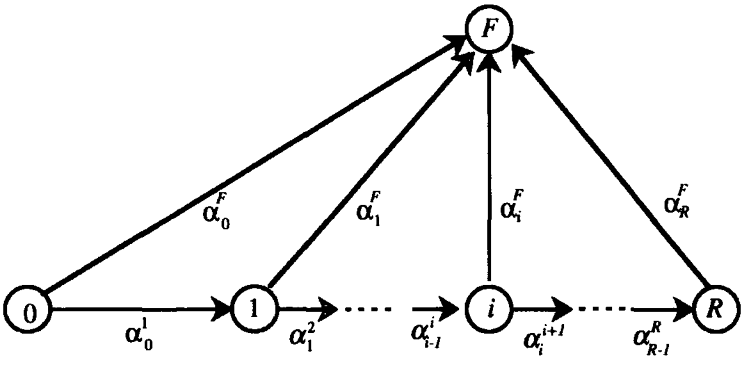


Figure 5.1 Markov model for an $(N - R)$-out-of-N system.

State i represents the case where the chip is operational with i failed units, $0 \leq i \leq R$. State F represents the failure of the chip either after the exhaustion of the spare units, or because of the impossibility of reconfiguring it after the i^{th} failure, $i < R$. The transition rate α_i^j from state i to state j is clearly a function of the failure rate of unit i which in turn is a function of the complexity of unit i. It is shown in [224] that transition rates α_i^j are also functions of the reconfiguration scheme.

State 0 in the Markov model of Fig. 5.1 is the initial state of the chip if all R spare units were defect free. The dependence of reliability on yield is introduced into the model by letting any state i, $0 \leq i \leq R$, be the initial state. The problem of estimating the failure rate of a unit is addressed by assuming a linear relationship between the failure rate of a unit and its area [224].

The work presented in this Chapter is basically an extension of the work by Koren et.al. [224, 225]. The main differences are as follows:

- The presented models are more tractable. This is achieved by avoiding Markov modelling which tend to become very complex when important parameters, discussed below, have to be taken into account.
- A concise method is used to estimate the failure rate of a section of a chip.
- The important question of 'What constitutes the best replaceable unit?' is easily addressed in the model presented.
- The effect of periodic testing on reliability, mentioned by Koren et. al. in [226] but not included in their models, is incorporated in our model in a simple way.
- The results that can be obtained from the models presented in this chapter are more practical since the areas of the units and the average defect density are used as inputs to the model, instead of the average number of faults per unit or per chip used by Koren. Working with average number of faults requires a further, and very complex processing step [197, 227] to get the areas of the units.

5.3 CHIP MODEL AND ASSUMPTIONS

There are numerous approaches for introducing redundancy into a design. As discussed in Chapter 2, these approaches belong to two classes. In the first class are those approaches that rely on hard-reconfiguration (or physical restructuring) and which are aimed principally at tolerating manufacturing defects. In addition to the requirement of a non-standard process, these approaches have very limited capabilities of dealing with operational failures.

The approaches belonging to the second class rely on soft-reconfiguration where the switching mechanism consists of ordinary MOS transistors. These approaches have the advantages of using a standard manufacturing process and of being capable of dealing with operational failures. The focus of this chapter is on the second class of approaches.

Among the various methods that rely on soft-reconfiguration for introducing redundancy, the most appropriate method is clearly dependent on the architecture of the non-fault-tolerant chip. In memory arrays, the introduction of redundancy in the form of spare rows and columns has proved cost-effective (although this is also due to the use physical restructuring). The same form of redundancy is being investigated for 2-D arrays of identical PEs, but the replaceable unit is a single PE in this case, as opposed to whole rows and columns in memory arrays, because even the simplest PE is substantially larger than a memory cell.

5.3.1 Redundancy Strategy

Any integrated circuit can be considered as a set of interconnected functional units. The IC design process itself starts by dividing the function of the chip into a number of smaller functional units. This fact can be used to propose a simple redundancy strategy, for the purpose of this Chapter, that makes the least assumptions about the architecture of the non-redundant chip, namely a chip of area A_0 is considered as consisting of n units of areas $A_1, A_2, \ldots, A_n$. Redundancy is introduced by replicating each unit, i, m_i times and adding some circuitry of area S_i for test and reconfiguration as shown in Fig. 5.2. The set of m_i units and their test/switch circuits will be referred to as a redundant unit or a redundant module.

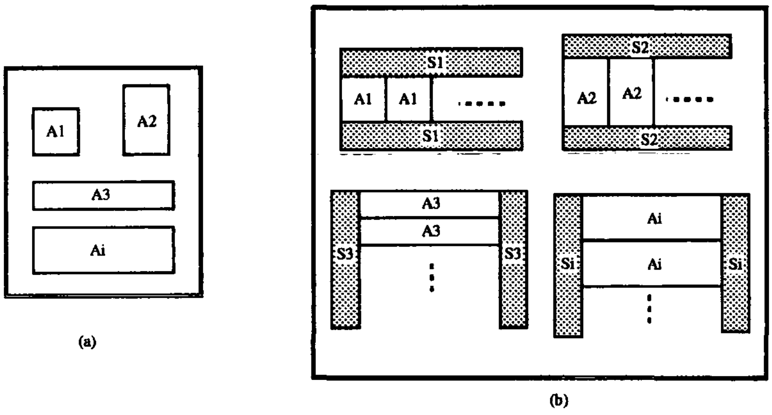


Figure 5.2 Non-redundant (a) and redundant (b) chips.

Each section, and its replicates, are assumed to be associated with a flag register. At power-up or at predefined time points, the chip enters a test mode where all sections are tested and their flags set according to the test outcome (faulty/working). These flag registers will drive the switches to connect working units and disconnect faulty ones.

5.3.2 Reconfiguration Logic Area

Figure 5.3 shows that the area S_i of the test/switch circuits is proportional to the replication factor m_i. The area of the block labelled 'switches' depends only on the number of terminals of unit i. Another observation from Fig. 5.3 is that if the faults that may affect the switches and/or flags of unit A_{i_j} do not affect the switches and flags of the other units, then the replaceable items become the whole of a unit and its test/switch circuits. This would have interesting consequences if it could be implemented, since the redundant unit can still be working even with faults in the test/switch circuits, i.e., the test/switch circuitry is not a hardcore any more. However, this would require that these circuits be designed so that they are fail-safe for all possible faults, which is not an easy task. Furthermore, a redundant unit also requires some global circuitry for selecting a single unit when there is more than one working unit.

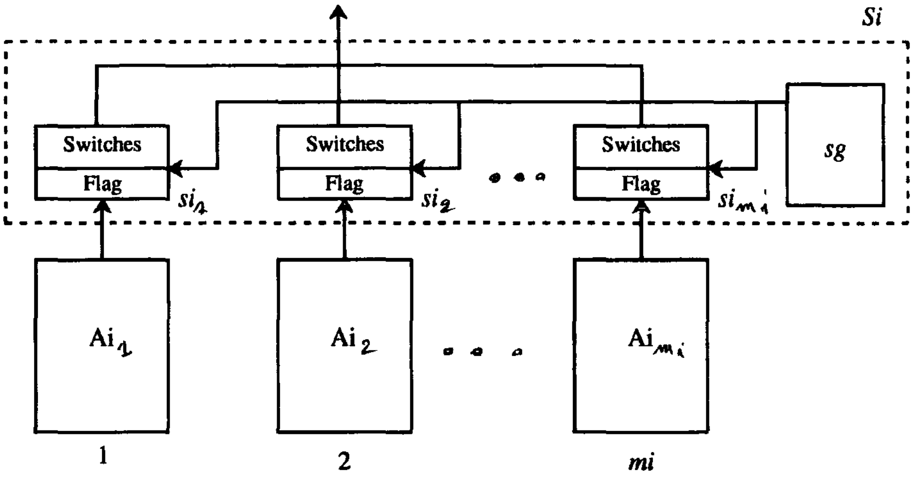


Figure 5.3 Relation between the number of spares and the size of the test/switching circuits.

An estimate of the area required for the test/switch circuits is derived from Fig. 5.3. We have $S_i = sg_i + s_{i_1} + s_{i_2} + \ldots + s_{i_{m_i}}$ where sg_i is the area of the global portion of the test/switch circuits. The area sg_i can be expressed as $sg_i = \alpha_i(A_1 + A_2 + \ldots + A_{m_i}) = \alpha_i m_i A_i$. Similarly, we can express s_{i_j} as $s_{i_j} = \beta_i A_{i_j}$. Hence,

$$S_i = m_i\alpha_i A_i + m_i\beta_i A_i = m_i(\alpha_i + \beta_i)A_i = m_i c_i A_i$$

5.3.3 Defect Distribution

The discussion over which defect density distribution is the most appropriate is well beyond the scope of any PhD thesis, and it could only be the subject of a strictly internal report for one of the large semiconductor manufacturers.

For the purpose of this Chapter, we assume that the defect distribution follows the triangular distribution suggested by Murphy [155] as an approximation to the bell-shaped curve followed by his yield data. Although the triangular distribution has not been used as extensively as the Gamma distribution in the literature, recent data from manufacturing lines at Plessey [228] do follow this distribution. Furthermore, Stapper [156] considers that the triangular distribution is likely to predict yields that are lower than observed yields. Therefore, the yield figures presented in the results sections of this Chapter can be regarded as lower bounds.

The other assumptions used in developing the yield model are as follows:

- The defect densities used are assumed to be the densities of fault-producing defects. It is clear that not all defects result in faults, and this depends on the type, size, and location of the defects and also on the precise layout of the circuits considered. The process of going from a defect density to a fault density is quite complex, as illustrated by Stapper [197, 227], and it requires precise information on the processing line in addition to a full layout.
- The defects are assumed to be small compared to the size of the replaceable units.
- The gross yield is assumed to be 100%.
- A single type of defects is considered.

5.4 NEW MODELS

In this section, yield and reliability models are developed under the assumptions of Section 5.3.

5.4.1 The Yield Model

The yield model for a fault-tolerant chip as shown in Fig. 5.2 is derived by computing the probability of having a working chip. Let

P_{A_i}: Probability of having no defects in a unit of area A_i.

P_{S_i}: Probability of having no defects in the reconfiguration logic of area S_i.

P_i: Probability that the redundant unit consisting of m_i units and their reconfiguration logic is working.

Redundant unit i is working if at least one out of the m_i units is defect-free and if the reconfiguration logic is defect-free. Therefore,

$$P_i = P_{S_i} \sum_{j=1}^{m_i} \binom{m_i}{j} P_{A_i}^j (1 - P_{A_i})^{m_i - j} = P_{S_i}(1 - (1 - P_{A_i})^{m_i}) \qquad (5.5)$$

The probability that the chip is working is given by the probability that all n redundant units are working, that is,

$$P_r = \prod_{i=1}^{n} P_i = \prod_{i=1}^{n} \left(P_{S_i}(1 - (1 - P_{A_i})^{m_i}) \right) \tag{5.6}$$

Probabilities P_{A_i} and P_{S_i} are given by e^{-A_iD} and e^{-S_iD}, respectively, where D is the defect density. Hence,

$$P_r(D) = \prod_{i=1}^{n} \left(e^{S_iD}(1 - (1 - e^{A_iD})^{m_i}) \right) \tag{5.7}$$

Since D is a random variable, it follows that $P_r(D)$, and also P_{A_i}, P_{S_i}, and P_i are random variables [221]. A random variable is completely characterised by its distribution or probability density function [221]. However, because it is not always possible to obtain accurate distributions, especially for functions of a random variable, other characteristics are usually employed. The characteristic that is used most is the expectation, also known as the mean or average, of a random variable. If $Z = \Phi(X)$ where X is a random variable with probability density function $f(X)$, the expected value of Z is given by

$$\overline{Z} = \int_0^{+\infty} \Phi(x) f(x) dx$$

This averaging process needs to be done only once. For example, if we compute the averages of P_{A_i} and P_{S_i} and use them in expression (5.5), then P_i is no longer a random variable. Similarly, if we compute the average of P_i and use it in expression (5.6) then P_r is not a random variable any more.

Selecting the appropriate stage at which the averaging process needs to be carried out is a subtle question that is rarely addressed in the literature. Section 5.4.1 discusses this question.

5.4.2 The Expectation of What?

The basic random variable is the defect density D. P_{A_i}, P_{S_i}, P_i, and P_r are random variables because they are expressed as a function of D. We may derive three different yield formulas for fault-tolerant chips depending on which average is considered.

Case 1

The average of P_{A_i} is given by

$$\overline{P_{A_i}} = \int_0^{+\infty} e^{-A_i D} f(D) dD = \Big(\frac{1 - e^{-A_i D_0}}{A_i D_0}\Big)^2$$

where $f(D)$ is the triangular distribution function of D and D_0 is the average defect density. Similarly, the average of P_{S_i} is

$$\overline{P_{S_i}} = \Big(\frac{1 - e^{-S_i D_0}}{S_i D_0}\Big)^2$$

The probability that redundant unit i is working is

$$P_i' = \overline{P_{S_i}}(1 - (1 - \overline{P_{A_i}})^{m_i})$$

and the first yield expression is

$$Y_{r1} = \prod_{i=1}^{n} P_i'$$

Case 2

The probability that redundant unit i is working is P_i with $P_{A_i} = e^{-A_i D}$ and $P_{S_i} = e^{-S_i D}$. The average of P_i is

$$\overline{P_i} = \int_0^{+\infty} \Big(e^{-S_i D}(1 - (1 - e^{-A_i D})^{m_i})\Big) f(D) dD$$

$$\overline{P_i} = \Big(\frac{1 - e^{-S_i D_0}}{S_i D_0}\Big)^2 - \sum_{j=0}^{m_i} \binom{m_i}{j} (-1)^j \Big(\frac{1 - e^{-(S_i + j A_i) D_0}}{(S_i + j A_i) D_0}\Big)^2$$

and the second yield expression is given by

$$Y_{r2} = \prod_{i=1}^{n} \overline{P_i}$$

Case 3

The third yield expression is obtained by taking the average of P_r, that is,

$$Y_{r3} = \int_0^{+\infty} \Big(\prod_{i=1}^{n} e^{-S_i D}(1 - (1 - e^{-A_i D})^{m_i})\Big) f(D) dD$$

The complete calculations of the three yield expressions are given in Appendix B. The question that arises is 'which of the three expressions is the most appropriate?'. The following discussion is an attempt to answer this question.

First, it is clear that the three yield expressions do not represent the same quantity, i.e., $Y_{r1} \neq Y_{r2}, Y_{r1} \neq Y_{r3}$ and $Y_{r2} \neq Y_{r3}$. A general proof of this is as follows. Let X be a random variable with distribution $f_X(X)$. Then $W = \Phi(X)$ and $Z = \Psi(W)$ are both random variables. The average of Z can be calculated as

$$\overline{Z} = \int \Psi(x) f_W(x) dx$$

if the distribution $f_W(x)$ is known. Alternatively,

$$\overline{Z} = \int \Psi(\Phi(x)) f_X(x) dx$$

The average of W is

$$\overline{W} = \int \Phi(x) f_X(x) dx$$

Proving that the three yield expressions are different is the same as proving that $\overline{Z} \neq \Psi(\overline{W})$, i.e.,

$$\int \Psi(\Phi(x)) f_X(x) dx \neq \Psi\Big(\int \Phi(x) f_X(x) dx\Big)$$

which is always true, except when Ψ is a linear function. $\triangle$

Case 1 is clearly the simplest. Furthermore, any distribution can be used with very little change. These are the reasons for its widespread use in yield models for fault-tolerant ICs. However, there are some problems in using Case 1. Taking the average of P_{A_i} and P_{S_i} assumes that the defects are randomly distributed within areas A_i and S_i only. Expressing P_i as a function of $\overline{P_{A_i}}$ and $\overline{P_{S_i}}$ does not take account of the fact that the defects are also randomly distributed, according to $f(D)$, among the m_i units and their test/switch circuits. A further disadvantage of the method of Case 1 is that it leads to the following contradiction. The yield of a non-fault-tolerant chip of area A_0 is given by $Y_0 = (\frac{1-e^{-A_0D_0}}{A_0D_0})^2$. If the chip is considered as a set of n sections of areas A_0/n, then the probability that a section is defect-free is $p_s = e^{-\frac{A_0}{n}D}$. According to the method of Case 1, the yield is

$$Y_0' = (\overline{p_s})^n = \left[\left(\frac{1 - e^{-\frac{A_0D_0}{n}}}{A_0D_0/n}\right)^2\right]^n \tag{5.8}$$

which is clearly different from Y_0. If the method of Case 3 is used instead, then the probability that the chip is defect-free is

$$p_c = p_s^n = (e^{-\frac{A_0D}{n}})^n = e^{-A_0D}$$

and the yield is $\overline{p_c}$ which is Y_0.

The same argument can be used against Case 2: expressing $Y_{r2} = \prod_i \overline{P_i}$ assumes that the distribution of defects follows $f(D)$ within the redundant units. This does not take into account the fact that the defects are also distributed over the entire chip according to $f(D)$.

Another way to argue against Cases 1 and 2 is to note that expressions P_{A_i}, P_{S_i} and P_i are just intermediate steps in obtaining the expression for P_r. The yield is the expected value of P_r. Hence, it is the expression for P_r that should be averaged. This makes Case 3 the most valid.

Figure 5.4 shows that the divergence between Y_0 and Y_0' increases as the area of a section gets smaller (i.e., as n increases). It can be shown that

$$\lim_{n \to \infty} Y_0' = e^{-A_0 D_0}$$

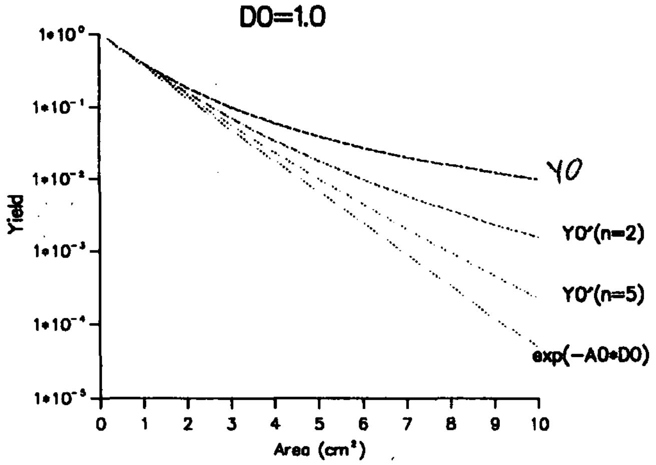


Figure 5.4 Comparison of Y_0 and Y_0'.

However, the difference between Y_0 and Y_0' is not very substantial. Figure 5.5 shows how the difference $\Delta_Y = Y_0 - Y_0'$ varies with the chip area in the limiting case (as $n \to \infty$). The upper bound on the absolute error is around -0.05. Although the error Δ_Y tends to 0 as the area A_0 becomes large, the curve of the ratio Y_0/Y_0' shows that Y_0 is many orders of magnitude higher than Y_0', but in this case both Y_0 and Y_0' are approximately zero. Hence, Y_0' is a good approximation for Y_0 for practical values of A_0 and D_0.

Figure 5.6 shows that Y_{r1} and Y_{r2} are also good approximations for Y_{r3} for practical values of the area of the non-redundant chip.

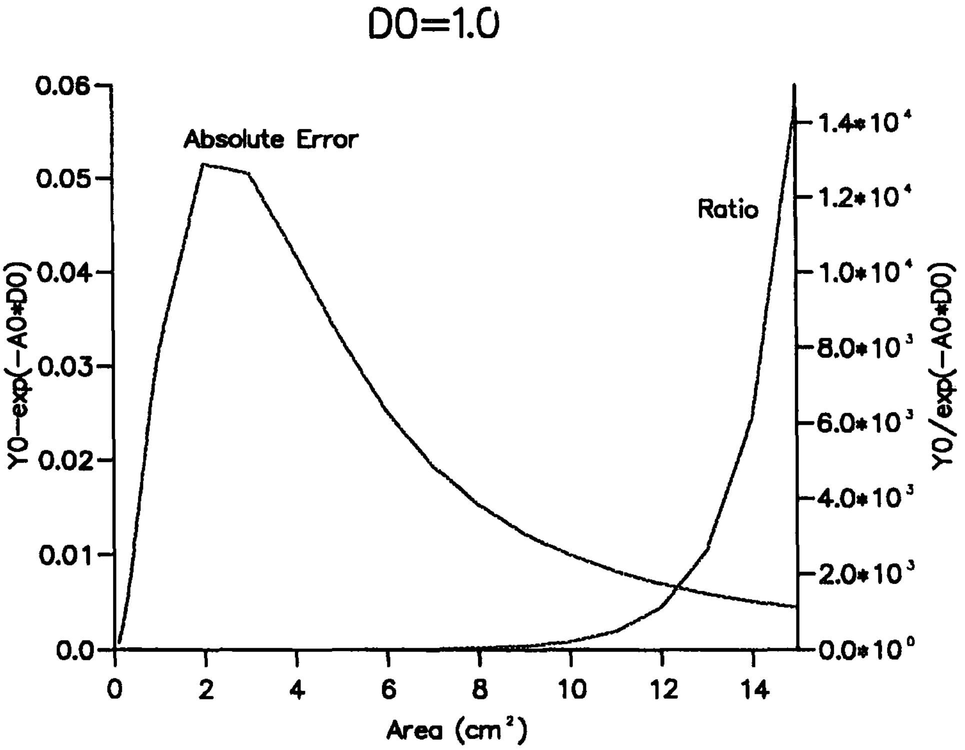


Figure 5.5 Maximum absolute error.

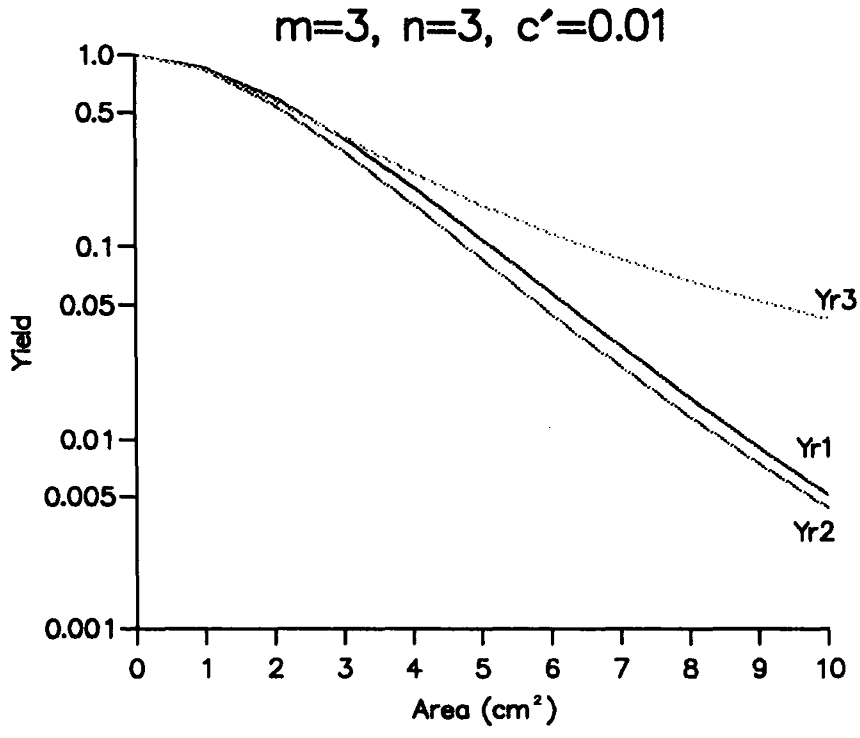


Figure 5.6 Comparison of the three cases.

5.4.3 Failure Rate of a Section of a Chip

The failure rate of a section of a chip is expressed as a fraction of the failure rate of the entire chip. The justification of this approach is given next.

The reliability of an IC is given by $R_0 = e^{-\lambda_0 t}$. If the same chip is considered as a partition of n sections each comprising N_0/n gates, then we may compute the reliability of the chip as the product of the probabilities that each section is working, i.e., if we call λ_0' the failure rate of a section containing N_0/n gates, then the reliability of the chip is

$$R_0' = (e^{-\lambda_0' t})^n = e^{-n\lambda_0' t}$$

Since R_0 and R_0' must be the same, we have $\lambda_0 = n\lambda_0'$. If we let $L = N_0/n$ then

$$\lambda_0' = \frac{L}{N_0}\lambda_0$$

which is the failure rate of a piece of silicon containing L gates that is part of an IC containing N_0 gates. This justifies the assumption made in [224].

An estimate of the failure rate λ_0 can be obtained from the MIL-HDBK-217 [219], where λ_0 is expressed as a function of many parameters, including the number of gates in the chip. The details of the estimation of the failure rate are given in Section B.3 of Appendix B.

5.4.4 The Reliability Model

The failure rate of a non-fault-tolerant chip is $\lambda_0 = f(N_0)$, where N_0 is the number of gates, and the reliability is $R_0(t) = e^{-\lambda_0 t}$. Similarly, and according to MIL-HDBK-217, the failure rate λ_r of the fault-tolerant chip can also be expressed as a function of the number of gates N_r, and we have $\lambda_r > \lambda_0$ since $N_r > N_0$. However, the reliability of the fault-tolerant chip is different from $e^{-\lambda_r t}$ since the chip may survive some failures. In other words, λ_r is the rate at which failures occur but these failures do not necessarily lead to complete chip failure.

Let R_{A_i} be the probability that a unit of area A_i is working, and R_{S_i} the probability that the reconfiguration logic of area S_i is working. With the result of Section 5.4.3, we can express R_{A_i} as $e^{-\lambda_{A_i} t}$ where λ_{A_i} is a function of the number gates in area A_i and t is the time. Similarly, $R_{S_i} = e^{-\lambda_{S_i} t}$.

The probability that redundant unit i is working is the probability of having at least 1-out-of-m_i units working multiplied by the probability that the reconfiguration logic is working, that is

$$R_{S_i}\sum_{j=1}^{m_i}\binom{m_i}{j}R_{A_i}^{j}(1-R_{A_i})^{m_i-j}=R_{S_i}(1-(1-R_{A_i})^{m_i}) \tag{5.9}$$

This would be the case if the units were continuously checked and upon the detection of a failure a reconfiguration process instantly replaces the failed unit by a spare one. Also, expression (5.9) assumes that all m_i units are operational at time $t = 0$. Processing defects may cause some of the units to be faulty at $t = 0$. Furthermore, the redundancy strategy introduced in Section 5.3 relies on periodic off-line testing. In the following, we derive a reliability expression, which incorporates the effect of processing defects, for the case of dynamic checking. Subsequently, this expression is extended to the case of periodic testing.

If there were k defective units after processing, then (5.9) becomes

$$R_{S_i}(1-(1-R_{A_i})^{m_i-k})\times \Pr\{k \text{ units were defective}\},$$

and since there can be up to $m_i - 1$ defective units, the probability that the redundant module is working is

$$R_i=\sum_{k=0}^{m_i-1}R_{S_i}(1-(1-R_{A_i})^{m_i-k})\times \Pr\{k \text{ units were defective}\}$$

Let $P_{A_i} = \Pr\{\text{unit of area} A_i \text{ is defect free}\}$,
$P_{S_i} = \Pr\{\text{reconfiguration logic of area} S_i \text{ is defect free}\}$, then $Q_{A_i} = 1 - P_{A_i}$ is the probability of having at least one defect in area A_i. The probability of having k out m_i units that are defective is

$$\binom{m_i}{k}Q_{A_i}^{k}(1-Q_{A_i})^{m_i-k}$$

Furthermore, the reconfiguration logic must be defect free at $t = 0$. Hence

$$\Pr\{k \text{ units were defective}\}=P_{S_i}\binom{m_i}{k}Q_{A_i}^{k}(1-Q_{A_i})^{m_i-k}$$

and the expression for R_i becomes

$$R_i=\sum_{k=0}^{m_i-1}R_{S_i}(1-(1-R_{A_i})^{m_i-k})P_{S_i}\binom{m_i}{k}(1-P_{A_i})^{k}P_{A_i}^{m_i-k}$$

The probability that the chip is working is the probability that all redundant modules are working, i.e.,

$$R_r=\prod_{i}^{n}R_i$$

P_{A_i} and P_{S_i} are functions of areas A_i and S_i and the defect density D. Since D is a random variable, it is necessary to use averaging to get expected values of the quantities considered. As for yield, we can distinguish three different ways of computing the average of R_r These are presented in Appendix B.

Reliability is defined as the conditional probability that a system is working at time $t > 0$ given that the system was working at $t = 0$.

$$R(t) = \Pr\{\text{system working at } t > 0 |\text{ system working at } t = 0\}$$

$$= \frac{\Pr\{\text{system working at } t > 0, \text{ system working at } t = 0\}}{\Pr\{\text{system working at } t = 0\}}$$

Given that if a chip is working at $t > 0$ then it must have been working at $t = 0$, we have

$$R(t) = \frac{\Pr\{\text{system working at } t > 0\}}{\Pr\{\text{system working at } t = 0\}}$$

The probability that a chip is working at $t = 0$ is the yield and the probability that the chip is working at $t > 0$ is R_r. Hence, the reliability of a dynamically checked chip is

$$R(t) = \frac{R_r}{Y_r} \tag{5.10}$$

In periodic testing, the operation of the chip is interrupted at regular intervals for the application of a test, the results of which are used to select working units. Between tests, the chip is unchecked: if it fails, it produces an incorrect output until the next test/repair cycle. Let T be the time between test/repair cycles.

Between tests, i.e., between times kT and $(k+1)T$, where k is an integer, the chip is working if the units that were selected at the last test are still working and, since part of the reconfiguration logic is used to steer the input/output signals from a unit to the input/output ports of a the redundant module, the reconfiguration logic must also be working. Therefore,

$$\Pr\{\text{Chip working at } t \in]kT, (k+1)T[\} = \prod_{i=1}^{n} e^{-\lambda_{A_i}(t-kT)} e^{-\lambda_{S_i}(t-kT)} \tag{5.11}$$

The probability in (5.11) must be multiplied by the probability that at the last test, i.e. at $t = kT$, there were at least 1-out-of-m_i unit working in every redundant module i and that all reconfiguration logic circuitry was working. This is exactly the reliability of the chip in the case of dynamic checking. Hence, the reliability of the chip in the case of periodic testing is

$$R_p(t) = R(kT)\prod_{i=1}^{n} e^{-(\lambda_{A_i}+\lambda_{S_i})(t-kT)}$$

where $k = \lfloor t/T \rfloor$: the largest integer such that $k \leq t/T$ and $R(kT)$ is given by (5.10). Figure 5.7 gives a graphical interpretation of the above result.

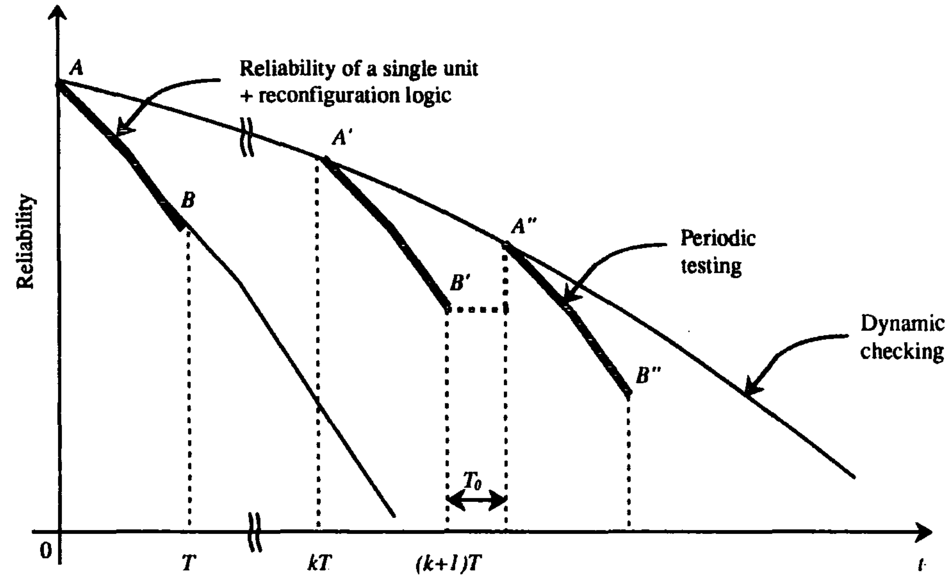


Figure 5.7 Graphical representation of the reliability of a chip in the case of periodic testing.

T_0 is the duration of the test/repair process and will be neglected. The chords $AB, A'B'$ and $A''B''$ are parallel to each other. It can be seen from Fig. 5.7 that the reliability in the case of dynamic checking constitutes an upper bound of the reliability of the chip in the case of periodic testing. Similarly, the curve that connects points $BB'B''\ldots$ is a lower bound for $R_p(t)$. The equation for this curve is given by

$$R_{pL}(t) = R_r(t)\Big(\prod_{i=1}^{n} e^{-(\lambda_{A_i}+\lambda_{S_i})T}\Big)$$

The curve that lies halfway between the upper and lower bounds represents the *median* reliability. Its equation is given by

$$R_{pM}(t) = R_r(t)\Big(\prod_{i=1}^{n} e^{-(\lambda_{A_i}+\lambda_{S_i})T/2}\Big)$$

R_{pM} is a good and practical approximation of R_p.

5.5 A CASE STUDY

In this section, we consider the application of the models developed in Section 5.4. The area of the non-redundant chip is assumed to be A_0 = 3cm^2 so that the yield Y_0 = 10% for an average defect density D_0 = 1.0cm^{-2}. These values of A_0 and D_0 correspond to the profitable region of Fig. 5.12.1 (p. 154). For the purpose of reliability calculations, it is also assumed that 200,000 gates can be put on such a chip. The chip is partitioned into n units of equal areas $A = A_0/n$, the units are replicated by the same factor m, and the area of the reconfiguration logic is assumed to be the same for all n redundant units. The yield and reliability expressions is this case are presented in Section B.2 of Appendix B.

The models can be used when the units have different areas, replication factors, and requirements for the reconfiguration logic. However, the number of input parameters to the models might then become too large to allow any analysis of their effects.

The yield and reliability figures presented in the following subsections are dependent on some parameters that have been given fixed, and somewhat arbitrary values (e.g., number of pins, junction temperature, in the case of failure rate estimation). The focus should therefore be on comparing these figures to the non-redundant case, rather than on their absolute values.

5.5.1 The Effect of the Size of the Reconfiguration Logic

The effect of the reconfiguration logic is studied first because it determines whether there is any improvement in yield and reliability. Furthermore, the assumptions concerning the size of the reconfiguration logic have a great impact on how these improvements vary with the other parameters of the model.

The size of the reconfiguration logic for a redundant unit is $S = m(\alpha + \beta)A_0/n = mcA_0/n$ (refer to Section 5.3.2), where α is the ratio of the size of the local reconfiguration logic to the size of one replicated unit, and β is the ratio of the size of the global reconfiguration logic to the size of the m replicated units. Let S_T be the total area of the reconfiguration logic, i.e., $S_T = nS = mcA_0$.

If the ratios α, β, and hence c, are assumed to remain constant when n varies, then S decreases as n increases, but S_T remains constant. This is referred to as the *optimistic* case. On the other hand, if c is assumed to vary linearly with n, then S remains constant as n varies, but S_T increases as n increases. This is referred to as the *pessimistic* case.

$$\begin{array}{llll}
\text{if } c = \text{const} & \Rightarrow & S \searrow \text{ as } n \nearrow & \text{but } S_T = \text{const} \\
 & & S \nearrow \text{ as } n \searrow & \\
\text{if } c = c'n, c' = \text{const} & \Rightarrow & S = \text{const} & \text{but } S_T \nearrow \text{ as } n \nearrow \\
 & & & S_T \searrow \text{ as } n \searrow
\end{array}$$

or in terms of the area of a unit A,

$$\begin{array}{llll}
\text{if } c = \text{const} & \Rightarrow & S \searrow \text{ as } A \searrow & \text{but } S_T = \text{const} \\
 & & S \nearrow \text{ as } A \nearrow & \\
\text{if } c = c'n, c' = \text{const} & \Rightarrow & S = \text{const} & \text{but } S_T \nearrow \text{ as } A \searrow \\
 & & & S_T \searrow \text{ as } A \nearrow
\end{array}$$

Taking c as a constant therefore has consequences that are hard, or impossible, to justify. A summary of the analysis of the yield results for this case is:-

- The optimum size of the replaceable unit is zero, i.e., the smaller the unit the better the yield and reliability.
- The yield figures are always higher than in the pessimistic case, $Y_r = 75\%$ when $n = 10, m = 4$ and $c = 0.03$, for example.
- The optimum values of the replication factor m tend to be large, $4 \leq m \leq 8$.
- There is no yield degradation when the size of reconfiguration logic is such that $0.006 \leq c \leq 0.15$.

In particular, the first result is the same as when the size of the reconfiguration logic is zero [216]. The reliability results follow the same pattern. Note that this case may occur for some chip architectures, as shown in [229].

Alternatively, taking $c = c'n$ also has consequences that are hard to justify. For example, the consequence that $S_T \searrow$ as the size of a unit increases is hard to accept. In fact, it was shown in Chapter 4 that the size of the BIST logic alone is minimum for a particular size of the unit. However, the yield and reliability figures obtained in the pessimistic cases are more consistent with intuitive expectations. Therefore, in the absence of a precise method of estimating the size of the reconfiguration logic, the results that have been selected for presentation are those corresponding to the pessimistic case.

Figure 5.8 illustrates the range of values of parameter c' for which there is any yield improvement. When the chip consist of five units, it can be seen that if $c' > 0.07$, then there is no yield improvement for any value of m. For a given value of m and n the value of c' above which there is no yield improvement determines the critical size of the reconfiguration logic. Table 5.2 gives these values for different m and n. The main observation from Table 5.2 is that the critical sizes are very low: If the reconfiguration logic occupies more than about 9% of the total chip area, the yield of the fault-tolerant chip is always lower than the non-redundant yield, for all values of m and n. This is a very stringent requirement, given that, as seen in Chapter 4, the implementation of BIST alone requires between 30% and 50% of the total chip area. Similar results can be obtained for reliability but there would be different tables for different mission times.

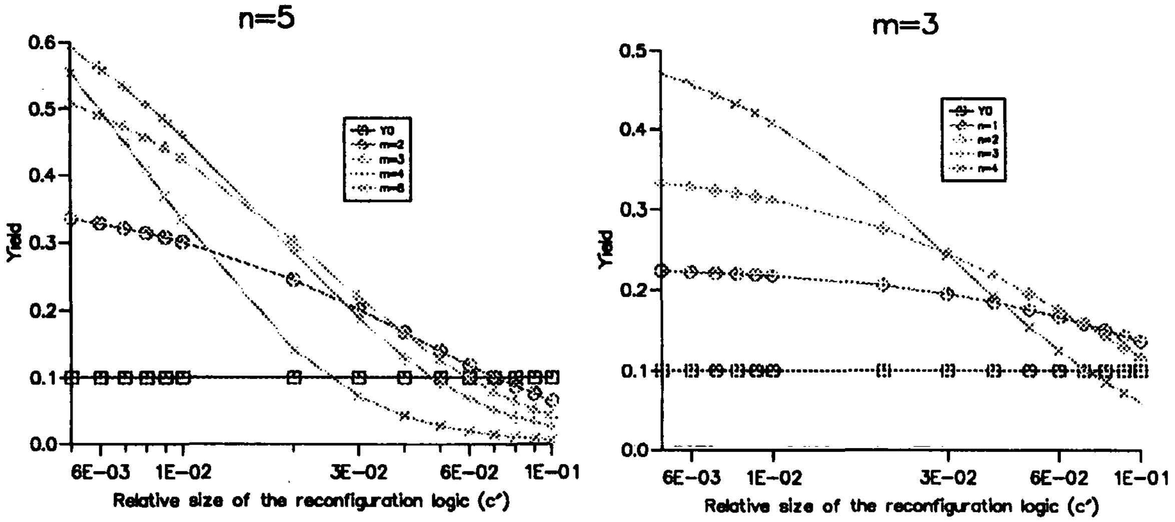


Figure 5.8 Effect of the size of the reconfiguration logic on yield.

Rounding errors make the result meaningless for the entries marked with (*). Also, no two distinct entries should be exactly equal, but the calculations were done at discrete points.

5.5.2 Optimum Replication Factor and Unit Size

The natural tendency is to keep the replication factor as small as possible, while achieving acceptable yield and reliability improvements. Hence, given that Y_0 = 10%, if Y_r = 48.2% for m = 6 and Y_r = 45.9% for m = 4, it is preferable to choose m = 4.

Figure 5.9 shows the variations of yield with the replication factor and the number of units. It clearly establishes that for a given number of sections n, there exists an optimum value of the replication factor m that maximizes the yield. Similarly, for a given value of m, there exists a best way to partition the chip into n units so that the yield is maximum.

Table 5.2 Critical values of the size of the reconfiguration logic.

Relative size c'							
m n	2	3	4	5	6	7	8
1	0.1	0.1	0.1	0.1	0.1	0.1	0.09
2	0.1	0.1	0.1	0.08	0.07	0.06	0.05
3	0.09	0.08	0.07	0.06	0.05	0.04	0.04
4	0.08	0.07	0.05	0.04	0.04	0.03	0.03
5	0.07	0.05	0.04	0.03	0.03	0.02	0.02
6	0.06	0.05	0.04	0.03	0.02	0.02	(*)
7	0.05	0.04	0.03	0.02	0.02	0.02	(*)
8	0.04	0.03	0.03	0.02	(*)	(*)	(*)
9	0.04	0.03	0.02	0.02	(*)	(*)	(*)
10	0.04	0.03	0.02	(*)	(*)	(*)	(*)

% of total chip area							
m n	2	3	4	5	6	7	8
1	9.09	9.09	9.09	9.09	9.09	9.09	8.26
2	9.09	9.09	9.09	7.41	6.54	5.56	4.76
3	8.26	7.41	6.54	5.66	4.76	3.85	3.85
4	7.41	6.54	4.76	3.85	3.85	2.91	2.91
5	6.54	4.76	3.85	2.91	2.91	1.96	1.96
6	5.66	4.76	3.85	2.91	1.96	1.96	(*)
7	4.76	3.85	2.91	1.96	1.92	1.96	(*)
8	3.85	2.91	2.91	1.96	(*)	(*)	(*)
9	3.85	3.85	1.96	1.96	(*)	(*)	(*)
10	3.85	2.91	1.96	(*)	(*)	(*)	(*)

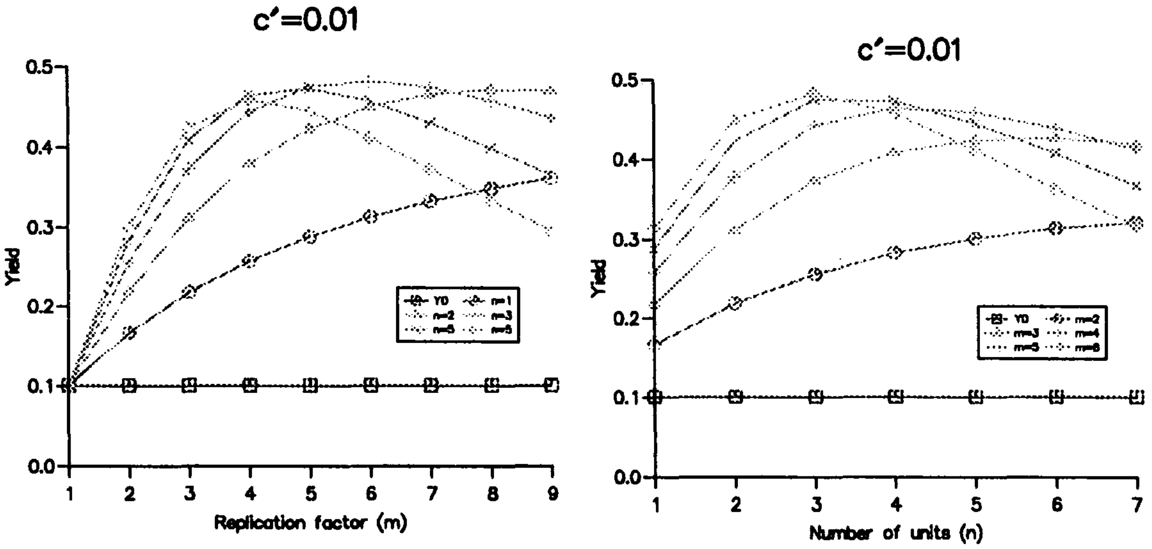


Figure 5.9 Effects of the replication factor and partitioning on yield.

If there are no constraints in choosing the values of m and n, then results as presented in Table 5.3 could be used for the selection of an optimum partitioning of the chip and an optimum replication factor.

The results for reliability follow the same pattern, as shown in Fig. 5.10. The expression used was the median reliability corresponding to Case 1 in Appendix B, in order to avoid rounding errors for values of m and n such that $m \times n \approx 45$. Note

Table 5.3 Maximum yield for different m and n.

n \ m	1	2	3	4	5	6	7	8
1	10.03	16.69	21.78	25.72	28.83	31.31	33.29	34.88
2	10.03	21.83	31.22	37.87	42.31	45.08	46.58	47.17
3	10.03	25.58	37.41	44.32	47.53	**48.28**	47.49	45.82
4	10.03	28.27	40.93	46.46	47.36	45.77	43.00	39.80
5	10.03	30.96	42.49	45.94	44.58	41.22	37.29	33.43
6	10.03	31.41	42.70	43.96	40.79	36.35	31.92	(*)
7	10.03	32.18	42.02	41.27	36.83	31.82	(*)	(*)
8	10.03	32.56	40.75	38.33	33.05	(*)	(*)	(*)
9	10.03	32.64	39.14	35.37	28.82	(*)	(*)	(*)
10	10.03	32.48	37.34	32.52	(*)	(*)	(*)	(*)

that the values of m and n that give the best improvement in reliability do not coincide with the values of m and n that give the best improvement in yield, suggesting some trade-offs between the two measures.

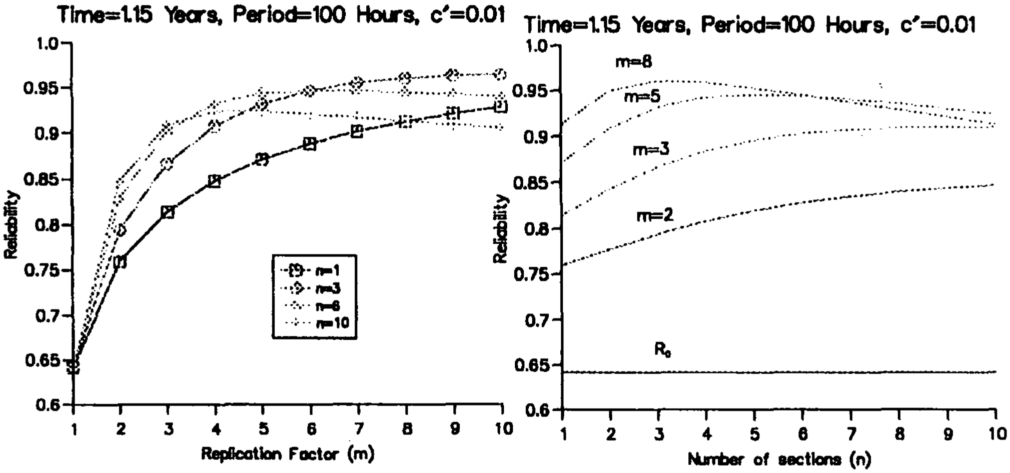


Figure 5.10 Reliability as a function of the replication factor and the number of units.

Another observation from Fig. 5.10 is that the curves do not have sharp maximums. This is due to the weak dependence of reliability on the number of gates in a chip.

5.5.3 Figures of Merit

It was shown in the previous sections that the yield can be improved if the size of the reconfiguration logic is small enough. However, the increased area of a redundant chip reduces the number of chips that can be placed on a wafer, assuming a constant wafer area. But since the yield is increased, it might still be possible to get a higher number of chips that are working from a wafer. In this section, the conditions for which a higher number of chips can be obtained from a wafer through the use of redundancy are considered.

Let A_W be the usable area of a wafer. We assume that the number of possible chip locations on a wafer is the ratio of the usable wafer area to the area of a chip. In the non-redundant case, the number of chips per wafer is is given by

$$N_0 = \frac{A_W}{A_0} Y_0$$

In the redundant case, this number is

$$N_r = \frac{A_W}{A_r} Y_r$$

$N_r > N_0$ if

$$FM_Y = N_r / N_0 = \frac{Y_r}{A_r} \frac{A_0}{Y_0} > 1$$

The ratio FM_Y is called *Figure of Merit* in [208, 214]. Note that a redundancy strategy that improves the yield by a factor $X = Y_r/Y_0$ would not result in $FM_Y > 1$, unless the area increase is less than X. Also, in cases where Y_0 is extremely small, $Y_0 = 0.01\%$ say, large values of FM_Y can be obtained even if Y_r is unacceptably small. If $Y_r = 1\%$, then even if the area is increased by a factor of ten, we still get $FM_Y = 10$. Hence, the value of FM_Y alone is not an accurate indicator of the benefits gained from a particular redundancy strategy.

Figure 5.11 shows the variations of the figure of merit with the relative size of the reconfiguration logic, calculated from the model of Section 5.4. The critical values of c' are lower than in the case of yield.

Figure 5.12 shows the variations of the figure of merit with the replication factor and the number of sections, i.e., the size of the units.

5.5.4 Effects of Periodic Testing on Reliability

Figure 5.13 illustrates the effect of periodic testing on reliability calculated from the model of Section 5.4.

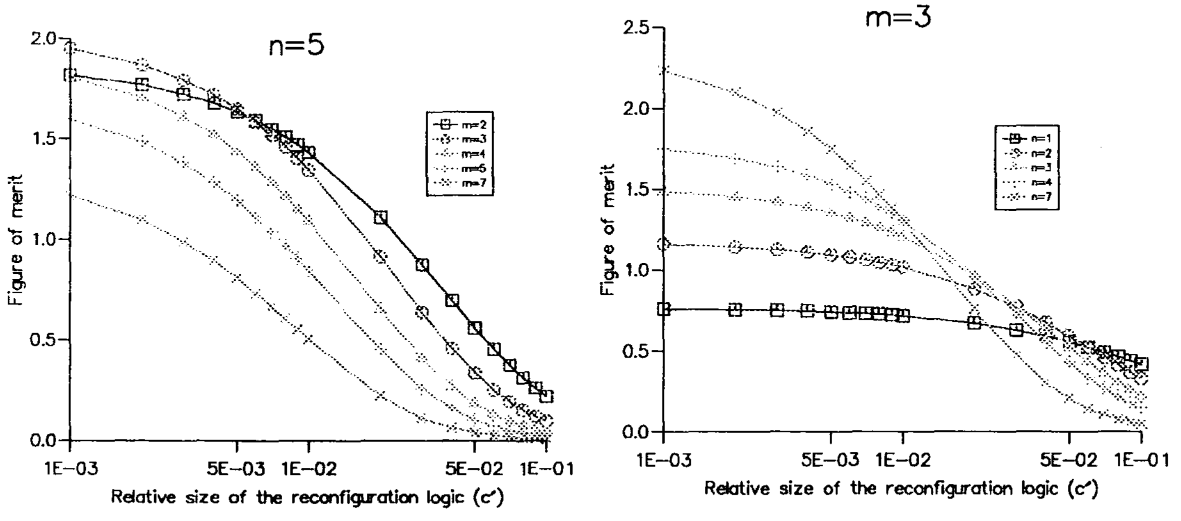


Figure 5.11 Relative size of the reconfiguration logic for which $FM_Y > 1$.

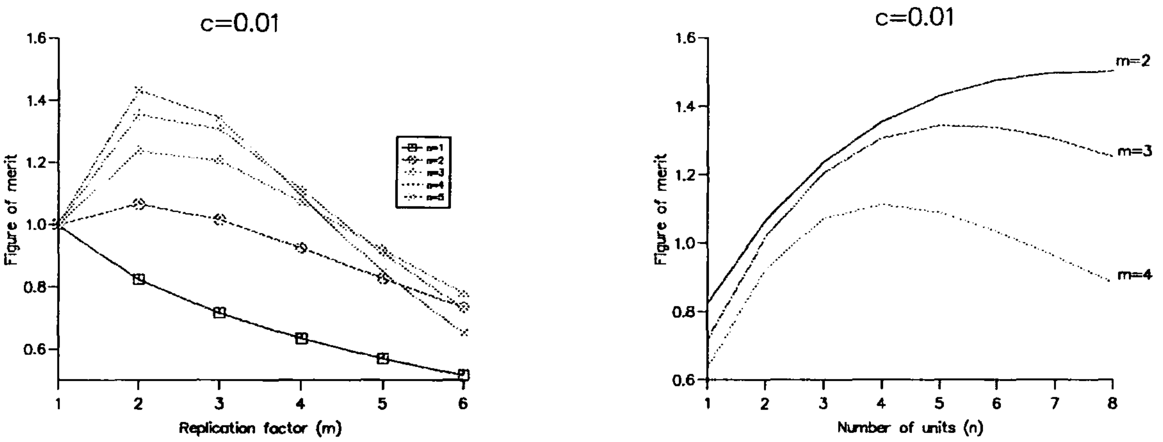


Figure 5.12 Optimum replication factor and unit size for FM_Y.

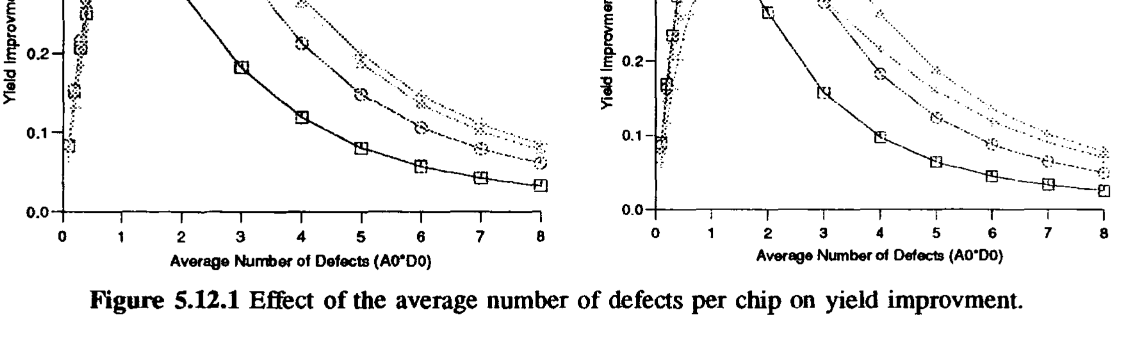


Figure 5.12.1 Effect of the average number of defects per chip on yield improvment.

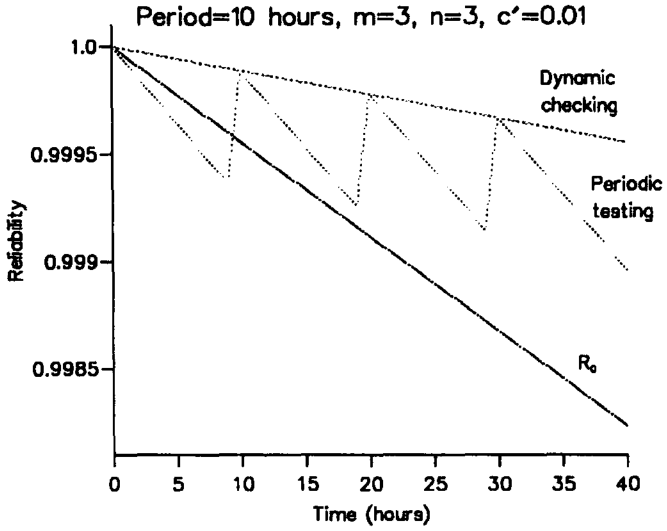


Figure 5.13 Effect of periodic testing on reliability.

Figure 5.14 shows that the reliability in the case of periodic testing remains very close to the reliability in the case of dynamic checking over a large range of values of the time interval between tests. Testing the chip every hour or every 100 hours is virtually the same as continuous checking. Only when the T is greater than few thousand hours does the reliability in the case of periodic testing become significantly lower than in the case of dynamic checking.

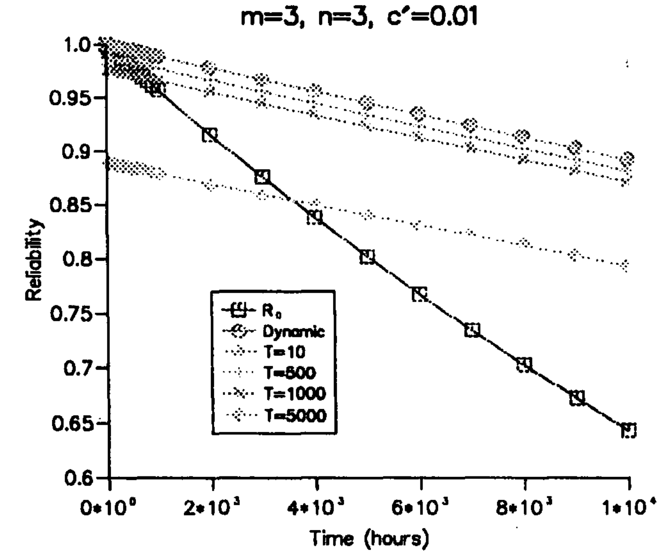


Figure 5.14 Comparison of dynamic checking and periodic testing.

However, before time $t = T$, the reliability of the fault-tolerant chip is lower than the reliability of the non-redundant chip since the reconfiguration logic must be working, in addition to one unit in every redundant module. Hence, it is preferable to have T of the order of few hours. Alternatively, it is possible to have shorter intervals between tests just after system startup and then increase the time interval between tests later.

5.6 CHAPTER SUMMARY

The models developed in this Chapter are simpler, more practical and more trackable than Koren's approach [225], and yet, they take into account important parameters such as the size of a unit, the replication factor and the time between tests. The models could also be adapted to other redundancy strategies, as shown in the Poster section of [229]. The main results from this Chapter are as follows:-

The simple redundancy strategy used to develop the yield and reliability models requires a large amount of redundancy: the optimum values of the replication factor m are in the range 2 to 5. Nevertheless, it improves yield and reliability substantially. Even the figure of merit becomes greater than one if the size of the reconfiguration logic is small enough. Therefore, it is not the amount of redundancy that is important, but rather the way in which the redundancy is used (distributed *vs* centralised, partitioning).

The drastic effect of the reconfiguration logic was illustrated is Section 5.5.1. If it occupies more than about 10% of the chip area, then no yield improvement can be achieved. This is probably the reason behind the use of hard reconfiguration in commercial DRAMs. If the size of the reconfiguration logic is small enough then it is shown in Section 5.5.2 that there exist optimum values of the replication factor and the number of units to maximise yield and reliability.

A chip that is periodically tested may fail to give the correct output between tests. However, it is shown in Section 5.5.4 that the probability of this happening is nearly the same as for the complete failure of a dynamically checked chip. Furthermore, it was surprising to find that the time between tests can be as long as a thousand hours without having a significant effect on the reliability.

Chapter 6
Summary and Conclusions

The subject of fault-tolerance is very extensive even when restricted to integrated circuits. The review and discussion of Chapter 2 centred around the following sub-subjects.

- Sources of faults.

- Effects of faults on circuits.

- Fault detection.

- Fault correction.

- Yield and reliability modelling.

It was made clear that each of these sub-subjects is itself very wide and that none of them can be addressed individually. For example, fault-detection requires a thorough knowledge of the effects of faults on circuits which in turn requires an identification of the possible sources of the faults.

At the structural level, faults can be described in terms of breaks, short circuits, variations in the resistances of the different interconnects, and so on. The consideration of faults at this level is useful only for improving the manufacturing process or for determining the best design rules of the process. For the purpose of fault detection and/or correction, it is necessary to consider faults at higher levels of abstraction. The higher the level, the easier it is to deal with faults. However, since higher levels of abstraction are obtained by hiding the structure of lower levels, they also hide the physical faults.

Switch level modelling seems to be the most appropriate for CMOS technology, since it can allow for transistor stuck-open and stuck-on faults. The dismissal of these types of fault by Shen et. al. [53, 52] is challenged in this thesis on three grounds:

- Their arguments are invalid.
- Their approach is inappropriate. If there is a need to assess the frequency of occurrence of stuck-open faults, then the approach of Woodhall et. al. [163] is much more appropriate.
- Even if their approach was correct, the dismissal of faults because of their low probability of occurrence is not justifiable.

Fault-detection mechanisms can be classified as either on-line or off-line. The concept of on-line error detection (using codes) appeared at the same time as the first large scale application of digital systems. A considerable research effort has been devoted to find effective methods for on-line error-detection. However, there is still no method that achieves high coverage of physical faults with acceptable hardware overheads. Hence, this work considered the use of off-line fault detection.

A distinction must also be made between fault-correction mechanisms that rely on physical restructuring and those employing soft-reconfiguration, as in this work. Defect-tolerance techniques have very limited capabilities for dealing with faults in field use. Failure-tolerance techniques, on the other hand, can deal with both manufacturing defects and field failures, and therefore they are more appropriate.

6.1 THE MAIN ACHIEVEMENTS OF THE RESEARCH

The aim of the work presented in this thesis was the investigation of techniques for the design of fault-tolerant ICs for improved yield and reliability (p.2). A large part of this work was devoted to the problem of fault-detection while the other part consisted of the development of a mathematical model for assessing the effects of fault-tolerance on yield and reliability. Off-line testing was selected because it achieves a better coverage of physical faults and it requires less hardware than on-line error-detection. The physical repair of faults is impractical or impossible, therefore, the fault-correction mechanism consists of a reconfiguration process whereby the part of the IC that contains a fault is disabled and replaced with a spare part initially included in the design. Decisions must be made at the design stage on what is the size of the part of the IC that must be discarded when faulty. The yield and reliability model can be used in making such decisions.

6.1.1 Fault Detection

The first objective of the research (p.2), a method for detecting all possible faults, was successfully achieved. The approach adopted in Chapter 3 to address the problem proved to be effective. It consisted of a detailed analysis of fault effects, not for the sake of deriving tests, but rather for the determination of some high level requirements for the detection of faults. This approach allowed the derivation of simple test generation procedures. The test sequences generated by these procedures detect all stuck-open faults and, hence, all other detectable faults, since these have been shown in Chapter 3 to be detectable by tests for stuck-open faults.

The test generation procedures of Chapter 3 have two novel characteristics:

- The fault-free response of the circuit under test is trivial.
- Faults are not considered individually.

In particular, the use of the first property is proposed in this thesis as an effective solution to the problem of test response analysis. The second property is also very useful since it eliminates the need for enumerating the faults. The number of possible faults in a VLSI circuit can be extremely large. Therefore, the only faults that may be enumerated are the ones that are not detected. A further advantage of the proposed test generation procedures is that they require a relatively high level description of the circuit under test while detecting faults that belong to the switch level.

The problem of test invalidation by circuit delays is also considered in Chapter 3. A procedure that generates robust tests, when they exist, was derived here. When robust tests do not exist, a circuit transformation technique is proposed. This transformation reduces the number of devices of the original circuit, whereas current proposed techniques increase the device count and introduce extra test inputs.

6.1.2 Built-In Self-Test

The second objective of the research (p.2), the implementation of BIST using the methods of Chapter 3, together with the consideration of the problem of testing the BIST circuits and the determination of the effect of partitioning on the size of the BIST hardware, were addressed in Chapter 4.

The implementation of BIST for the detection of all faults in the functional circuits represents a significant departure from current BIST approaches where even the detection of all stuck-at faults is not guaranteed, either because of the test patterns that are used and/or because of the compression of the test response into a signature. The test sequences used in Chapter 4 are derived using the procedures of Chapter 3, hence, they detect all detectable faults. Furthermore, because they produce a trivial output, test response compression is eliminated with no loss in the fault coverage. This allows the implementation of all-fault BIST with a hardware overhead comparable to that usually associated with current BIST techniques.

The testing of the extra hardware required for BIST itself was also considered in Chapter 4. The detection of faults in the test circuitry requires the addition of further hardware that also needs testing through the addition of further hardware that also must be tested, and so on. For the parts of the test circuitry used for response analysis, it is shown that this infinite loop can be stopped. For the multiplexers, used to select between data and test inputs, a simple re-design is suggested to make these circuits testable while they are performing their function. However, no acceptable method was found to test for faults in the ROM array of the test sequence generator. This led to the investigation of LFSR-based test sequence generators, discussed below.

Examples are given in Chapter 4 to illustrate the difficulties in achieving complete test coverage, i.e., the detection of all faults in both the functional and the test hardware. The examples considered lead us to conclude that improved test coverage may be achieved if different chips were to cooperate in testing each other's untested parts.

From the observation that the test response circuitry and the input multiplexers are distributed, as opposed to the centralised nature of the test sequence generator, we also conclude that the test circuits should be as distributed as possible and that the best way towards complete test coverage would be to design these circuits so that they are tested while performing their functions.

Another issue addressed in Chapter 4 concerns the determination of the optimum size of the unit for BIST. It is shown that the hardware overhead achieves a minimum for a particular partitioning scheme. Alternative implementations of the test sequence generator have been investigated and the use of a central LFSR in conjunction with distributed filters is proposed as a promising method towards distributed BIST and hence complete test coverage. The time redundancy technique studied in Chapter 4 was shown to have some potential for on-line fault-detection.

6.1.3 Evaluation of Yield and Reliability Improvements

The fourth objective of the research (p.3) was to consider how to evaluate the likely improvement in yield and reliability that might be achieved using the fault-tolerance technique described in Chapter 5. The main features of the proposed yield and reliability models are:

- They are simple, compared with the only published models of Koren et. al. [225].
- They incorporate the effects of important parameters: size of the reconfiguration logic, amount of redundancy, unit size, defect density, dependence of reliability on yield, and periodic testing.
- They can be extended to other fault-tolerance strategies.

The consideration of the stage at which the averaging process is carried out, section 5.4.2, is believed to be the first treatment of this question.

The illustration of the application of the models gave two interesting results. The first concerns the stringent requirements on the size of the reconfiguration logic. For the example considered, if the size of the reconfiguration logic exceeds about 10% of the total area of the chip, then no amount of redundancy will improve the yield.

The second interesting and surprising result concerns the effect of periodic testing. It was found that the time interval between tests can be as long as many hundred hours without significantly degrading the reliability of the chip as compared to a dynamically checked chip, provided that the time between tests is less than the critical down-time of the system using the chip.

6.2 GENERAL CONCLUSIONS OF THE RESEARCH

The contributions of the work, described in section 6.1, lead to the following general conclusions:

- The derivation of test patterns by considering the effect of faults on circuit operation individually is a time consuming task, as illustrated in Appendix A. The procedures presented in Chapter 3 generate complete test sequences without enumerating the faults. Therefore, the derivation of test sequences using some high level requirements for the detection of faults is much more effective.
- The highest levels of fault coverage can only be attained through the use of deterministically derived test sequences, as opposed to pseudo-random or pseudo-exhaustive test sequences. The benefit of a guaranteed high level fault coverage will be negated if the test response analysis uses data compression techniques. On the other hand, the procedures for test sequence derivation can be adapted so that the fault-free response of circuits follows a trivial pattern that can be generated by simple on-chip circuitry. Therefore, there is no need for compromising on the quality of testing through the use of pseudo-random or exhaustive sequences and/or signature analysis.

- The BIST circuitry should not be considered as a hardcore: the aim should be to test as much of the test circuits as possible. The best way to achieve this aim would be to design the circuits in such a way that they are tested while performing their function. However, this would make it impossible to distinguish between faults in the BIST hardware and faults in the functional hardware. Hence, a further conclusion is that the BIST circuits should be as distributed as possible.

- The implementation of BIST at the level of very small units would result in large area overheads, because each unit requires some test circuits. Similarly, implementing BIST at the level of very large units also has disadvantages: the test sequences would be difficult to derive and their lengths would be excessive. Therefore, different partitioning schemes should be investigated in order to minimise the size of the BIST circuitry.

- It was shown that there exists a critical size of the reconfiguration logic beyond which there is no improvement in yield or reliability. Therefore, the size of the reconfiguration logic has a significant role in determining whether there is any improvement in yield and reliability.

- For the redundancy strategy considered and under the 'pessimistic' assumptions on the variations of the size of the reconfiguration logic with the size of the units, there is an optimum replication factor and an optimum unit size that maximise yield and reliability. These optima are different for yield and reliability, suggesting some trade-offs between the two measures.

- The way in which redundancy is used has more influence on yield and reliability improvements than the amount of redundancy.

- Periodic testing can achieve reliability levels comparable to those achieved in dynamically checked systems.

Currently, system testing and maintenance activities are more of an art than an exact science. This is due to imperfect testing at lower levels. BIST at the chip level, coupled with very high levels of fault coverage will not only make these activities more systematic, but will also result in savings, at least because of the fact that the field-replaceable units become chips rather than whole PCBs. Product quality will also be improved since there will be a reduction in the number of faulty items that pass the final manufacturing tests.

6.3 SUGGESTIONS FOR FURTHER RESEARCH

6.3.1 Reconfiguration

As mentioned in the previous sections, the requirements on the size of the reconfiguration logic are very stringent. Furthermore, the reconfiguration logic is ignored in many proposed fault-tolerance strategies. As far as the number of publications is concerned, this sub-subject of fault-tolerance is the one that received the least attention.

Designing the reconfiguration logic so that it is fully distributed is a challenging task. The principal requirement in this case, in addition to the requirement on size, is that the faults in the reconfiguration logic of one unit should not affect another unit. This requires a kind of fail-safe design; this needs to be investigated further.

6.3.2 Testing

Combining the outputs of a circuit into a single signal for test response analysis, using a circuit that is tested while performing its function would greatly simplify the implementation of BIST. The method used in Chapter 4 was possible only because there were just two outputs to combine, and an exhaustive search was feasible. A more systematic method for determining the appropriate circuit would be very useful.

On the test generation side, the use of a central LFSR with distributed filters is also promising. The minimum length of the LFSR is determined by the longest test sequence required by the CUTs. The inputs to each filter are a subset of the outputs of the LFSR. It would be interesting to investigate procedures for selecting the inputs to each filter so that it is testable while performing its function and its size is minimised. Increasing the length of the LFSR increases the choices of the inputs to the filters. In addition, there are many different complete test sequences for a given CUT. Hence, the search space is very large.

Fault simulation, used to asses the effectiveness of test sequences is a computationally intensive task. Given that the requirements for the detection of any fault can be established, it is believed that an analysis of the results of a fault-free simulation can be used as an alternative to fault simulation. The analysis consists simply of verifying that the requirements are met. This may be less expensive than fault simulation since the enumeration of very large numbers of faults can be avoided.

The time redundancy technique also needs further investigation, especially as a possible low-cost on-line fault detection approach.

With further improvements on these lines, there are prospects for the production of VLSI circuits that are at least able to report on their faulty/operational status.

Appendix A
Fault Analysis

A.1 BREAKS IN DIFFUSION LAYER

Figure 3.3 shows the possible breaks in the diffusion layer. For clarity, only the breaks in the pull-up network are shown. With the adopted layout style, it can be seen that all open circuits in the diffusion layer result in a source or drain of a transistor being disconnected from the rest of the circuit, making the transistor stuck-open.

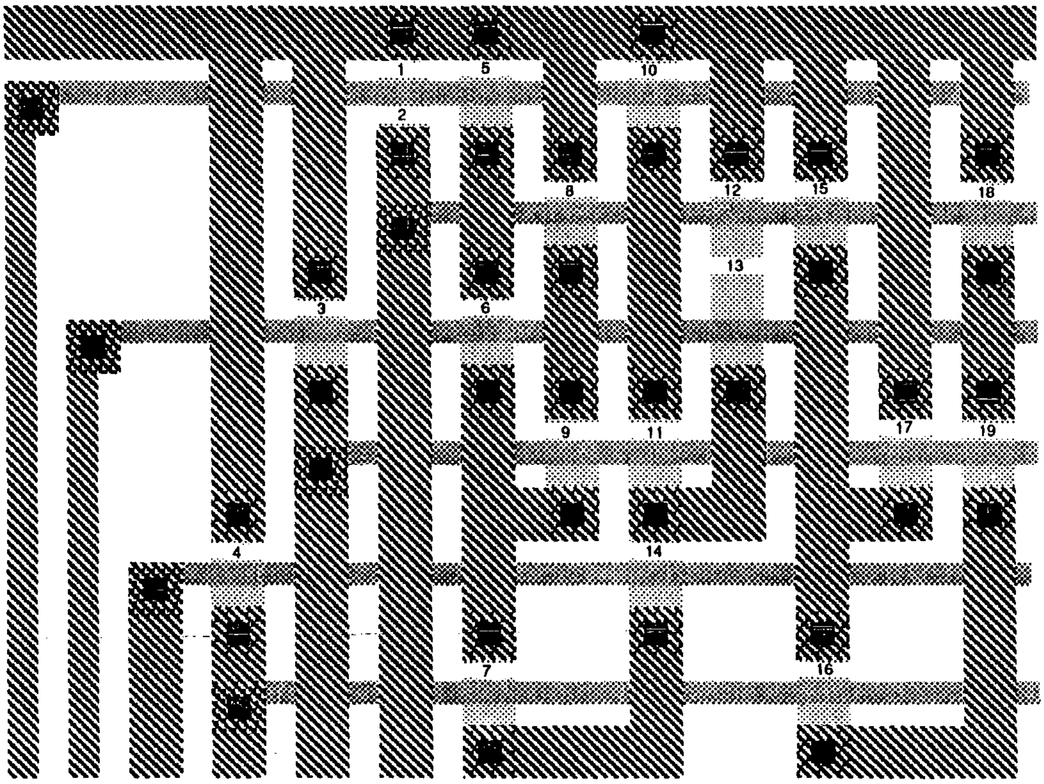


Figure A.1 Location of breaks in diffusion layer in the pull-up network.

Break bdp1: This break disconnects the source of transistor P1 from VDD. Hence, the output of the inverter cannot be driven to logic one. It is effectively stuck-at zero. The only possible test vectors are those that set A to zero, and an analysis of the response of the circuit for these vectors gives the following:

A	B	C	SUM		$CARRY$	
0	0	0	1	*	0	
0	0	1	1		1	*
0	1	0	1		1	*
0	1	1	1	*	1	

From the above table, we note that the input vectors $ABC = 000$ and $ABC = 011$ produce incorrect values at the sum output and that the input vectors ABC = 001 and $ABC = 010$ produce incorrect values at the carry output. Therefore, there are four possible test vectors for the fault under consideration. These will be noted as $000(S), 011(S), 001(C), 010(C)$ where the symbol between brackets refers to the output where the fault is detected.

Break bdp2: This break has exactly the same effect as the previous one, and hence is detected by the same test vectors. In the following, breaks like *bdp1* and *bdp2* will be treated as a single one.

Break bdp3: Break *bpd3* disconnects the source of transistor P2 from VDD, making the output of the inverter $\overline{B}$ stuck-at zero. The response of the circuit to the four likely test vectors is:

A	B	C	SUM		$CARRY$	
0	0	0	1	*	0	
0	0	1	1		1	*
1	0	0	1		1	*
1	0	1	1	*	1	

The possible test vectors are $000(S), 101(S), 001(C), 100(C)$.

Break bdp4: This makes $\overline{C}$ stuck-at zero. The circuit response to the four likely test vectors is:

A	B	C	SUM		$CARRY$	
0	0	0	1	*	0	
0	1	0	1		1	*
1	0	0	1		1	*
1	1	0	1	*	1	

Any of $000(S), 110(S), 010(C), 100(C)$ can be selected as a test vector.

Break bdp5: Break *bpd5* disconnects the source of transistor P4 from VDD, making it impossible to charge the sum output through the path {P4 P5 P6}, but since there are other paths from VDD to the output, this does not result in a stuck-at zero fault as in the previous cases. The response of the circuit under such a fault is :

A	B	C	SUM	
0	0	0	0	
0	0	1	Z	*
0	1	0	1	
0	1	1	0	
1	0	0	1	
1	0	1	0	
1	1	0	0	
1	1	1	1	

Z denotes a high impedance state. This occurs when both pull-up and pull-down networks are off, isolating the output node from both VDD and GND. In this case, the output holds its previous logic state, in the form of a charge on the capacitance of the output node.

This fault prevents the output from being charged through the path comprising transistors P4, P5 and P6, as said before. Hence, to test for the presence of this fault we must attempt to charge the output through the path {P4 P5 P6}. However, if the output is already charged to logic 1, then when attempting to charge it again through the path {P4 P5 P6}, it will not be possible to say whether the fault is present or not, since if the fault is present the output will be high (retaining its previous value) which is the same as when the fault is not present. Therefore, another requirement to test such a fault is that before we attempt to charge the output, we must make sure that the output is discharged first, so that if the fault is present, the output will be low when attempting to charge it through path {P4 P5 P6}. In other words, this type of faults require a pair of vectors for their detection. The first vector being an initialisation one and the second is the test vector.

For the fault under consideration, the initialisation vector can be any input combination that sets the output to logic 0, and the test vector must activate the path {P4 P5 P6}. The only vector that activates this path is $ABC = 001$. The pair of vectors is denoted $ABC = (d, 001)(S)$ where the d stands for any input combination that discharges the output.

Break bdp6: Break *bdp6* disconnects the source of P5 from the drain of P4. It is detected by the same pair of vectors as *bpd5*.

Break bdp7: Break *bdp7* disconnects the source of transistor P6 from the drain of transistors P5 and P8 making it impossible to charge the output through neither path {P4 P5 P6} nor {P7 P8 P6}. We can test for this fault by attempting to charge the output through either paths. So we have the two possible pairs of vectors: $ABC = (d, 001)(S)$ or $ABC = (d, 111)(S)$.

Break bdp8: Break *bdp8* disconnects the source of transistor P7 from VDD, making it impossible to charge the output through path {P7 P8 P6}. A test for this fault is $ABC = (d, 111)(S)$.

Break bdp9: Break *bdp9* results in the source of transistor P8 being disconnected from the drain of transistor P7 resulting in a break in the path {P7 P8 P6}. Hence, this fault is detected by the same pair of vectors as for *bdp8*.

Break bdp10: Break *bdp10* disconnects the source of transistor P9 from VDD. To test for this fault, path {P9 P10 P13} must be activated, therefore, $ABC = (d, 010)(S)$.

Break bdp11: Disconnects the source of P10 from the drain of P9. It is detected by the same pair of vectors as for *bdp10*.

Break bdp12: Disconnects source of P11 from VDD. Path {P11 P12 P13} must be activated. $ABC = (d, 100)(S)$.

Break bdp13: Disconnects the source of P12 from the drain of P11. It is detected by the same pair of vectors as *bdp12*.

Break bdp14: Disconnects the source of P13 from the drains of P10 and P12. Either paths {P9 P10 P13} or {P11 P12 P13} can be activated to test this fault, so $ABC = (d, 010)(S)$ or $ABC = (d, 100)(S)$.

Break bdp15: Causes the source of transistor P14 to be disconnected from VDD. The initialisation vector can be any vector that discharges the carry output. The test vector must activate the path {P14 P15} only (making sure that P16 is off). Therefore, the pair of vectors required is $ABC = (d, 101)(C)$.

Break bdp16: Disconnects the source of transistor P15 from the drains of transistor P14 and P16. The possible test pairs are $ABC = (d, 101)(C)$, or $ABC = (d, 011)(C)$.

Break bdp17: Disconnects the source of P16 from VDD. $ABC = (d, 011)(C)$.

Break bdp18: Disconnects the source of P17 from VDD. $ABC = (d, 110)(C)$.

Break bdp19: Disconnects the source of P18 from the drain of P17. It is detected by the same test pair of vectors as *bdp18*.

Figure 3.4 shows the locations of breaks in diffusion layer of the pull-down networks.

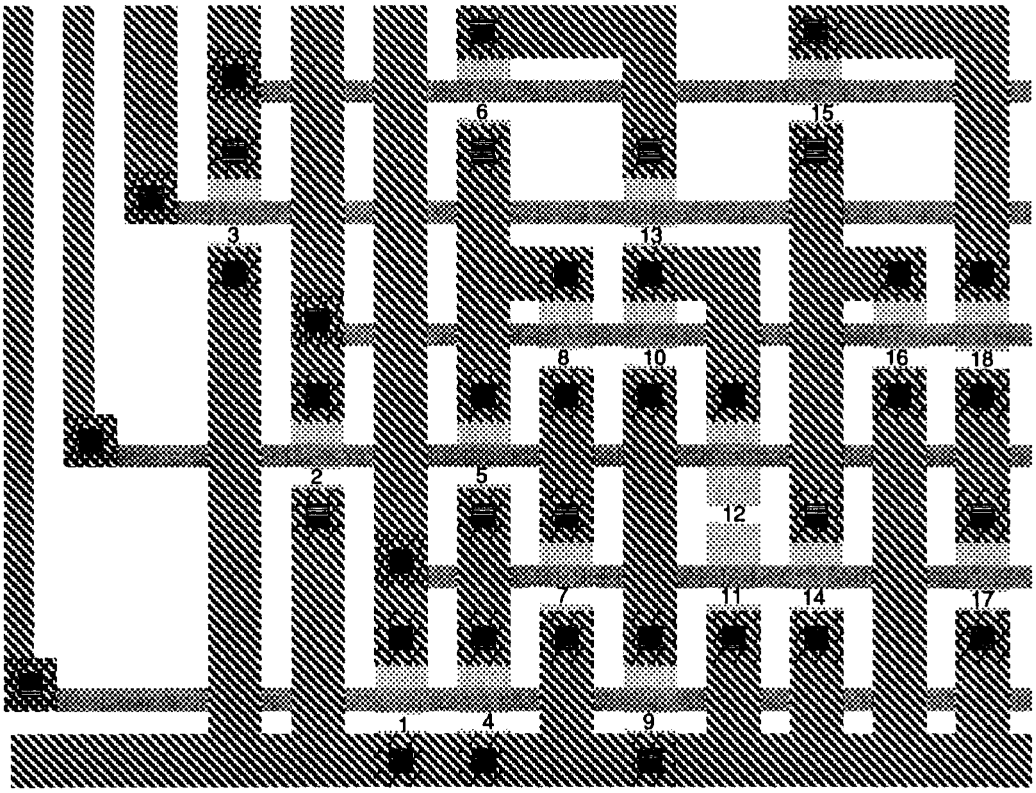


Figure A.2 Breaks in the diffusion layer of the pull-down network.

Break bdn1: Causes the source of transistor N1 to be disconnected from GND, making it impossible to discharge the output. The output of the inverter is therefore stuck-at 1. The four possible test vectors are those that set A to 1. The response of the circuit for these vectors is as follows:

A	B	C	SUM		$CARRY$	
1	0	0	0	*	0	
1	0	1	0		0	*
1	1	0	0		0	*
1	1	1	0	*	1	

The possible test vectors are: ABC = 100(S), ABC = 111(S), ABC = 101(C) and ABC = 110(C).

Break bdn2: Causes the source of the transistor N2 to be disconnected from GND resulting in the output of the inverter being stuck-at 1. The response of the circuit to the four possible test vectors is:

A	B	C	SUM		$CARRY$	
0	1	0	0	*	0	
0	1	1	0		0	*
1	1	0	0		0	*
1	1	1	0	*	1	

Any of the following can be selected as a test vector: ABC = 010(S), ABC = 111(S), ABC = 011(C) and ABC = 110(C).

Break bdn3: Causes the source of transistor N3 to be disconnected from GND, resulting in node $\overline{C}$ being stuck-at 1. The circuit response to the four candidate test vectors is:

A	B	C	SUM		$CARRY$	
0	0	1	0	*	0	
0	1	1	0		0	*
1	0	1	0		0	*
1	1	1	0	*	1	

Any of the following can be chosen as a test vector: ABC = 001(S), ABC = 111(S), ABC = 011(C) or ABC = 101(C).

Break bdn4: Causes the source of N6 to be disconnected from GND making it impossible to discharge the output through the path {N6 N5 N4}. The input vector that activates this path is ABC = 110, which must be preceded by an initialisation vector that can be any input vector that sets the output to logic 1. Therefore, the fault is detected by the pair of vectors denoted $ABC = (c, 110)(S)$ where c stands for any input combination that charges the output in question, in this case S.

Break bdn5: Disconnects the source of N5 from the drain of N6. It has the same effect as *bdn4* and it is detected by the same pair of vectors.

Break bdn6: Disconnects the source of N4 from the drains of N6 and N7. The possible pairs of vectors that detect this fault are $ABC = (c, 110)(S)$ or $ABC = (c, 000)(S)$.

Break bdn7: Disconnects the source of N8 from GND. Path {N8 N7 N4} needs to be activated to detect such a fault, which requires ABC = 000 and the pair of vectors is $ABC = (c, 000)(S)$.

Break bdn8: Disconnects the source of N7 from the drain of N8. It has the same effect as *bdn7* and it is detected by the same test pair.

Break bdn9: Disconnects the source of N11 from GND. It is detected by the two-pattern test $ABC = (c, 101)(S)$.

Break bdn10: Disconnects the source of N10 form the drain of N11. It has the same effect as *bdn9* and is detected by the same pair of vectors.

Break bdn11: Disconnects the source of N13 from GND. $ABC = (c, 011)(S)$.

Break bdn12: Disconnects the source of N12 form the drain of N13. It has the same effect as *bdn11* and is detected by the same pair of vectors.

Break bdn13 Disconnects the source of N9 from the drains of N10 and N12. The two possible test pairs are $ABC = (c, 101)(S)$ or $ABC = (c, 011)(S)$.

Break bdn14: Disconnects the source of N15 from GND. $ABC = (c, 010)(C)$.

Break bdn15: Disconnects the source of N14 from the drains of N15 and N16. The possible test pairs are $ABC = (c, 010)(C)$ or $ABC = (c, 100)(C)$.

Break bdn16: Disconnects the source of N16 from GND. $ABC = (c, 100)(C)$.

Break bdn17: Disconnects the source of N18 from GND. $ABC = (c, 001)(C)$.

Break bdn18: Disconnects the source of N17 from the drain of N18. It has the same effect as *bdn17* and is detected by the same test pair.

A.2 SHORT CIRCUITS IN THE DIFFUSION LAYER

Shorts in the diffusion layer may be caused by extra patterns. The nature of the short circuit is dependent on the size and shape of the extra pattern. It is assumed that these extra patterns are small enough so that they affect only physically close regions. To study their effects it is helpful to extract the resulting faulty circuit, analyze its operation and compare it to the fault-free circuit.

Figure 3.5 shows the possible shorts in the pull-up networks. Shorts between two diffusion regions that are always at the same potential will have no effect, and will not be considered. Examples of such shorts are *sdp1*, *sdp2*, *sdp14*, and *sdp19*. When two nodes are shorted and the input combination is such that the two nodes are at the same potential, the short will not affect the logical value of the circuit output. Therefore, a test for such a fault is derived by searching the set of input combinations that set the two nodes to different potentials. In addition, the test must also propagate the fault to an observable output.

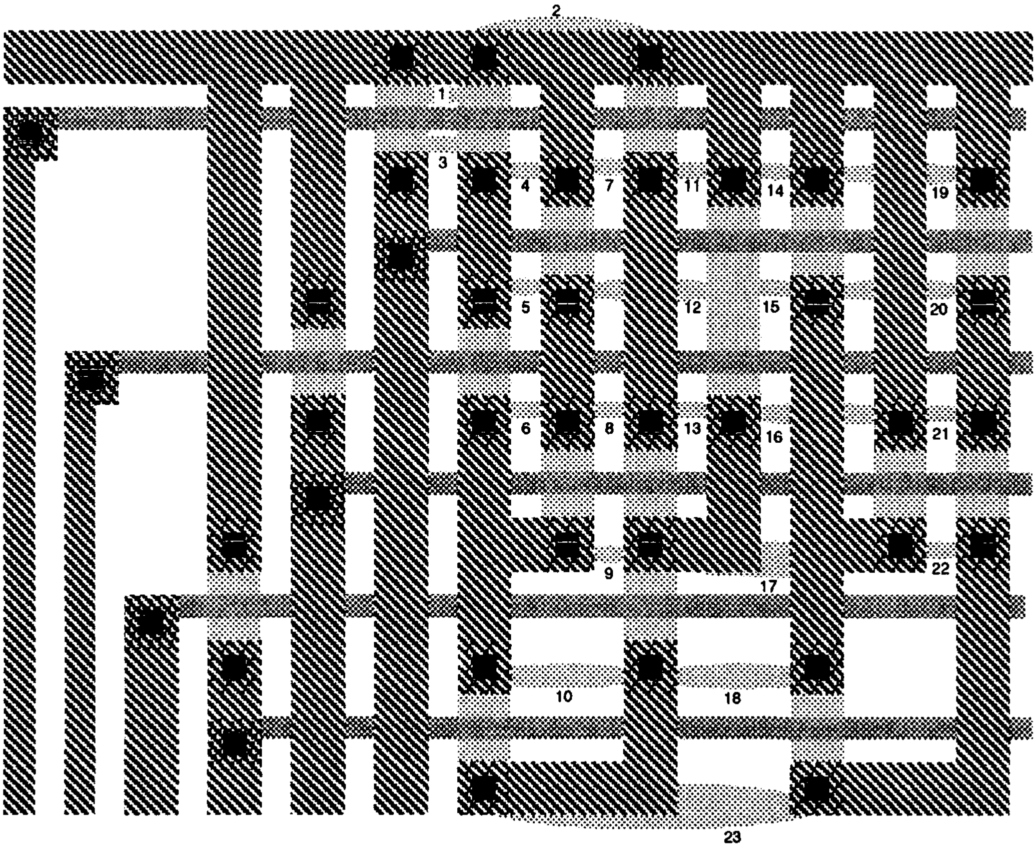


Figure A.3 Location of short circuits in the diffusion layer of the pull-up network.

Short sdp3: Short *sdp3* causes the output of the inverter $\overline{A}$ to be shorted to node $S1$. If we set A to zero, $\overline{A}$ would be at logic one and so would node $S1$. Hence A must be set to 1. This will cause node $S1$ to take on the value 0, which it never gets in a fault free circuit. To propagate the value of node $S1$ to the sum output we must set B to 0 and C to 1. However, with an input ABC = 101, the output will be low, masking the effect of the fault. Therefore, the fault is undetectable since it does not affect the operation of the circuit.

Short sdp4: Short *sdp4* connects the drain of P4 to VDD, making it stuck-at 1. If A is set to zero, node $S1$ will be at logic 1, regardless of the presence of the fault. We must therefore set A to 1, and in order to propagate the state of node $S1$ to the output SUM, B and C must be set to 0 and 1 respectively. With input ABC = 101, the output is discharged through path {N11 N10 N9}. However, because path {P6 P5} is conducting and node $S1$ is at VDD, we are in the general situation where both pull-up and pull-down networks are conducting. The output will be somewhere between VDD and GND, the exact voltage being a strong function of the ratio of the impedances of the paths that are conducting, which itself is a function of devices dimensions and shapes.

If another gate is driven by an output node at such an intermediate voltage, this gate may see it as either logic 1 or 0. If the signal is seen as the same as the fault-free value, we say that the fault is not detected. Otherwise it is detected. For all subsequent faults that result in a similar situation as above, we will derive an input vector that will propagate the fault effect to the output, but fault detection is not guaranteed in this case.

For short *sdp4*, the input vector that propagates the fault effect to the output is $ABC = 101(S)$.

Short sdp5: Short *sdp5* causes the drains of P7 and P4 to be shorted, creating new paths in the pull-up. The two new additional paths created by this fault are {P7 P5 P6} and {P4 P8 P6}. Activating either of these paths will lead to both pull-up and pull-down networks being conducting. This is achieved by either $ABC = 101(S)$ or $ABC = 011(S)$.

Short sdp6: Short *sdp6* cause the drain of P5 and P7 to be shorted, creating new paths in the pull-up. The additional paths are {P7 P6}, and {P4 P5 P8 P6}. However, path {P4 P5 P8 P6} cannot be activated since it requires B and $\overline{B}$ to be zero. To activate {P7 P6} we need $A = 1$ and $C = 1$. The value of B must then be chosen so that the sum output is low. Therefore, the input $ABC = 101(S)$ will propagate the fault effect to the sum output.

Short sdp7: Causes node $S4$ to be stuck-at 1. This can also be seen as the addition of the extra path {P10 P13}. To activate this path, we need $BC = 10$ and A must be selected so that the output is low in a fault-free circuit. Therefore, the test vector is $ABC = 110(S)$.

Short sdp8: Causes nodes $S4$ and $S3$ to be shorted. The additional paths are {P7 P10 P13} and {P9 P8 P6} which are activated by $ABC = 110$ and $ABC = 011$, respectively. The two possible test vectors are therefore, $ABC = 110(S)$ or $ABC = 011(S)$.

Short sdp9: Causes nodes $S2$ and $S6$ to be shorted, creating the extra paths {P4 P5 P13}, {P7 P8 P13}, {P9 P10 P6} and {P11 P12 P6}. These paths are activated by $ABC = 000, 110, 011$ and 101, respectively. Therefore, any input combination that discharges the output would detect this fault. This is noted as $ABC = d(S)$.

Short sdp10: Causes node $S2$ and the sum output to be shorted. The additional paths are {P4 P5} and {P7 P8}. The first path is activated by setting A and B to zero, while C must be chosen so that the sum output is low, resulting in the first vector $ABC = 000(S)$. A second test vector can be obtained by activating the second path ($AB = 11$) and choosing C so that the output is low, giving $ABC = 110(S)$ as another possible test vector.

Short sdp11: Causes node $S4$ to stuck-at 1. Same as *sdp7.*

Short sdp12: Causes nodes $S3$ and $S5$ to be shorted. The extra paths created are {P11 P8 P6} and {P7 P12 P13}. The first path is activated by ABC = 111 and the second by ABC = 100. However, when either path is activated, a legitimate path in the pull-up network is also activated, masking the fault. Therefore, the fault is not detectable. Transistors P11 and P7 can be merged into a single one.

Short sdp13: Causes nodes $S6$ and $S4$ to be shorted. The additional paths in the pull-up are {P9 P13} and {P11 P12 P10 P13}. However, path {P11 P12 P10 P13} cannot be activate since it requires B and $\overline{B}$ to be set to zero. Therefore, the test for this short is ABC = 000(S).

Short sdp14: No effect. Both nodes are at VDD.

Short sdp15: Causes node $S5$ of the sum circuit to be shorted to node $S13$ of the carry circuit, creating extra paths in both circuits. The paths created in the sum circuit are {P14 P12 P13} and {P16 P12 P13}. When the first path is activated, another legitimate path in the sum circuit is also activated. The second path cannot be activated.

The path created in the carry circuit is {P11 P15} which is activated by an input combination that also activates a legitimate path in the circuit. Hence, this fault is not detectable since it does not affect the operation of the circuit.

Short sdp16: Causes node $S6$ in the sum circuit to be stuck-at 1. Hence, setting C to 0 will drive the sum output to logic 1, regardless of the other inputs. The possible test vectors are ABC = 000(S) or ABC = 110(S).

Short sdp17: Causes node $S6$ in the sum circuit and node $S13$ in the carry circuit to be shorted. The extra paths created in the sum circuit are {P14 P13} and {P16 P13}. The first one is activated by AC = 10 which yields the test vector ABC = 110(S). Activating the second path with BC = 10 yielding the same test vector. The extra paths in the carry circuit are {P11 P12 P15} and {P9 P10 P15} which are activated by ABC = 101 and ABC = 011, respectively. However, since this will set the carry output to logic 1 in both the faulty and fault-free circuits, these inputs do not constitute test vectors for this fault.

Short sdp18: Causes node $S13$ in the carry circuit to be shorted to the sum output. Two new paths are added to the sum circuit: {P14} and {P16}. Therefore, setting A, B or both, to 1 will drive the sum output to 1, regardless of the other values. The other values should be chosen so that the output is low in a fault-free situation, giving the three possible test vectors: ABC = 011(S), ABC = 101(S), ABC = 110(S).

Additional paths are also created in the carry circuit; {P4 P5 P6 P15} and {P7 P8 P6 P15} being the activable ones. The second path is activated by ABC = 111, but this will result in a carry output being high, i.e., the same as the fault-free response. Activating the first path requires ABC = 001, which will set the carry output to zero in a fault-free circuit, but if the fault is present there will also be a path from carry to VDD, so ABC = 001(C) is another test vector. Vectors ABC = 011 and ABC = 101 may also detect the fault at the carry output since in this case the sum output is low, inducing an intermediate voltage at node $S13$.

Short sdp19: No effect. Both nodes are at VDD.

Short sdp20: Causes nodes $S14$ and $S13$ of the carry circuit to be shorted. This will create three new paths in the circuit. The first {P14 P18} when activated results in a legitimate path in the pull-up being also activated. The second, {P17 P15} also leads to the same situation. However, the third path {P16 P18} can be activated by simply setting B to 1. To produce a fault at the output, the other inputs must be set so that the fault-free response is low. Therefore, the test vector for this fault is ABC = 010(C).

Short sdp21: Causes node $S14$ to be stuck-at 1. Setting B to 1 will cause the carry output to be at logic 1, regardless of the other inputs. Therefore, the test vector for this fault is ABC = 010(C).

Short sdp22: Cause node $S13$ and the carry output to be shorted. Setting A or B to logic 1 will set the output to logic 1. ABC = 001(C), ABC = 100(C) and ABC = 010(C) are all possible test vectors.

Short sdp23: Causes the sum and the carry outputs to be shorted. Since both nodes are observable outputs, any input combination that sets sum and carry to opposite values is a test vector. ABC = 001(SC), ABC = 010(SC), ABC = 100(SC), ABC = 011(SC), ABC = 101(SC), ABC = 110(SC).

Figure 3.7 shows the possible shorts in the diffusion layer of the pull-down networks.

Short sdn1: Short *sdn1* causes the output of the inverter $\overline{A}$ to be shorted to node $S8$. This fault does not affect circuit operation and thus is undetectable.

Short sdn2: Causes node $S8$ to be stuck-at 0, creating the new path {N5 N4} in the pull-down, which can be activated by setting BC = 10. Hence, A must be set so that the output is high in a fault-free circuit. The test vector ABC = 010(S) achieves this, and if the fault is present, both paths {P9 P10 P13} and {N6 N4} are conducting.

Short sdn3: Causes nodes $S8$ and $S9$ to be shorted. The paths created by this short are {N6 N7 N4} and {N8 N5 N4}. They are activated by ABC = 100 and ABC = 010,

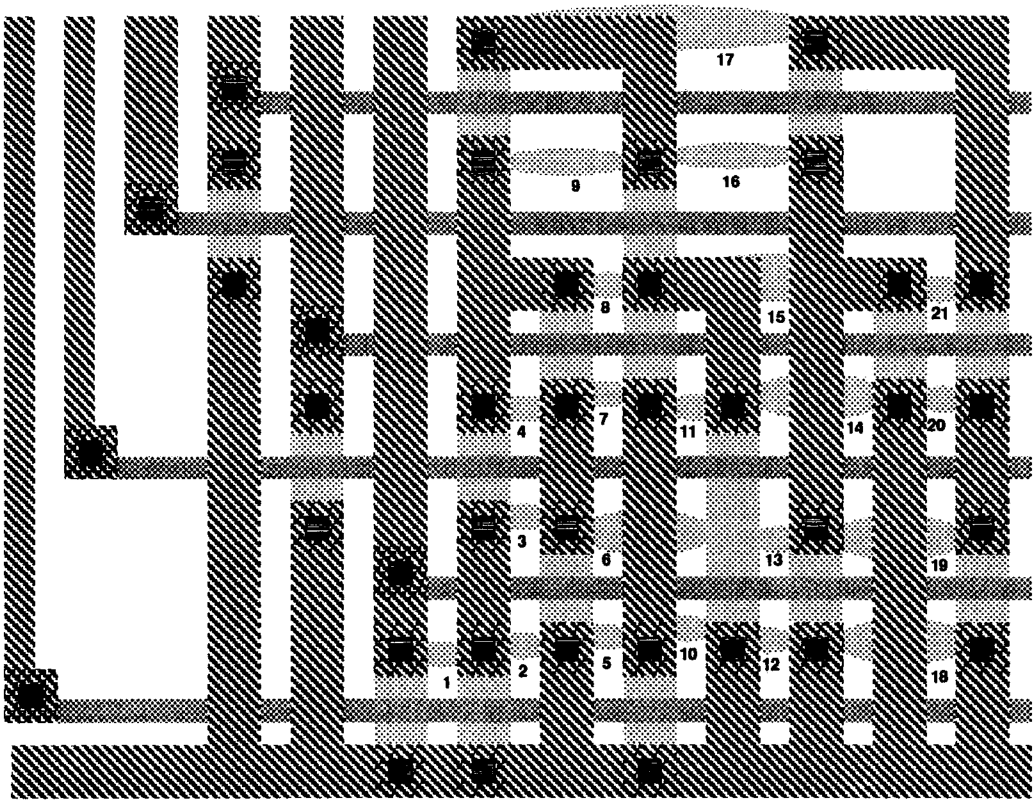


Figure A.4 Location of short circuits in the diffusion layer of the pull-down network.

respectively. Both input combinations would set the sum output to logic 1 in a fault-free circuit, and so either ABC = 100(S) or ABC = 010(S) can be used to test for this fault.

Short sdn4: Causes nodes $S7$ and $S9$ to be shorted, adding the path {N8 N4}, which is activated by AC = 00, so that ABC = 010(S) is a test vector for this fault.

Short sdn5: Causes node $S10$ to be stuck-at 0 adding the extra path {N10 N9} which is activated by BC = 01 and hence ABC = 001(S) is a test vector.

Short sdn6: Causes nodes $S12$ and $S10$ to be shorted creating the paths {N11 N12 N9} and {N13 N10 N9} which can be activated with ABC = 111 and ABC = 001, respectively, and the test vectors are ABC = 111(S) or ABC = 001(S).

Short sdn7: Causes nodes $S10$ and $S9$ to be shorted adding the paths {N8 N10 N9} and {N11 N7 N4} which are activated by ABC = 001 and ABC = 100, respectively. Both input combinations set the sum output to logic 1 in a fault-free circuit and therefore, constitute possible test vectors for this fault. ABC = 001(S) or ABC = 100(S).

Short sdn8: Causes a short between nodes $S11$ and $S7$ creating the extra paths {N6 N5 N9}, {N8 N7 N9}, {N11 N10 N4}, and {N13 N12 N4} which are activated by

ABC = 111,001,100, and 010, respectively, i.e., any input combination that charges the sum output is a test vector. This is denoted as ABC = c(S).

Short sdn9: Causes the sum output to be shorted to node $S7$, creating the new paths {N6 N5} and {N8 N7} which are activated by AB = 11 and AB = 00. Input C must be set to 1 in order to detect the fault. ABC = 111(S) and ABC = 001(S) are both test vectors for this fault.

Short sdn10: Causes node $S10$ to be stuck-at 0. Detected by ABC = 001(S).

Short sdn11: Causes nodes $S11$ and $S10$ to be shorted creating the path {N11 N9} activated by AC = 11. Input B must be set to 1 in order to detect this fault. The test vector is ABC = 111(S).

Short sdn12: No effect. The shorted nodes are at GND.

Short sdn13: Causes nodes S12 in the sum circuit and S15 in the carry circuit to be shorted. The extra paths created in the sum circuit are {N15 N12 N9} and {N16 N12 N9}. When the first path is activated a legitimate path in the pull-down is also activated. The second path cannot be activated since it requires B and $\overline{B}$ to be set to 1. The extra path created in the carry circuit is {N13 N14} which is activated by AC = 00. Whatever value we choose for input B, the carry output will always be low when AC = 00. Therefore, this fault is not detectable.

Short sdn14: Causes node S11 to be stuck-at 0. In this case setting input C to 1 will result in the sum output being at logic 0. Therefore, the possible test vectors are ABC = 001(S) and ABC = 111(S).

Short sdn15: Causes nodes S15 and S11 to be shorted. The extra paths in the sum circuit are {N15 N9} and {N16 N9} activated by AC = 01 and BC = 01, respectively, yielding the same test vector ABC = 001(S). The extra paths in the carry circuit are {N13 N12 N14} and {N11 N10 N14} which are activated by ABC = 010 and ABC = 100. Both input vectors would produce a low output in a fault-free carry circuit, and therefore, do not constitute test vectors.

Short sdn16: Causes the sum output and node S15 to be shorted. Two extra paths are added to the sum circuit: {N15} and {N16}. Hence, setting A, B, or both to 0 will force the sum output to 0. Therefore, ABC = 001(S), ABC = 010(S) and ABC = 100(S) are all possible test vectors for this fault. In addition, input vectors ABC = 110, ABC = 010 and ABC = 100 also produces an intermediate voltage on the carry output giving the other possible test vector ABC = 110(C), ABC = 010(C) or ABC = 100(C).

Short sdn17: Causes the sum and the carry outputs to be shorted. Any input combination that sets the two output to different values is likely to detect this fault. The possible test vectors are ABC = 001(SC), ABC = 010(SC), ABC = 011(SC), ABC = 100(SC), ABC = 101(SC), ABC = 110(SC).

Short sdn18: No effect. The shorted nodes are at GND.

Short sdn19: Causes nodes S16 and S15 to be shorted, creating the extra paths {N15 N17} and {N16 N17}. Activating the first path would result in a legitimate path in the pull-down network being activated. The second path is activated by setting B to 0 yielding the test vector ABC = 101(C).

Short sdn20: Causes node S16 to be stuck-at 0. Setting B to 0 will force the carry output to 0. Therefore the test vector is ABC = 101(C).

Short sdn21: Causes the carry output to be shorted to node S15 adding the two extra paths {N15} and {N16}. Hence setting A, B, or both to 0 will force the $CARRY$ output to 0, yielding the test vectors ABC = 001(C), ABC = 010(C), and ABC = 100(C).

A.3 OPEN CIRCUITS IN THE POLYSILICON LAYER

Open circuits in the polysilicon layer result in the gates of transistors being floating. The state of a transistor (ON or OFF) with a floating gate is difficult to determine. The transistor state depends on the charge present on the gate (although this charge may leak away after a while). In addition, the charge on the floating gate is very sensitive to coupling from neighbouring or overlapping lines. Therefore, the transistor may be ON or OFF, but in both cases it will be a 'weak' ON or OFF state, that is, if the transistor is ON then its resistance would be much higher than a normally ON transistor. Similarly, if the transistor is OFF, the current from source to drain is far from being negligible.

Because polysilicon is used as a mask when diffusing the source and drain regions, an open circuit may also result in a transistor being stuck-closed, but in this case it will have its drain and source shorted. In the following, we will consider that a transistor that has a floating gate is either stuck-on or stuck-open and derive test vectors for both cases. Figure 3.8 shows the possible locations of breaks in the polysilicon layer of the pull-up network.

In the first part of this section, we will assume that any transistor with a floating gate is stuck-closed. The second part of this section will consider the case where it is stuck-open. Transistor stuck-on faults behave in the same way as many of the diffusion shorts of the previous section in that they give rise to intermediate voltages. Hence,

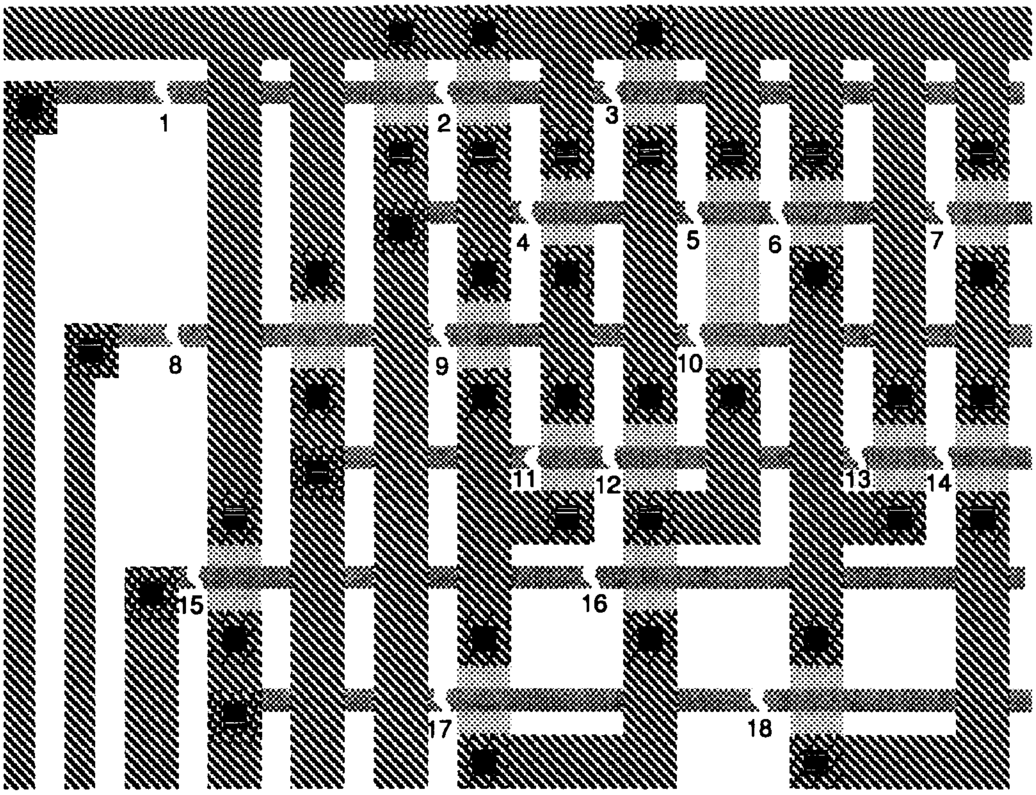


Figure A.5 Locations of the possible breaks in the polysilicon layer of the pull-up network.

depending on how far is the intermediate voltage from the fault-free value, the fault may or may not be detected. In deriving tests for such faults we will try to induce an intermediate voltage at the fault location and then propagate it to an observable output. The detectability of the fault will depend on whether any subsequent gate, driven by the observable output, sees the output as the same or not as the fault-free one.

Break bpp1: Break *bpp1* affects P1, P4 and P9. Since these transistors are in the pull-up networks, the test vectors are selected from the set of input combinations that turn the pull-down on and activate the path containing an affected transistor. For P1, any input combination with $A = 1$ will produce an intermediate voltage on output $\overline{A}$. However, whether this intermediate voltage is propagated to the sum and/or carry outputs, and whether it is far enough from the fault-free value, can only be determined by a circuit simulation. It is also possible to test for this fault by activating the paths containing P4 and P9. The possible test vectors for this fault are ABC = 100(SC), ABC = 101(SC), ABC = 110(SC), and ABC = 111(SC).

Break bpp2: Transistors P4 and P9 are stuck-closed. The possible test vectors are ABC = 101(S) and ABC = 110(S).

Break bpp3: P9 stuck-closed. Detected by ABC = 110(S).

Break bpp4: P7, P11, P14 and P17 are stuck-on. The fault is detected by ABC = 000(S), ABC = 001(C), ABC = 010(C) or ABC = 011(S).

Break bpp5: P11, P14 and P17 are stuck-on. The test vectors are ABC = 000(S), ABC = 001(C) and ABC = 010(C).

Break bpp6: P14 and P17 are stuck-on. Test vectors: ABC = 001(C) or ABC = 010(C).

Break bpp7: P17 is stuck-on. Test vector: ABC = 010(C).

Break bpp8: P2, P5 and P12 are stuck-on. Test vectors: ABC = 010(SC), ABC = 011(SC), ABC = 110(SC) and ABC = 111(SC).

Break bpp9: P5 and P12 are stuck-on. Test vectors: ABC = 011(S) and ABC = 110(S).

Break bpp10: P12 is stuck-on. Test vector: ABC = 110(S).

Break bpp11: P8, P10, P16 and P18 are stuck-on. Test vectors are ABC = 000(S), ABC = 001(C), ABC = 100(C), and ABC = 101(S).

Break bpp12: P10, P16 and P18 are stuck-on. Test vectors: ABC = 000(S), ABC = 001(C), and ABC = 100(C).

Break bpp13: P16 and P18 are stuck-on. Test vectors: ABC = 001(C) and ABC = 100(C).

Break bpp14: P18 stuck-on. Test vector: ABC = 100(C).

Break bpp15: P3 and P13 stuck-on. Test vectors ABC = 001(SC), ABC = 011(SC), ABC = 101(SC), and ABC = 111(SC).

Break bpp16: P13 stuck-on. Test vectors ABC = 011(S) and ABC = 101(S).

Break bpp17: P6 and P15 stuck-on. Test vectors ABC = 000(S), ABC = 010(C), ABC = 100(C), and ABC = 110(S).

Break bpp18: P15 stuck-on. Test vector ABC = 010(C) or ABC = 100(C).

Figure 3.9 shows the locations of possible open circuits in the polysilicon layer of the pull-down networks.

Break bpn1: Transistors N1, N6 and N11 are stuck-closed. The possible test vectors are ABC = 000(SC), ABC = 001(SC), ABC = 010(SC), or ABC = 011(SC).

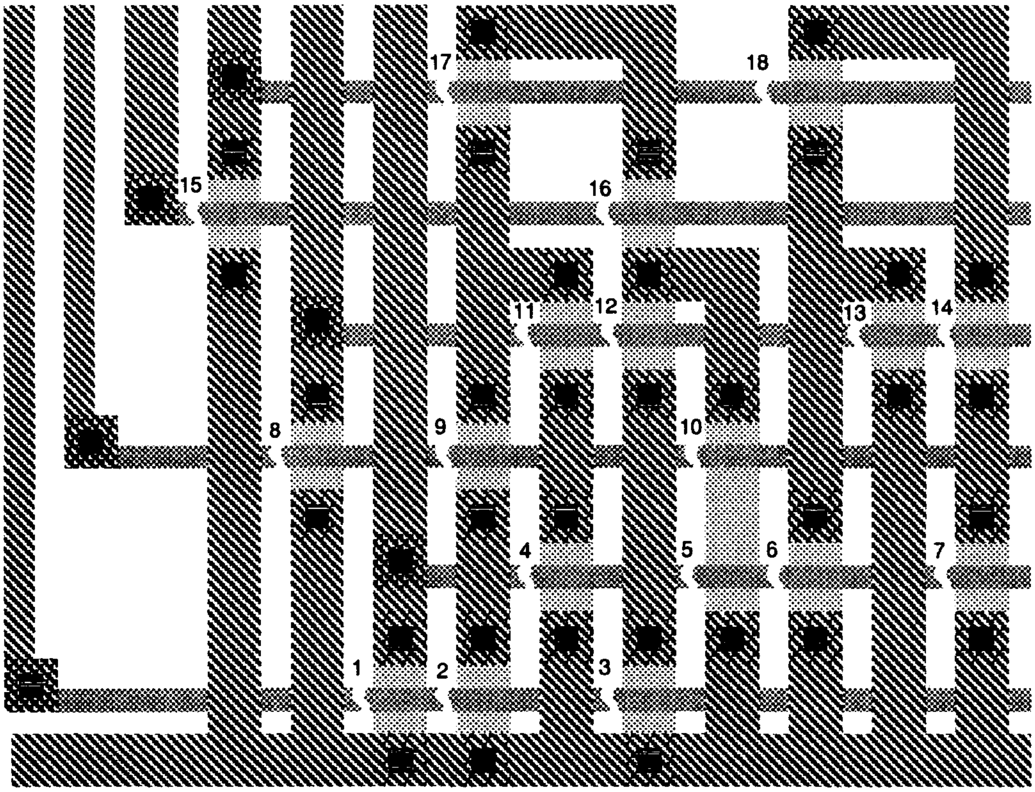


Figure A.6 Breaks in the polysilicon layer of the pull-down network.

Break bpn2: N6 and N11 stuck-closed. Test vectors ABC = 001(S) or ABC = 010(S).

Break bpn3: N11 stuck-on. Test vector ABC = 001(S).

Break bpn4: N8, N13, N15 and N18 stuck-closed. Test vector ABC = 100(S), ABC = 101(C), ABC = 110(C), or ABC = 111(S).

Break bpn5: N13, N15 and N18 stuck-closed. Test vector ABC = 101(C), ABC = 110(C), or ABC = 111(S).

Break bpn6: N15 and N18 stuck-on. Test vector ABC = 101(C) or ABC = 110(C).

Break bpn7: N18 stuck-on. Test vector ABC = 101(C).

Break bpn8: N2, N5 and N12 stuck-on. Test vector ABC = 000(SC), ABC = 001(SC), ABC = 100(SC) or ABC = 101(SC).

Break bnp9: N5 and N12 stuck-on. Test vector ABC = 001(S) or ABC = 100(S).

Break bnp10: N12 stuck-on. Test vector ABC = 001(S).

Break bnp11: N7, N10, N16 and N17 stuck-closed. Test vector ABC = 010(S), ABC = 011(C), ABC = 110(C) or ABC = 111(S).

Break bnp12: N10, N16 and N17 stuck-closed. Test vector $ABC = 011(C)$, $ABC = 110(C)$ or $ABC = 111(S)$.

Break bnp13: N16 and N17 stuck-closed. Test vector $ABC = 011(C)$ or $ABC = 110(C)$.

Break bnp14: N17 stuck-on. Test vector $ABC = 011(C)$.

Break bnp15: N3 and N9 stuck-closed. Test vector $ABC = 000(SC)$, $ABC = 010(SC)$, $ABC = 100(SC)$ or $ABC = 110(SC)$.

Break bnp16: N9 stuck-on. Test vector $ABC = 010(S)$ or $ABC = 100(S)$.

Break bnp17: N4 and N14 stuck-closed. Test vector $ABC = 001(S)$, $ABC = 011(C)$, $ABC = 101(C)$ or $ABC = 111(S)$.

Break bnp18: N14 stuck-on. Test vector $ABC = 011(C)$ or $ABC = 101(C)$.

Next, the case where a transistor with a floating gate is permanently off is considered.

Break bpp1: Transistors P1, P4 and P9 are stuck-open. P1 stuck-open causes $\overline{A}$ to be stuck-at 0. This yields the test vector $ABC = 000(S)$, $ABC = 001(C)$, $ABC = 010(C)$ or $ABC = 011(S)$. It is also possible to test for this break with a pair of vectors.

Break bpp2: P4 and P9 stuck-open. The test pair is $ABC = (d, 001)(S)$ or $ABC = (d, 010)(S)$.

Break bpp3: P9 stuck-open. Test pair $ABC = (d, 010)(S)$.

Break bpp4: P7, P11, P14 and P17 stuck-open. Test pair $ABC = (d, 100)(S)$, $ABC = (d, 101)(C)$, $ABC = (d, 110)(C)$ or $ABC = (d, 111)(S)$.

Break bpp5: P11, P14 and P17 stuck-open. Test pair $ABC = (d, 100)(S)$, $ABC = (d, 101)(C)$ or $ABC = (d, 110)(C)$.

Break bpp6: P14 and P17 stuck-open. Test pair $ABC = (d, 101)(C)$ or $ABC = (d, 110)(C)$.

Break bpp7: P17 stuck-open. Test pair $ABC = (d, 110)(C)$.

Break bpp8: P2, P5 and P12 stuck-open. $\overline{B}$ is stuck-at 0 as a result. Test vector $ABC = 000(S)$, $ABC = 001(C)$, $ABC = 100(C)$ or $ABC = 101(S)$.

Break bpp9: P5 and P12 stuck-open. Test pair $ABC = (d, 001)(S)$ or $ABC = (d, 100)(S)$.

Break bpp10: P12 stuck-open. Test pair $ABC = (d, 100)(S)$.

Break bpp11: P8, P10, P16 and p18 stuck-open. Test pair $ABC = (d, 010)(S)$, $ABC = (d, 011)(C)$, $ABC = (d, 110)(C)$ or $ABC = (d, 111)(S)$.

Break bpp12: P10, P16 and P18 stuck-open. Test pair $ABC = (d, 010)(S)$, $ABC = (d, 011)(C)$ or $ABC = (d, 110)(C)$.

Break bpp13: P16 and P18 stuck-open. Test pair $ABC = (d, 011)(C)$ or $ABC = (d, 110)(C)$.

Break bpp14: P18 stuck-open. Test pair $ABC = (d, 110)(C)$.

Break bpp15: P3 and P13 stuck-open. $\overline{C}$ is stuck-at 0 as a result. Test vector $ABC = 000(S)$, $ABC = 010(C)$, $ABC = 100(C)$ or $ABC = 110(S)$.

Break bpp16: P13 stuck-open. Test pair $ABC = (d, 010)(S)$ or $ABC = (d, 100)(S)$.

Break bpp17: P6 and P15 stuck-open. Test pair $ABC = (d, 001)(S)$, $ABC = (d, 011)(C)$, $ABC = (d, 101)(C)$ or $ABC = (d, 111)(S)$.

Break bpp18: P15 stuck-open. Test pair $ABC = (d, 011)(C)$ or $ABC = (d, 101)(C)$.

Break bpn1: Transistors N1, N6 and N11 are stuck-open. $\overline{A}$ is stuck-at 1. Test vector $ABC = 100(S)$, $ABC = 101(C)$, $ABC = 110(C)$, or $ABC = 111(S)$.

Break bpn2: N6 and N11 stuck-open. Test pair $ABC = (c, 101)(S)$ or $ABC = (c, 110)(S)$.

Break bpn3: N11 stuck-open. Test pair $ABC = (c, 101)(S)$.

Break bpn4: N8, N13, N15 and N18 stuck-open. Test pair $ABC = (c, 000)(S)$, $ABC = (c, 001)(C)$, $ABC = (c, 010)(C)$ or $ABC = (c, 011)(S)$.

Break bpn5: N13, N15 and N18 stuck-open. Test pair $ABC = (c, 001)(C)$, $ABC = (c, 010)(C)$ or $ABC = (c, 011)(S)$.

Break bpn6: N15 and N18 stuck-open. Test pair $ABC = (c, 001)(C)$ or $ABC = (c, 010)(C)$.

Break bpn7: N18 stuck-open. Test pair $ABC = (c, 001)(C)$.

Break bpn8: N2, N5 and N12 stuck-open. $\overline{B}$ is stuck-at 1. Test vector $ABC = 010(S)$, $ABC = 011(C)$, $ABC = 110(C)$ or $ABC = 111(S)$.

Break bnp9: N5 and N12 stuck-open. Test pair ABC = $(c,011)(S)$ or ABC = $(c,110)(S)$.

Break bnp10: N12 stuck-open. Test pair $ABC = (c,011)(S)$.

Break bnp11: N7, N10, N16 and N17 stuck-open. Test pair ABC = $(c,000)(S)$, $ABC = (c,001)(C)$, $ABC = (c,100)(C)$ or $ABC = (c,101)(S)$.

Break bnp12: N10, N16 and N17 stuck-open. Test pair ABC = $(c,001)(C)$, ABC = $(c,100)(C)$ or $ABC = (c,101)(S)$.

Break bnp13: N16 and N17 stuck-open. Test pair ABC = $(c,001)(C)$ or ABC = $(c,100)(C)$.

Break bnp14: N17 stuck-open. Test pair $ABC = (c,001)(C)$

Break bnp15: N3 and N9 stuck-open. $\overline{C}$ is stuck-at 1. Test vector ABC = $001(S)$, $ABC = 011(C)$, $ABC = 101(C)$ or $ABC = 111(S)$.

Break bnp16: N9 stuck-open. Test pair $ABC = (c,011)(S)$ or $ABC = (c,101)(S)$.

Break bnp17: N4 and N14 stuck-open. Test pair ABC = $(c,000)(S)$, ABC = $(c,010)(C)$, $ABC = (c,100)(C)$ or $ABC = (c,110)(S)$.

Break bnp18: N14 stuck-open. Test pair $ABC = (c,010)(C)$ or $ABC = (c,100)(C)$.

A.4 SHORT CIRCUITS IN THE POLYSILICON LAYER

Figure A.7 shows the locations of shorts in the polysilicon layer for the pull-up network.

Short spp1: Causes A and B to be shorted. The test vectors should be searched among the input combinations for which A and B have different values. In fact, any such input combination is a test vector: ABC = $010(SC)$, ABC = $011(SC)$, ABC = $100(SC)$ and $ABC = 101(SC)$.

Short spp2: Causes B and C to be shorted. Any input combination for which $B \neq C$ is a test vector. $ABC = 001(SC)$, $ABC = 101(SC)$, $ABC = 010(SC)$ or $ABC = 110(SC)$.

Short spp3: Causes A and $\overline{A}$ to be shorted. Any input combination will induce an intermediate voltage on nodes A and $\overline{A}$ that may be propagated to the sum and carry outputs.

Short spp4: Causes $\overline{A}$ and B to be shorted. $\overline{A}$ and B should be set to different values, i.e., any input combination with $A = B$ is a test vector: ABC = $110(SC)$, $ABC = 111(SC)$, $ABC = 001(SC)$ or $ABC = 000(SC)$.

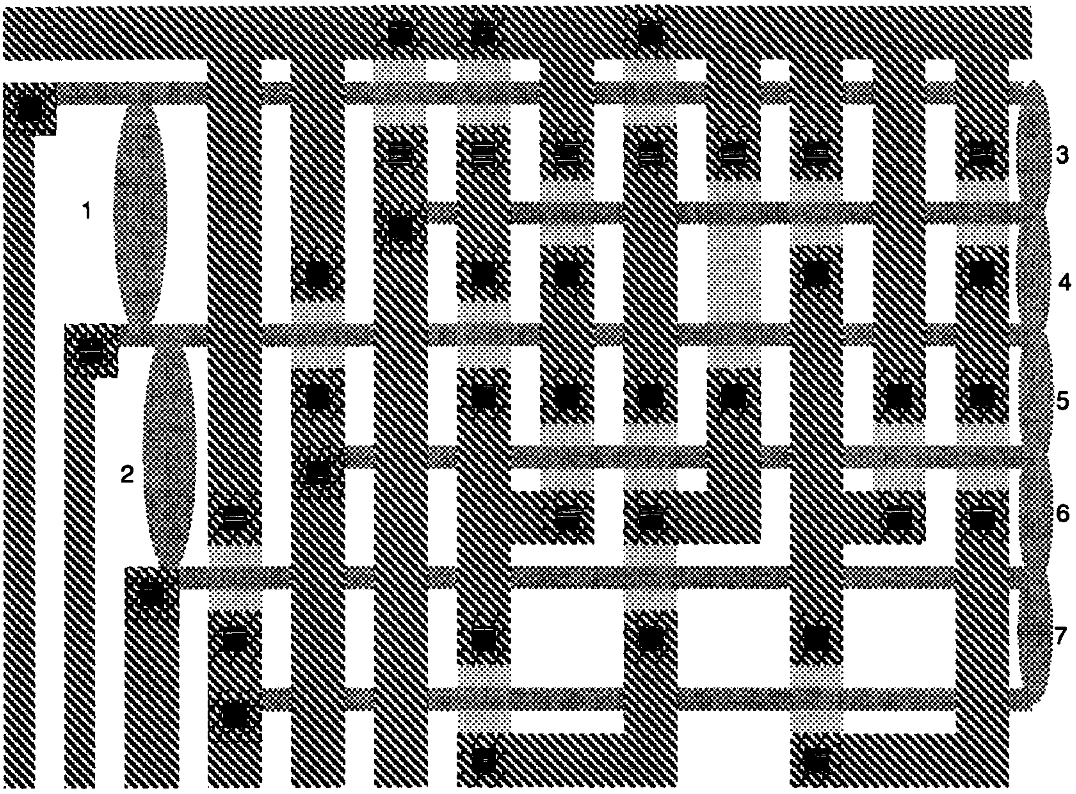


Figure A.7 Locations of short circuits in the polysilicon layer for the pull-up network.

Short spp5: B and $\overline{B}$ are shorted. Detected by any input vector.

Short spp6: $\overline{B}$ and C shorted. We need $B = C$. Therefore, ABC = 011(SC), ABC = 111(SC), ABC = 100(SC) or ABC = 000(SC) can be used as test vector.

Short spp7: C and $\overline{C}$ are shorted. Any input combination is a test vector.

Shorts in the polysilicon tracks in the pull-down are covered by the above ones.

A.5 OPEN CIRCUITS IN THE METAL LAYER

This type of fault has similar effects to open circuits in the diffusion layer, i.e., most of the breaks in the metal layer result in transistors being stuck-open. This section considers only faults that are not covered by the faults encountered in the previous sections. Figure A.8 shows the locations of the breaks in the pull-up network.

Break bmp1: Disconnects all the pull-up of the sum circuit from the output. The sum output is effectively stuck-at 0. Any input combination that attempts to charge the output is a test vector: ABC = c(S).

Break bmp2: Does the same for the carry circuit. ABC = c(C).

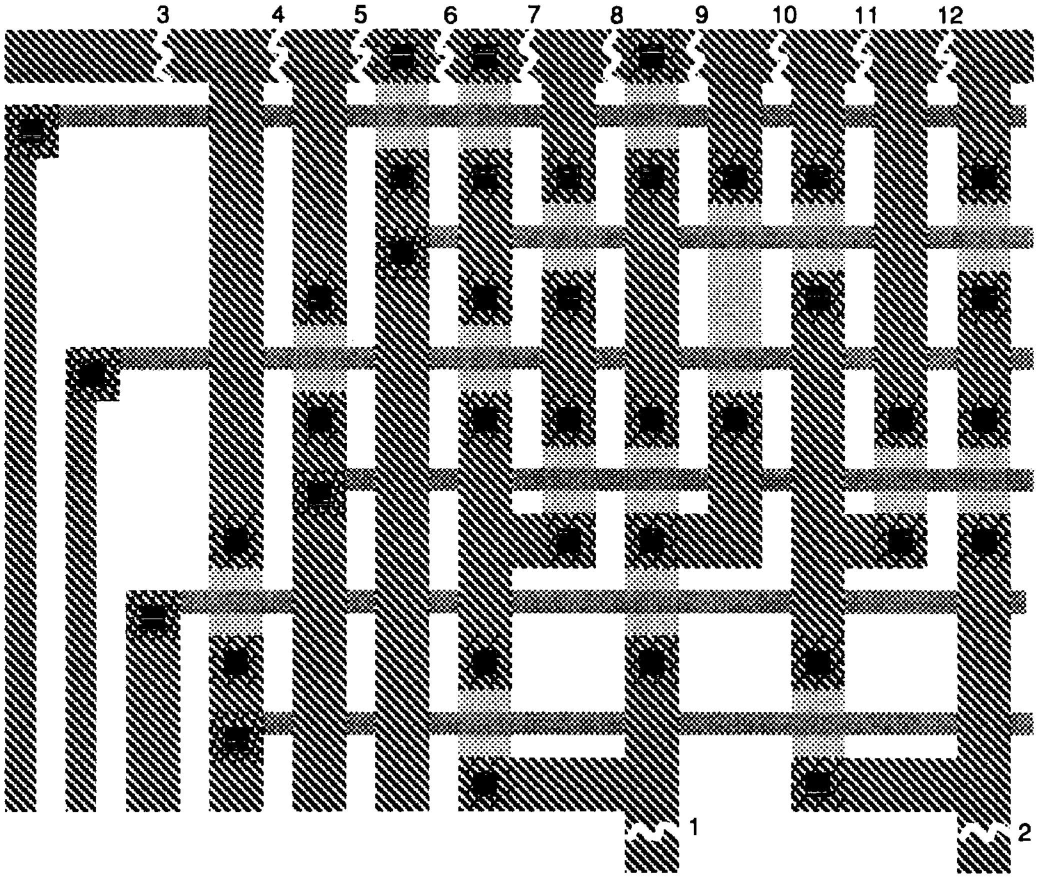


Figure A.8 Breaks in the metal layer of the pull-up network.

Break bmp3: We assume that VDD is supplied from the left of the circuit. $\overline{A}$, $\overline{B}$, and $\overline{C}$ are stuck-at 0. The sum and the carry outputs are floating. It is not possible to charge these outputs because of the break in the VDD line and it is also impossible to discharge them because of the stuck-at 0 lines. If the value held on the floating outputs is logic one, then any input combination that sets them to 0, in a fault-free circuit, will detect the fault. Similarly, if the value held at these nodes is low, any charging input vector will detect the fault. Therefore, to detect this fault, we just need to make sure that the test sequence contains at least one vector that charges the output and another that discharges it.

Break bmp4: $\overline{A}$ and $\overline{B}$ are stuck-at zero. In addition it is impossible to discharge the carry output. Therefore, this fault can be detected at the carry output by ensuring that the test sequence contains at least one vector that discharges the output of a fault-free circuit and one vector that would charge it. Alternatively, this fault can be detected using the test pair $ABC = (110, c)(S)$. After the vector $ABC = 110$ is applied to the circuit, the sum will be stuck-at 0.

Break bmp5: $\overline{A}$ is stuck-at 0. Input combinations ABC = 110 and ABC = 101 will discharge the sum output, and since it is not possible charge it again, it will be stuck-at 0. Similarly, inputs ABC = 100 and ABC = 000 will discharge the carry output and subsequently, it will be stuck-at 0. If we assume that these vectors have already appeared as inputs to the circuit, then we may use any of the following as a test vector: $ABC = 001(S)$, $ABC = 010(S)$, $ABC = 100(S)$, $ABC = 111(SC)$, $ABC = 011(C)$, $ABC = 101(C)$, $ABC = 110(C)$. If these vectors have not appeared previously as inputs to the circuit, then it is necessary to use a pair of vectors to detect this fault. The possible test pairs are $ABC = (110, c)(S)$, $ABC = (101, c)(S)$, $ABC = (100, c)(C)$ and $ABC = (000, c)(C)$.

Break bmp6: The sum and the carry outputs are stuck-at 0. $ABC = 001(S)$, $ABC = 010(S)$, $ABC = 100(S)$, $ABC = 111(SC)$, $ABC = 011(C)$, $ABC = 101(C)$, $ABC = 110(C)$ are all possible test vectors.

Break bmp7: Only the carry output is stuck-at 0 since now it is possible to charge the sum output through path {P4 P5 P6}. The possible test vectors are $ABC = c(C)$.

Break bmp8: Same as *bmp7*. Can also be detected as carry stuck-at 0.

Break bmp9: Same as *bmp7*.

Break bmp10: Same as *bmp7*.

Break bmp11: Test pair: $ABC = (d, 011)(C)$ or $ABC = (d, 110)(C)$.

Break bmp12: Test pair: $ABC = (d, 110)(C)$.

Figure A.9 shows the locations of the open circuits in the metal layer for the pull-down network.

Break bmn1: The sum output is stuck-at 1. $ABC = d(S)$.

Break bmn2: The carry output is stuck-at 1. $ABC = d(C)$.

Break bmn3: $\overline{A}$, $\overline{B}$ and $\overline{C}$ are stuck-at 1. The sum and carry outputs will be floating. This can be detected in the same way as *bmp3*.

Break bmn4: $\overline{A}$ and $\overline{B}$ stuck-at 1. $CARRY$ is floating. Can be detected in the same way as *bmn3*.

Break bmn5: $\overline{A}$ is stuck-at 1. Test pair $ABC = (001, d)(S)$, $ABC = (010, d)(S)$, $ABC = (111, d)(C)$ or $ABC = (011, d)(C)$.

Break bmn6: The sum and carry outputs are stuck-at 1. $ABC = d(S)$ or $ABC = d(C)$.

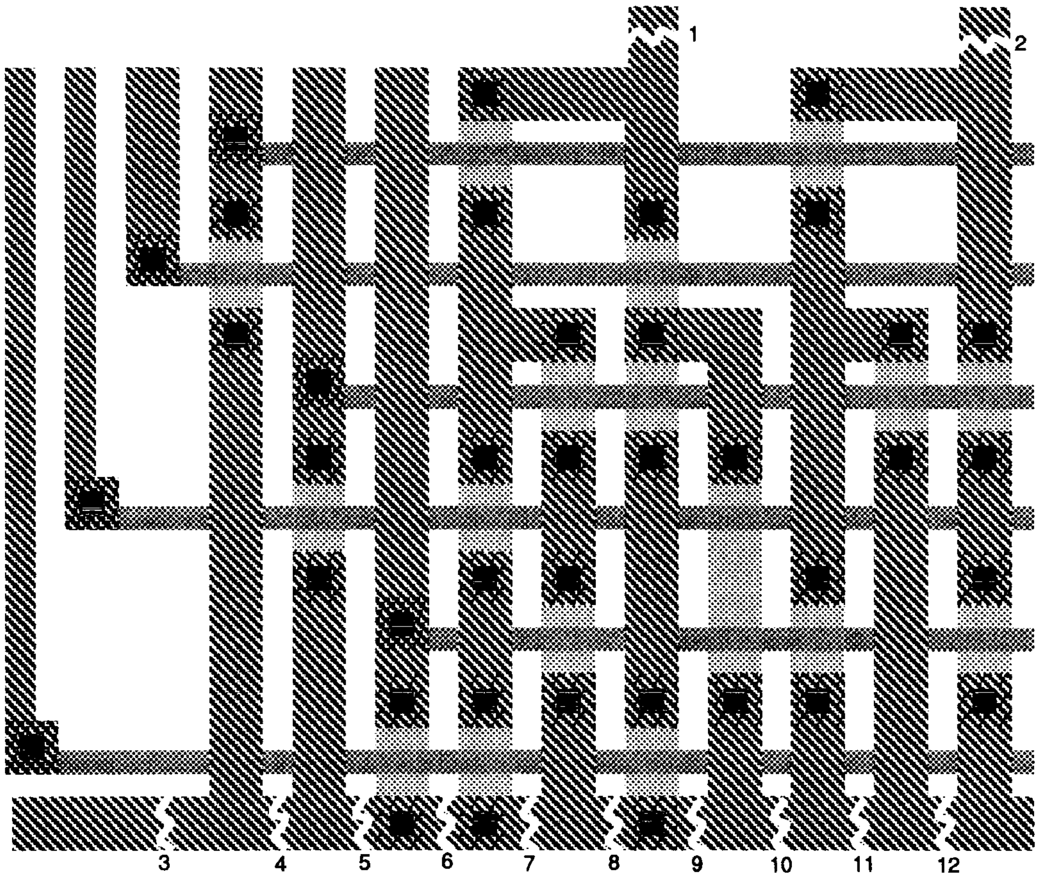


Figure A.9 Breaks in the metal layer of the pull-down network.

Break bmn7: The carry output is stuck-at 1. $ABC = d(C)$.

Break bmn8: Same as *bmn7.*

Break bmn9: Same as *bmn7.*

Break bmn10: same as *bmn7.*

Break bmn11: Test pair: $ABC = (c, 100)(C)$ or $ABC = (c, 001)(C)$.

Break bmn12: Test pair $ABC = (c, 001)(C)$.

A.6 SHORT CIRCUITS IN THE METAL LAYER

Many of the short circuits in the metal layer have the same effect as short circuits in the diffusion layer. In the following, we will consider only the shorts that were not covered previously. Figure A.10 shows the location of these shorts.

Short smp1: Cause nodes $\overline{B}$ and $\overline{C}$ to be shorted. Any input vector for which $B \neq C$ induces an intermediate voltage at both nodes. The possible test vectors are ABC =

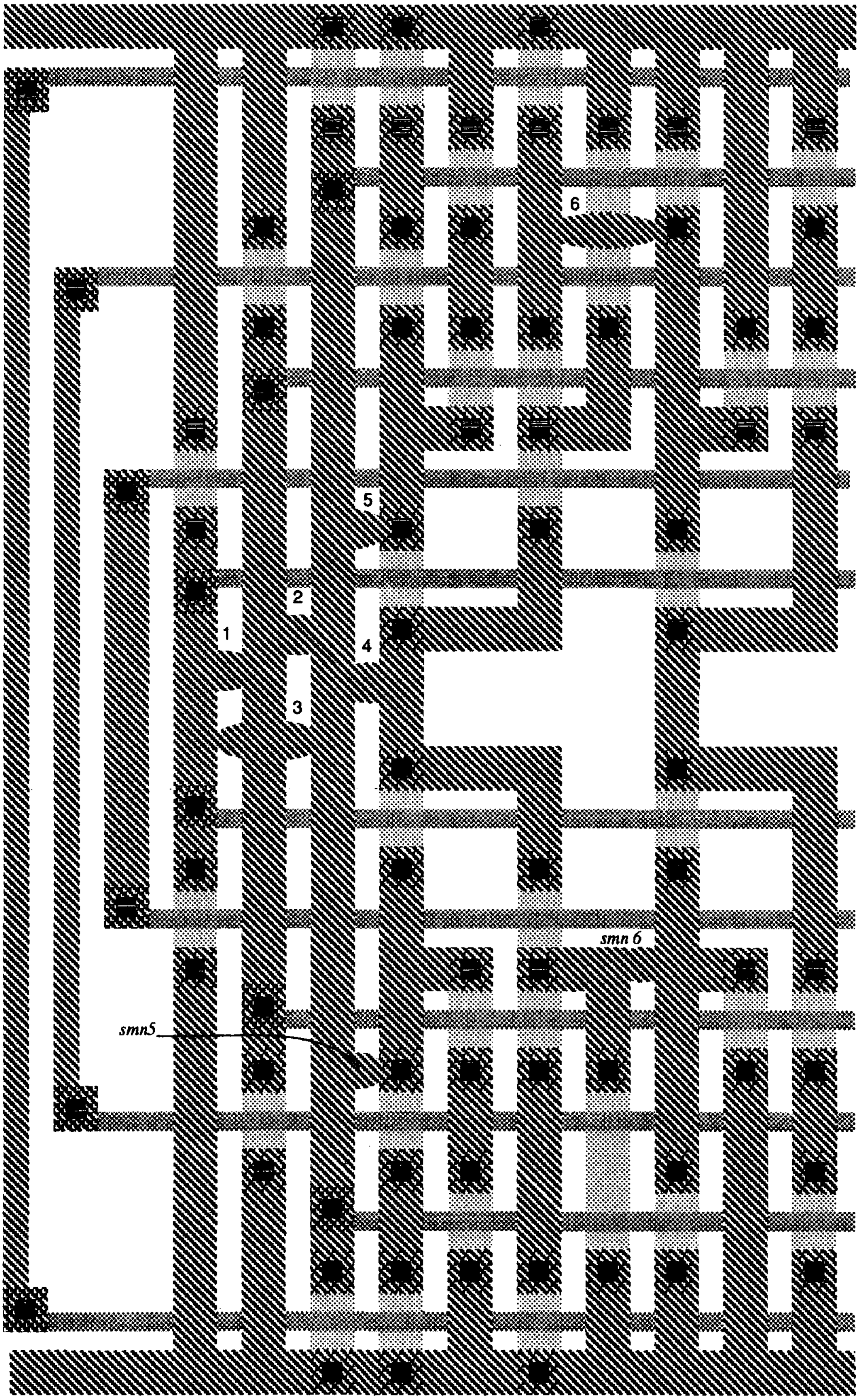


Figure A.10 Short circuits in the metal layer.

001(*SC*), *ABC* = 101(*SC*), *ABC* = 010(*SC*), *ABC* = 110(*SC*).

Short smp2: $\overline{A}$ and $\overline{B}$ shorted. *ABC* = 010(*SC*), *ABC* = 011(*SC*), *ABC* = 100(*SC*), *ABC* = 101(*SC*).

Short smp3: $\overline{A}$, $\overline{B}$ and $\overline{C}$ are shorted. *ABC* = 001(*SC*), *ABC* = 010(*SC*), *ABC* = 011(*SC*), *ABC* = 100(*SC*), *ABC* = 101(*SC*), *ABC* = 110(*SC*).

Short smp4: $\overline{A}$ and the sum output are shorted. *ABC* = 111(*S*), *ABC* = 100(*S*), *ABC* = 011(*S*) or *ABC* = 000(*S*).

Short smp5: $\overline{A}$ and S2 are shorted. The extra path in the sum circuit is {P1 P6}. Setting $A = 0$ and $C = 1$ will set the sum output to logic 1. *B* must be chosen so that the sum output is low in the fault-free circuit. Therefore, *ABC* = 011(*S*) produces an intermediate voltage at the sum output.

Short smp6: S4 and S13 are shorted. The extra paths in the sum circuit are {P14 P10 P13} and {P16 P10 P13}. The extra path in the carry circuit is {P9 P15}. *ABC* = 110(*S*) or *ABC* = 001(*C*).

The possible short circuits in the metal layer of the pull-down network are also shown in Fig. A.10.

Short smn5: Nodes S7 and $\overline{A}$ are shorted. Extra path {N1 N4}. *ABC* = 100(*S*).

Short smn6: S10 and S15 are shorted. Extra paths in the sum circuit {N15 N10 N9} and {N16 N10 N9}. Extra path in the carry circuit {N11 N14}. *ABC* = 001(*S*) or *ABC* = 110(*C*).

A.7 SHORTS BETWEEN METAL AND POLYSILICON

Only shorts not covered by the previously considered faults are treated. Figure A.11 shows the locations of these shorts for the pull-up network.

Short smp1: $\overline{A}$ and S4 are shorted. Undetectable.

Short smp2: $\overline{A}$ and *B* are shorted. *ABC* = 000(*SC*), *ABC* = 001(*SC*), *ABC* = 110(*SC*) or *ABC* = 111(*SC*).

Short smp3: *B* and S3 are shorted. *ABC* = 011(*S*), *ABC* = 100(*SC*) or *ABC* = 101(*SC*).

Short smp4: *B* and S4 are shorted. *ABC* = 000(*SC*), *ABC* = 001(*SC*) or *ABC* = 110(*S*).

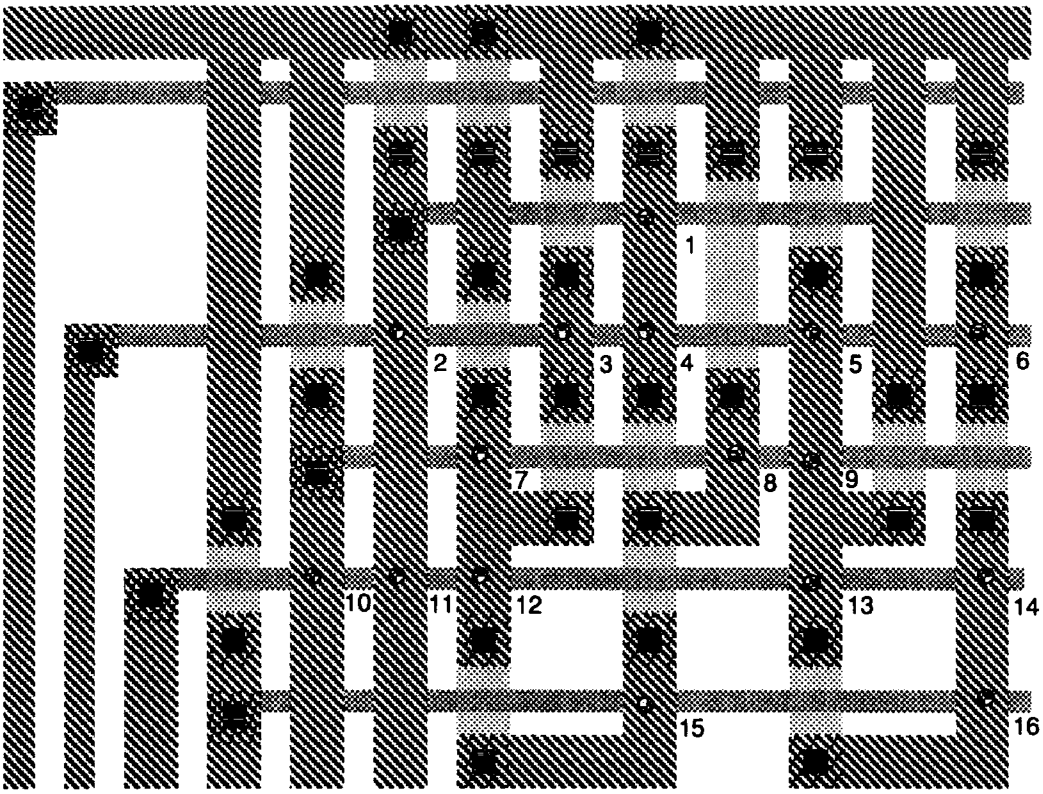


Figure A.11 Short circuits between metal and polysilicon layers in the pull-up network.

Short smp5: B and S13 shorted. $ABC = 101(C)$.

Short smp6: B and S14 shorted. $ABC = 010(C)$, $ABC = 100(SC)$ or $ABC = 101(SC)$.

Short smp7: $\overline{B}$ and S2 shorted. $ABC = 111(S)$ or $ABC = 101(S)$.

Short smp8: $\overline{B}$ and S6 shorted. $ABC = 100(S)$ or $ABC = 010(S)$.

Short smp9: $\overline{B}$ and S13 shorted. $ABC = 001(C)$ or $ABC = 011(C)$.

Short smp10: C and $\overline{B}$ shorted. $ABC = 000(SC)$, $ABC = 011(SC)$, $ABC = 100(SC)$ or $ABC = 111(SC)$.

Short smp11: C and $\overline{A}$ shorted. $ABC = 000(SC)$, $ABC = 010(SC)$, $ABC = 101(SC)$ or $ABC = 111(SC)$.

Short smp12: C and S2 shorted. $ABC = 000(SC)$, $ABC = 011(S)$, $ABC = 101(S)$ or $ABC = 110(SC)$.

Short smp13: C and S13 shorted. $ABC = 001(C)$, $ABC = 010(SC)$, $ABC = 100(SC)$ or $ABC = 110(SC)$.

Short smpl4: C and the carry output are shorted. $ABC = 001(C)$ or $ABC = 110(C)$.

Short smpl5: $\overline{C}$ and SUM shorted. $ABC = 000(S)$, $ABC = 001(S)$, $ABC = 110(S)$, or $ABC = 111(S)$.

Short smpl6: $\overline{C}$ and the carry output are shorted. $ABC = 000(C)$, $ABC = 010(C)$, $ABC = 011(C)$, $ABC = 100(C)$, $ABC = 101(C)$, or $ABC = 111(C)$.

Figure A.12 shows the locations of shorts between metal and polysilicon in the pull-down network.

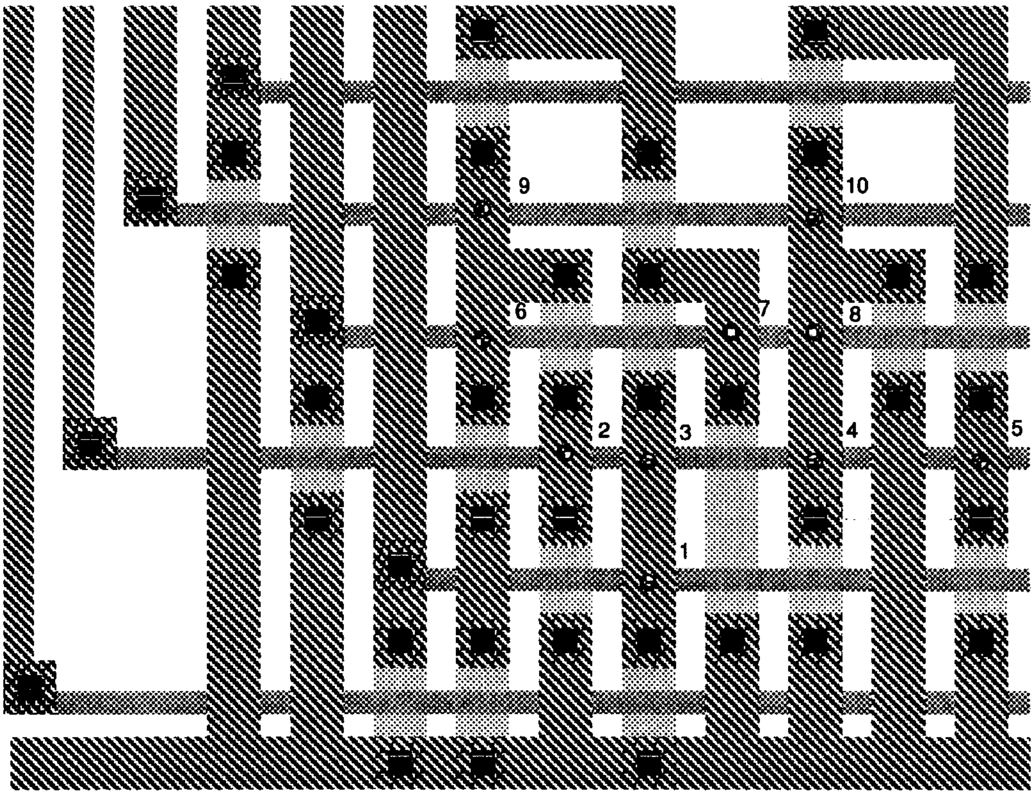


Figure A.12 Short circuits between metal and polysilicon in the pull-down network.

Short smn1: $\overline{A}$ and S10 shorted. Undetectable.

Short smn2: B and S9 shorted. $ABC = 010(SC)$, $ABC = 011(SC)$ or $ABC = 100(S)$.

Short smn3: B and S10 shorted. $ABC = 001(S)$, $ABC = 110(SC)$ or $ABC = 111(SC)$.

Short smn4: B and S15 shorted. $ABC = 010(C)$.

Short smn5: B and S16 shorted. $ABC = 010(SC)$, $ABC = 011(SC)$ or $ABC = 101(C)$.

Short smn6: $\overline{B}$ and S7 shorted. ABC = 010(S), or ABC = 000(S).

Short smn7: $\overline{B}$ and S11 shorted. ABC = 111(S) or ABC = 101(S).

Short smn8: $\overline{B}$ and S15 shorted. ABC = 100(C), ABC = 001(C), or ABC = 000(C).

Short smn9: C and S7 shorted. ABC = 001(SC), ABC = 010(S), ABC = 100(S) or ABC = 111(SC).

Short smn10: C and S15 shorted. ABC = 001(SC), ABC = 011(SC), ABC = 101(SC) or ABC = 110(C).

A.8 SHORTS BETWEEN METAL AND DIFFUSION

Non-existant in this layout.

A.9 OPEN CONTACTS

All possible open contact faults are covered by open circuits in the diffusion, metal and polysilicon layers.

A.10 SUMMARY

Table A.1 Breaks in the diffusion layer (pull-up).

Open	Effect	Test(s)
bdp1	$\overline{A}$ stuck-at 0	000(S), 011(S), 001(C), 010(C)
bdp3	$\overline{B}$ stuck-at 0	000(S), 101(S), 001(C), 100(C)
bdp4	$\overline{C}$ stuck-at 0	000(S), 110(S), 010(C), 100(C)
bdp5	P4 stuck-open	(d, 001)(S)
bdp6	P5 stuck-open	(d, 001)(S)
bdp7	P6 stuck-open	(d, 001)(S), (d, 111)(S)
bdp8	P7 stuck-open	(d, 111)(S)
bdp9	P8 stuck-open	(d, 111)(S)
bdp10	P9 stuck-open	(d, 010)(S)
bdp11	P10 stuck-open	(d, 010)(S)
bdp12	P11 stuck-open	(d, 100)(S)
bdp13	P12 stuck-open	(d, 010)(S)
bdp14	P13 stuck-open	(d, 010)(S), (d, 100)(S)
bdp15	P14 stuck-open	(d, 101)(C)
bdp16	P15 stuck-open	(d, 101)(C), (d, 011)(C)
bdp17	P16 stuck-open	(d, 011)(C)
bdp18	P17 stuck-open	(d, 110)(C)
bdp19	P18 stuck-open	(d, 110)(C)

Table A.2 Breaks in the diffusion layer (pull-down).

Open	Effect	Test(s)
bdn1	$\overline{A}$ stuck-at 1	100(S), 111(S), 101(C), 110(C)
bdn2	$\overline{B}$ stuck-at 1	010(S), 111(S), 011(C), 110(C)
bdn3	$\overline{C}$ stuck-at 1	001(S), 111(S), 011(C), 101(C)
bdn4	N6 stuck-open	(c, 110)(S)
bdn5	N5 stuck-open	(c, 110)(S)
bdn6	N4 stuck-open	(c, 110)(S), (c, 000)(S)
bdn7	N8 stuck-open	(c, 000)(S)
bdn8	N7 stuck-open	(c, 000)(S)
bdn9	N11 stuck-open	(c, 101)(S)
bdn10	N10 stuck-open	(c, 101)(S)
bdn11	N13 stuck-open	(c, 011)(S)
bdn12	N12 stuck-open	(c, 011)(S)
bdn13	N9 stuck-open	(c, 101)(S), (c, 011)(S)
bdn14	N15 stuck-open	(c, 010)(C)
bdn15	N14 stuck-open	(c, 010)(C), (c, 100)(C)
bdn16	N16 stuck-open	(c, 100)(C)
bdn17	N18 stuck-open	(c, 001)(C)
bdn18	N17 stuck-open	(c, 001)(C)

Table A.3 Shorts in the p-type diffusion layer.

Open	Effect	Test(s)
sdp3	$\overline{A}$ and S1 shorted	Undetectable
sdp4	S1 stuck-at 1	101(S)
sdp5	S1 and S3 shorted	101(S), 011(S)
sdp6	S2 and S3 shorted	101(S)
sdp7	S4 stuck-at 1	110(S)
sdp8	S4 and S3 shorted	110(S), 011(S)
sdp9	S2 and S6 shorted	d(S)
sdp10	S2 and SUM shorted	000(S), 110(S)
sdp11	S4 stuck-at 1	110(S)
sdp12	S3 and S5 shorted	Undetectable
sdp13	S6 and S4 shorted	000(S)
sdp15	S5 and S13 shorted	Undetectable
sdp16	S6 stuck-at 1	000(S), 110(S)
sdp17	S6 and S13 shorted	110(S)
sdp18	S13 and SUM shorted	011(SC), 101(S), 110(S)
sdp20	S13 and S14 shorted	010(C)
sdp21	S14 stuck-at 1	010(C)
sdp22	S13 and CARRY shorted	001(C), 100(C), 010(C)
sdp23	SUM and CARRY shorted	001(SC), 010(SC), 100(SC), 011(SC), 101(SC), 110(SC)

Table A.4 Shorts in the n-type diffusion layer.

Open	Effect	Test(s)
sdn1	$\overline{A}$ and S8 shorted	Undetectable
sdn2	S8 stuck-at 0	010(S)
sdn3	S8 and S9 shorted	100(S), 010(S)
sdn4	S7 and S9 shorted	010(S)
sdn5	S10 stuck-at 0	001(S)
sdn6	S12 and S10 shorted	111(S), 001(S)
sdn7	S10 and S9 shorted	001(S), 100(S)
sdn8	S11 and S7 shorted	c(S)
sdn9	SUM and S7 shorted	111(S), 001(S)
sdn10	S10 stuck-at 0	001(S)
sdn11	S11 and S10 shorted	111(S)
sdn13	S12 and S15 shorted	Undetectable
sdn14	S11 stuck-at 0	001(S), 111(S)
sdn15	S15 and S11 shorted	001(S)
sdn16	S15 and SUM shorted	001(S), 010(S), 100(S), 110(C)
sdn17	SUM and CARRY shorted	001(SC), 010(SC), 011(SC), 100(SC), 101(SC), 110(SC)
sdn19	S16 and S15 shorted	101(C)
sdn20	S16 stuck-at 0	101(C)
sdn21	CARRY and S15 shorted	001(C), 010(C), 100(C)

Table A.5 Open circuits in the polysilicon layer (stuck-closed case, pull-up).

Open	Effect	Test(s)
bpp1	P1, P4 and P9 stuck-closed	100(SC), 101(SC), 110(SC), 111(SC)
bpp2	P4 and P9 stuck-closed	101(S), 110(S)
bpp3	P9 stuck-closed	110(S)
bpp4	P7, P11, P14 and P17 stuck-closed	000(S), 011(C), 010(C), 011(S)
bpp5	P11, P14 and P17 stuck-closed	000(S), 001(C), 010(C)
bpp6	P14 and P17 stuck-closed	001(C), 010(C)
bpp7	P17 stuck-closed	010(C)
bpp8	P2, P5 and P12 stuck-closed	010(SC), 011(SC), 110(SC), 111(SC)
bpp9	P5 and P12 stuck-closed	011(S), 110(S)
bpp10	P12 stuck-closed	110(S)
bpp11	P8, P10, P16 and P18 stuck-closed	000(S), 001(C), 100(C), 101(S)
bpp12	P10, P16 and P18 stuck-closed	000(S), 001(C), 100(C)
bpp13	P16 and P18 stuck-closed	001(C), 100(C)
bpp14	P18 stuck-closed	100(C)
bpp15	P3 and P13 stuck-closed	001(SC), 011(SC), 101(SC), 111(SC)
bpp16	P13 stuck-closed	011(S), 101(S)
bpp17	P6 and P15 stuck-closed	000(S), 010(C), 100(C), 110(S)
bpp18	P15 stuck-closed	010(C), 100(C)

Table A.6 Open circuits in the polysilicon layer (stuck-closed case, pull-down).

Open	Effect	Test(s)
bpn1	N1, N6 and N11 stuck-closed	000(SC), 001(SC), 010(SC), 011(SC)
bpn2	N6 and N11 stuck-closed	001(S), 010(S)
bpn3	N11 stuck-closed	001(S)
bpn4	N8, N13, N15 and N18 stuck-closed	100(S), 101(C), 110(C), 111(S)
bpn5	N13, N15 and N18 stuck-closed	101(C), 110(C), 111(C)
bpn6	N15 and N18 stuck-closed	101(C), 110(C)
bpn7	N18 stuck-closed	101(C)
bpn8	N2, N5 and N12 stuck-closed	000(SC), 001(SC), 100(SC), 101(SC)
bpn9	N5 and N12 stuck-closed	001(S), 100(S)
bpn10	N12 stuck-closed	001(S)
bpn11	N7, N10, N16 and N17 stuck-closed	010(S), 011(C), 110(C), 111(S)
bpn12	N10, N16 and N17 stuck-closed	011(C), 110(C), 111(S)
bpn13	N16 and N17 stuck-closed	011(C), 110(C)
bpn14	N17 stuck-closed	011(C)
bpn15	N3 and N9 stuck-closed	000(SC), 010(SC), 100(SC), 110(SC)
bpn16	N9 stuck-closed	010(S), 100(S)
bpn17	N4 and N14 stuck-closed	001(S), 011(C), 101(C), 111(S)
bpn18	N14 stuck-closed	011(C), 101(C)

Table A.7 Open circuits in the polysilicon layer (stuck-open case, pull-up).

Open	Effect	Test(s)
bpp1	P1, P4 and P9 stuck-open, $\overline{A}$ stuck-at 0	000(S), 001(C), 010(C), 011(S)
bpp2	P4 and P9 stuck-open	(d, 001)(S), (d, 010)(S)
bpp3	P9 stuck-open	(d, 010)(S)
bpp4	P7, P11, P14 and P17 stuck-open	(d, 100)(S), (d, 101)(C), (d, 110)(C), (d, 111)(S)
bpp5	P11, P14 and P17 stuck-open	(d, 100)(S), (d, 101)(C), (d, 110)(C)
bpp6	P14 and P17 stuck-open	(d, 101)(C), (d, 110)(C)
bpp7	P17 stuck-open	(d, 110)(C)
bpp8	P2, P5 and P12 stuck-open, $\overline{B}$ stuck-at 0	000(S), 001(C), 100(C), 101(S)
bpp9	P5 and P12 stuck-open	(d, 001)(S), (d, 100)(S)
bpp10	P12 stuck-open	(d, 100)(S)
bpp11	P8, P10, P16 and P18 stuck-open	(d, 010)(S), (d, 011)(C), (d, 110)(C), (d, 111)(S)
bpp12	P10, P16 and P18 stuck-open	(d, 010)(S), (d, 011)(C), (d, 110)(C)
bpp13	P16 and P18 stuck-open	(d, 011)(C), (d, 110)(C)
bpp14	P18 stuck-open	(d, 110)(C)
bpp15	P3 and P13 stuck-open, $\overline{C}$ stuck-at 0	000(S), 010(C), 100(C), 110(S)
bpp16	P13 stuck-open	(d, 010)(S), (d, 100)(S)
bpp17	P6 and P15 stuck-open	(d, 001)(S), (d, 011)(C), (d, 101)(C), (d, 111)(S)
bpp18	P15 stuck-open	(d, 011)(C), (d, 101)(C)

Table A.8 Open circuits in the polysilicon layer (stuck-open case, pull-down).

Open	Effect	Test(s)
bpn1	N1, N6 and N11 stuck-open, $\overline{A}$ stuck-at 1	100(S), 101(C), 110(C), 111(S)
bpn2	N6 and N11 stuck-open	(c, 101)(S), (c, 110)(S)
bpn3	N11 stuck-open	(c, 101)(S)
bpn4	N8, N13, N15 and N18 stuck-open	(c, 000)(S), (c, 001)(C), (c, 010)(C), (c, 011)(S)
bpn5	N13, N15 and N18 stuck-open	(c, 001)(C), (c, 010)(C), (c, 011)(S)
bpn6	N15 and N18 stuck-open	(c, 001)(C), (c, 010)(C)
bpn7	N18 stuck-open	(c, 001)(C)
bpn8	N2, N5 and N12 stuck-open, $\overline{B}$ stuck-at 1	010(S), 011(C), 110(C), 111(S)
bpn9	N5 and N12 stuck-open	(c, 011)(S), (c, 110)(S)
bpn10	N12 stuck-open	(c, 011)(S)
bpn11	N7, N10, N16 and N17 stuck-open	(c, 000)(S), (c, 001)(C), (c, 100)(C), (c, 101)(S)
bpn12	N10, N16 and N17 stuck-open	(c, 001)(C), (c, 100)(C), (c, 101)(S)
bpn13	N16 and N17 stuck-open	(c, 001)(C), (c, 100)(C)
bpn14	N17 stuck-open	(c, 001)(C)
bpn15	N3 and N9 stuck-open, $\overline{C}$ stuck-at 1	001(S), 011(C), 101(C), 111(S)
bpn16	N9 stuck-open	(c, 011)(S), (c, 101)(S)
bpn17	N4 and N14 stuck-open	(c, 000)(S), (c, 010)(C), (c, 100)(C), (c, 110)(S)
bpn18	N14 stuck-open	(c, 010)(C), (c, 100)(C)

Table A.9 Short circuits in the polysilicon layer.

Open	Effect	Test(s)
spp1	A and B shorted	010(SC), 011(SC), 100(SC), 101(SC)
spp2	B and C shorted	001(SC), 101(SC), 010(SC), 110(SC)
spp3	B and $\overline{A}$ shorted	Any
spp4	$\overline{A}$ and B shorted	110(SC), 111(SC), 001(SC), 000(SC)
spp5	B and $\overline{B}$ shorted	Any
spp6	$\overline{B}$ and C shorted	011(SC), 111(SC), 100(SC), 000(SC)
spp7	C and $\overline{C}$ shorted	Any

Table A.10 Open circuits in the metal layer (pull-up).

Open	Effect	Test(s)
bmp1	Disconnects SUM from pull-up	c(S)
bmp2	Disconnects CARRY from pull-up	c(C)
bmp3	Disconnects VDD from the circuit	A true and a false vertex
bmp4	$\overline{A}$, $\overline{B}$ stuck-at 0 CARRY floating	Same as *bmp3*
bmp5	$\overline{A}$ stuck-at 0, SUM and CARRY floating	(100, c)(S), (101, c)(S), (100, c)(C), (000, c)(C)
bmp6	SUM and CARRY stuck-at 0	c(S), c(C)
bmp7	CARRY stuck-at 0	c(C)
bmp8-10	CARRY stuck-at 0	c(C)
bmp11	Part of pull-up of carry disconnected	(d, 011)(C), (d, 110)(C)
bmp12	Part of pull-up of carry disconnected	(d, 110)(C)

Table A.11 Open circuits in the metal layer (pull-down).

Open	Effect	Test(s)
bmn1	Disconnects SUM from pull-down	d(S)
bmn2	Disconnects CARRY from pull-down	d(C)
bmn3	Disconnects GND from the circuit	A true and a false vertex
bmn4	$\overline{A}$, $\overline{B}$ stuck-at 1 CARRY floating	Same as *bmn3*
bmn5	$\overline{A}$ stuck-at 0, SUM and CARRY floating	(001, d)(S), (010, d)(S), (111, d)(C), (011, d)(C)
bmn6	SUM and CARRY stuck-at 1	d(S), d(C)
bmn7	CARRY stuck-at 1	d(C)
bmn8-10	CARRY stuck-at 1	d(C)
bmn11	Part of pull-down of carry disconnected	(c, 100)(C), (c, 001)(C)
bmn12	Part of pull-down of carry disconnected	(c, 001)(C)

Table A.12 Shorts between metal tracks.

Open	Effect	Test(s)
smp1	$\overline{B}$ and $\overline{C}$ shorted	001(SC), 101(SC), 010(SC), 110(SC)
smp2	$\overline{A}$ and $\overline{B}$ shorted	010(SC), 011(SC), 100(SC), 101(SC)
smp3	$\overline{A}$, $\overline{B}$ and $\overline{C}$ shorted	001(SC), 010(SC), 011(SC), 100(SC), 101(SC), 110(SC)
smp4	$\overline{A}$ and SUM shorted	111(S), 100(S), 011(S), 000(S)
smp5	$\overline{A}$ and S2 shorted	011(S)
smp6	S4 and S13 shorted	110(S), 001(C)
smn5	S7 and $\overline{A}$ shorted	100(S)
smn6	S10 and S15 shorted	001(S),110(C)

Table A.13 Shorts between metal and polysilicon layers (pull-up).

Open	Effect	Test(s)
smp1	$\overline{A}$ and S4 shorted	Undetectable
smp2	$\overline{A}$ and B shorted	000(SC), 001(SC), 110(SC), 111(SC)
smp3	B and S3 shorted	011(S), 100(SC), 101(SC)
smp4	B and S4 shorted	000(SC), 001(SC), 110(SC)
smp5	B and S3 shorted	101(C)
smp6	B and S14 shorted	010(C), 100(SC), 101(SC)
smp7	$\overline{B}$ and S2 shorted	111(S), 101(S)
smp8	$\overline{B}$ and S6 shorted	100(S), 010(S)
smp9	$\overline{B}$ and S13 shorted	001(C), 011(C)
smp10	C and $\overline{B}$ shorted	000(SC), 011(SC), 100(SC), 111(SC)
smp11	C and $\overline{A}$ shorted	000(SC), 010(SC), 101(SC), 111(SC)
smp12	C and S2 shorted	000(SC), 011(S), 101(S), 110(SC)
smp13	C and S13 shorted	001(C), 010(SC), 100(SC), 110(SC)
smp14	C and $CARRY$ shorted	001(C), 110(C)
smp15	$\overline{C}$ and SUM shorted	000(S), 001(S), 110(S), 111(S)
smp16	$\overline{C}$ and $CARRY$ shorted	000(C), 010(C), 011(C), 100(C), 101(C), 111(C)

Table A.14 Shorts between metal and polysilicon layers (pull-down).

Open	Effect	Test(s)
smn1	$\overline{A}$ and S10 shorted	Undetectable
smn2	B and S9 shorted	010(SC), 011(SC), 100(SC)
smn3	B and S10 shorted	001(S), 100(SC), 111(SC)
smn4	B and S15 shorted	010(C)
smn5	B and S11 shorted	010(SC), 011(SC), 101(C)
smn6	$\overline{B}$ and S7 shorted	010(S), 000(S)
smn7	$\overline{B}$ and S11 shorted	111(S), 101(S)
smn8	$\overline{B}$ and S15 shorted	100(C)¡ 001(C), 000(C)
smn9	C and S7 shorted	001(SC), 010(S), 100(S), 111(SC)
smn10	C and S15 shorted	001(SC), 011(SC), 101(SC), 110(SC)

Appendix B
Yield and Reliability Calculations

B.1 GENERAL CASE

The non-redundant chip of area A_0 is assumed to consist of n units. Each unit i has an area A_i and it is replicated m_i times. The area of the test/reconfiguration logic is S_i. Let

P_{A_i}: Probability of having no defects in a unit of area A_i.

P_{S_i}: Probability of having no defects in the reconfiguration logic of area S_i.

P_i: Probability that the redundant unit consisting of m_i units and their reconfiguration logic is working.

B.1.1 Yield

Case 1

The averages of P_{A_i} and P_{S_i} are given by

$$\overline{P_{A_i}} = \int_0^{+\infty} e^{-A_iD} f(D)dD = \left(\frac{1-e^{-A_iD_0}}{A_iD_0}\right)^2 \text{ and } \overline{P_{S_i}} = \left(\frac{1-e^{-S_iD_0}}{S_iD_0}\right)^2$$

and the probability that redundant unit i is working can be expressed as

$$P_i' = \overline{P_{S_i}} \sum_{j=1}^{m_i} \binom{m_i}{j} \overline{P_{A_i}}^j (1-\overline{P_{A_i}})^{m_i-j} = \overline{P_{S_i}}(1-(1-\overline{P_{A_i}})^{m_i})$$

and the yield is

$$Y_{r_1} = \prod_{i=1}^{n} P_i'$$

Case 2

The probability that redundant unit i is working is

$$P_i = P_{S_i} \sum_{j=1}^{m_i} \binom{m_i}{j} P_{A_i}^j (1-P_{A_i})^{m_i-j} = P_{S_i} - P_{S_i}(1-P_{A_i})^{m_i}$$

with $P_{A_i} = e^{-A_iD}, P_{S_i} = e^{-S_iD}$. The average of P_i is

$$\overline{P_i} = \int_0^{+\infty} P_i f(D)dD$$

$$\overline{P_i} = \int_0^{+\infty} e^{-S_iD} f(D)dD - \int_0^{+\infty} P_{S_i}(1-P_{A_i})^{m_i} f(D)dD$$

$$\overline{P_i} = \Big(\frac{1 - e^{S_i D0}}{S_i D_0}\Big)^2 - \int_0^{+\infty} P_{S_i} \sum_{j=0}^{m_i} \binom{m_i}{j} (-1)^j P_{A_i}^j f(D) dD$$

$$\overline{P_i} = \Big(\frac{1 - e^{S_i D0}}{S_i D_0}\Big)^2 - \int_0^{+\infty} \sum_{j=0}^{m_i} \binom{m_i}{j} (-1)^j e^{-(S_i + jA_i)D} f(D) dD$$

$$\overline{P_i} = \Big(\frac{1 - e^{S_i D0}}{S_i D_0}\Big)^2 - \sum_{j=0}^{m_i} \binom{m_i}{j} (-1)^j \Big(\frac{1 - e^{-(S_i + jA_i)D_0}}{(S_i + jA_i)D_0}\Big)^2$$

and the second yield expression is

$$Y_{r_2} = \prod_{i=1}^{n} \overline{P_i}$$

Case 3

The probability of having a working chip is

$$P_r = \prod_{i=1}^{n} \Big(P_{S_i} \sum_{j=1}^{m_i} \binom{m_i}{j} P_{A_i}^j (1 - P_{A_i})^{m_i - j} \Big)$$

$$P_r = \prod_{i=1}^{n} \Big(P_{S_i} \sum_{j=1}^{m_i} \binom{m_i}{j} P_{A_i}^j \sum_{k=0}^{m_i - j} \binom{m_i - j}{k} (-1)^k P_{A_i}^k \Big)$$

$$P_r = \prod_{i=1}^{n} \Big(\sum_{j=1}^{m_i} \sum_{k=0}^{m_i - j} (-1)^k \binom{m_i}{j} \binom{m_i - j}{k} P_{A_i}^{j+k} P_{S_i} \Big)$$

$$P_r = \prod_{i=1}^{n} \Big(\sum_{j=1}^{m_i} \sum_{k=0}^{m_i - j} (-1)^k \binom{m_i}{j} \binom{m_i - j}{k} e^{-(S_i + (j+k)A_i)D} \Big)$$

$$P_r = \sum_{j(1)=1}^{m(1)} \sum_{k(1)=0}^{m(1)-j(1)} \sum_{j(2)=1}^{m(2)} \sum_{k(2)=0}^{m(2)-j(2)} \cdots \sum_{j(n)=1}^{m(n)} \sum_{k(n)=0}^{m(n)-j(n)} (-1)^{\sum_{i=1}^{n} k(i)}$$

$$\Big(\prod_{i=1}^{n} \binom{m(i)}{j(i)} \binom{m(i) - j(i)}{k(i)} \Big) e^{-(\sum_{i=1}^{n} S_i + (j(i)+k(i))A_i)D}$$

and the third expression for the yield is the average of P_r which is $Y_{r_3} = \int_0^{+\infty} P_r f(D) dD$

$$Y_{r_3} = \sum_{j(1)=1}^{m(1)} \sum_{k(1)=0}^{m(1)-j(1)} \sum_{j(2)=1}^{m(2)} \sum_{k(2)=0}^{m(2)-j(2)} \cdots \sum_{j(n)=1}^{m(n)} \sum_{k(n)=0}^{m(n)-j(n)} (-1)^{\sum_{i=1}^{n} k(i)}$$

$$(\prod_{i=1}^{n} \binom{m(i)}{j(i)} \binom{m(i)-j(i)}{k(i)})(\frac{1 - e^{-(\sum_{i=1}^{n} S_i + (j(i)+k(i))A_i)D_0}}{(\sum_{i=1}^{n} S_i + (j(i) + k(i))A_i)D_0})^2$$

B.1.2 Reliability

Case 1

$\overline{P_{A_i}} = \int e^{-A_i D} f(D) dD = (\frac{1-e^{-A_i D_0}}{A_i D_0})^2$, and $\overline{P_{S_i}} = (\frac{1-e^{-S_i D_0}}{S_i D_0})^2$. The probability that redundant unit i is working is

$$R_i' = \sum_{k=0}^{m_i-1} \overline{P_{S_i}} \binom{m_i}{k} (1 - \overline{P_{A_i}})^k \overline{P_{A_i}}^{(m_i-k)} R_{S_i} \sum_{j=1}^{m_i-k} \binom{m_i - k}{j} R_{A_i}^j (1 - R_{A_i})^{m_i-k-j}$$

$$R_i' = \sum_{k=0}^{m_i-1} \overline{P_{S_i}} \binom{m_i}{k} (1 - \overline{P_{A_i}})^k \overline{P_{A_i}}^{(m_i-k)} R_{S_i} (1 - (1 - R_{A_i})^{m_i-k})$$

$$R_i' = \overline{P_{S_i}} R_{S_i} (1 - (1 - \overline{P_{A_i}} R_{A_i})^{m_i})$$

and the probability that the chip is working is

$$R_{r_1} = \prod_{i=1}^{n} \left(\overline{P_{S_i}} R_{S_i} (1 - (1 - \overline{P_{A_i}} R_{A_i})^{m_i}) \right)$$

Case 2

$P_{A_i} = e^{-A_i D}$, $P_{S_i} = e^{-S_i D}$ and the probability that redundant unit i is working is

$$R_i = \sum_{k=0}^{m_i-1} P_{S_i} \binom{m_i}{k} (1 - P_{A_i})^k P_{A_i}^{m_i-k} R_{S_i} (1 - (1 - R_{A_i})^{m_i-k})$$

$$R_i = P_{S_i} R_{S_i} (1 - (1 - P_{A_i} R_{A_i})^{m_i}) = P_{S_i} R_{S_i} - P_{S_i} R_{S_i} (1 - P_{A_i} R_{A_i})^{m_i}$$

The average of R_i is $\overline{R_i} = \int_0^{+\infty} R_i f(D) dD$

$$\overline{R_i} = \int_0^{+\infty} P_{S_i} R_{S_i} f(D) dD - \int_0^{+\infty} P_{S_i} R_{S_i} \sum_{j=0}^{m_i} \binom{m_i}{j} (-1)^j P_{A_i}^j R_{A_i}^j f(D) dD$$

$$\overline{R_i} = R_{S_i} (\frac{1 - e^{-S_i D_0}}{S_i D_0})^2 - \sum_{j=0}^{m_i} \binom{m_i}{j} (-1)^j R_{S_i} R_{A_i}^j (\frac{1 - e^{(S_i + j A_i) D_0}}{(S_i + j A_i) D_0})^2$$

and the probability that the chip is working is

$$R_{r_2} = \prod_{i=1}^{n} \overline{R_i}$$

Case 3

$P_{A_i} = e^{-A_i D}$, $P_{S_i} = e^{-S_i D}$ and the probability that redundant unit i is working is

$$R_i = \sum_{k=0}^{m_i-1} P_{S_i} \binom{m_i}{k} (1 - P_{A_i})^k P_{A_i}^{m_i-k} R_{S_i}(1 - (1 - R_{A_i})^{m_i-k})$$

The probability that the chip is working is

$$R_r = \prod_{i=1}^{n} R_i = \prod_{i=1}^{n} \Big(\sum_{k=0}^{m_i-1} P_{S_i} \binom{m_i}{k} (1 - P_{A_i})^k P_{A_i}^{m_i-k} R_{S_i}(1 - (1 - R_{A_i})^{m_i-k}) \Big)$$

$$R_r = \prod_{i=1}^{n} \Big(\sum_{k=0}^{m_i-1} \sum_{j=0}^{k} \binom{m_i}{k} \binom{k}{j} (-1)^j P_{S_i} P_{A_i}^{m_i-k+j} R_{S_i}(1 - (1 - R_{A_i})^{m_i-k}) \Big)$$

$$R_r = \prod_{i=1}^{n} \Big(\sum_{k=0}^{m_i-1} \sum_{j=0}^{k} \binom{m_i}{k} \binom{k}{j} (-1)^j e^{-(S_i+(m_i-k+j)A_i)D} R_{S_i}(1 - (1 - R_{A_i})^{m_i-k}) \Big)$$

$$R_r = \sum_{k(1)=0}^{m(1)-1} \sum_{j(1)=0}^{k(1)} \sum_{k(2)=0}^{m(2)-1} \sum_{j(2)=0}^{k(2)} \cdots \sum_{k(n)=0}^{m(n)-1} \sum_{j(n)=0}^{k(n)}$$

$$\binom{m(1)}{k(1)} \binom{k(1)}{j(1)} \binom{m(2)}{k(2)} \binom{k(2)}{j(2)} \cdots \binom{m(n)}{k(n)} \binom{k(n)}{j(n)} (-1)^{\sum_{i=1}^{n} j(i)}$$

$$\Big(\prod_{i=1}^{n} R_{S_i}(1 - (1 - R_{A_i})^{m(i)-k(i)}) \Big) e^{-(\sum_{i=1}^{n} S_i+(m(i)-k(i)+j(i))A_i)D}$$

The average of R_r is $R_{r_3} = \int R_r f(D)dD$

$$R_{r_3} = \sum_{k(1)=0}^{m(1)-1} \sum_{j(1)=0}^{k(1)} \sum_{k(2)=0}^{m(2)-1} \sum_{j(2)=0}^{k(2)} \cdots \sum_{k(n)=0}^{m(n)-1} \sum_{j(n)=0}^{k(n)} \Big(\prod_{i=1}^{n} \binom{m(i)}{k(i)} \binom{k(i)}{j(i)} \Big) (-1)^{\sum_{i=1}^{n} j(i)}$$

$$\Big(\prod_{i=1}^{n} R_{S_i}(1 - (1 - R_{A_i})^{m(i)-k(i)}) \Big) \Big(\frac{1 - e^{-(\sum_{i=1}^{n} S_i+(m(i)-k(i)+j(i))A_i)D_0}}{(\sum_{i=1}^{n} S_i + (m(i) - k(i) + j(i))A_i)D_0} \Big)^2$$

B.2 SPECIAL CASE

All sections are supposed to have the same size and are replicated the same number of time: $A_i = A_0/n = A, S_i = S, m_i = m,$ for $i = 1, 2, \ldots n.$

B.2.1 Yield

Case 1

$$\overline{P_A} = \int e^{-AD} f(D)dD = (\tfrac{1-e^{-AD_0}}{AD_0})^2; \quad \overline{P_S} = (\tfrac{1-e^{-SD_0}}{SD_0})^2.$$

$$Y_{r_1} = \Big(\overline{P_S}\sum_{i=1}^{m}\binom{m}{i}\overline{P_A}^{i}(1-\overline{P_A})^{m-i}\Big)^n = \overline{P_S}^{n}(1-(1-\overline{P_A})^m)^n$$

Case 2

$$P_A = e^{-AD}; \quad P_S = e^{-SD}.$$

$$P = P_S\sum_{i=1}^{m}\binom{m}{i}P_A^i(1-P_A)^{m-i} = P_S(1-(1-P_A)^m) = P_S - P_S(1-P_A)^m$$

and the average of P is $\overline{P} = \int Pf(D)dD$ which is

$$\overline{P} = \int_0^{+\infty} P_S f(D)dD - \int_0^{+\infty} P_S(1-P_A)^m f(D)dD$$

$$\overline{P} = \Big(\frac{1-e^{-SD_0}}{SD_0}\Big)^2 - \int_0^{+\infty} P_S\sum_{i=0}^{m}\binom{m}{i}(-1)^i P_A^i f(D)dD$$

$$\overline{P} = \Big(\frac{1-e^{-SD_0}}{SD_0}\Big)^2 - \sum_{i=0}^{m}\binom{m}{i}(-1)^i\Big(\frac{1-e^{-(S+iA)D_0}}{(S+iA)D_0}\Big)^2$$

and $Y_{r_2} = \overline{P}^n$.

Case 3

$$P_r = \Big(P_S\sum_{i=1}^{m}\binom{m}{i}P_A^i(1-P_A)^{m-i})\Big)^n$$

$$P_r = P_S^n(1-(1-P_A)^m)^n$$

$$P_r = P_S^n\sum_{i=0}^{n}\binom{n}{i}(-1)^i(1-P_A)^{mi}$$

$$P_r = P_S^n\sum_{i=0}^{n}\binom{n}{i}(-1)^i\sum_{j=0}^{mi}\binom{mi}{j}(-1)^j P_A^j$$

$$P_r = P_S^n\sum_{i=0}^{n}\sum_{j=0}^{mi}\binom{n}{i}\binom{mi}{j}(-1)^{i+j}P_A^j$$

and $Y_{r_3} = \overline{P_r} = \int P_r f(D)dD$ which gives

$$Y_{r_3} = \sum_{i=0}^{n}\sum_{j=0}^{mi}\binom{n}{i}\binom{mi}{j}(-1)^{i+j}\int_0^{+\infty} e^{-(nS+jA)D} f(D)dD$$

$$Y_{r_3} = \sum_{i=0}^{n}\sum_{j=0}^{mi}\binom{n}{i}\binom{mi}{j}(-1)^{i+j}\Big(\frac{1-e^{-(nS+jA)D_0}}{(nS+jA)D_0}\Big)^2$$

B.2.2 Reliability

Case 1

$\overline{P_A} = (\frac{1-e^{-AD_0}}{AD_0})^2; \overline{P_S} = (\frac{1-e^{-SD_0}}{SD_0})^2; R_A = e^{-\lambda_A t}; R_S = e^{-\lambda_s t}.$

$$R' = \sum_{k=0}^{m-1} \overline{P_S}\binom{m}{k}(1-\overline{P_A})^k\overline{P_A}^{m-k}R_S(1-(1-R_A)^{m-k})$$

$R' = \overline{P_S}R_S(1-(1-\overline{P_A}R_A)^m)$ and $R_{r_1} = R'^n = \overline{P_S}^n R_S^n(1-(1-\overline{P_A}R_A)^m)^n$.

Case 2

$P_A = e^{-AD}; P_S = e^{-SD}.$

$$R = \sum_{k=0}^{m-1} P_S\binom{m}{k}(1-P_A)^k P_A^{m-k}R_S(1-(1-R_A)^{m-k})$$

$R = P_S R_S(1-(1-P_A R_A)^m)$ and the average of R is $\overline{R} = \int r f(D)dD$ which is

$$\overline{R} = \int_0^{+\infty} P_S R_S f(D)dD - \int_0^{+\infty} P_S R_S(1-P_A R_A)^m f(D)dD$$

$$\overline{R} = \int_0^{+\infty} R_S e^{-SD} f(D)dD - \int_0^{+\infty} P_S R_S \sum_{i=0}^{m}\binom{m}{i}(-1)^i P_A^i R_A^i f(D)dD$$

$$\overline{R} = R_S(\frac{1-e^{-SD_0}}{SD_0})^2 - \sum_{i=0}^{m}\binom{m}{i}(-1)^i R_S R_A^i\Big(\frac{1-e^{-(S+iA)D_0}}{(S+iA)D_0}\Big)^2$$

and R_{r_2} is given by $R_{r_2} = \overline{R}^n$.

Case 3

$$R = \sum_{k=0}^{m-1} P_S\binom{m}{k}(1-P_A)^k P_A^{m-k}R_S(1-(1-R_A)^{m-k})$$

$$R = P_S R_S(1 - (1 - P_A R_A)^m)$$

$$R_r = R^n = P_S^n R_S^n(1 - (1 - P_A R_A)^m)^n$$

$$R_r = P_S^n R_S^n \sum_{i=0}^{n} \binom{n}{i}(-1)^i(1 - P_A R_A)^{mi}$$

$$R_r = P_S^n R_S^n \sum_{i=0}^{n} \binom{n}{i}(-1)^i \sum_{j=0}^{mi} \binom{mi}{i}(-1)^j P_A^j R_A^j$$

$$R_r = \sum_{i=0}^{n} \sum_{j=0}^{mi} \binom{n}{i} \binom{mi}{j}(-1)^{i+j} R_S^n R_A^j e^{-(nS+jA)D}$$

$R_{r_3} = \overline{R_r} = \int R_r f(D)dD$ which is

$$R_{r_3} = \sum_{i=0}^{n} \sum_{j=0}^{mi} \binom{n}{i} \binom{mi}{j}(-1)^{i+j} R_S^n R_A^{mij} \left(\frac{1 - e^{-(nS+jA)D_0}}{(nS + jA)D_0}\right)^2$$

B.3 FAILURE RATE OF A SECTION OF A CHIP

In the MIL-HDBK-217 model of chip failure [219], the reliability function is assumed to be $e^{-\lambda t}$ where λ is the failure rate and takes the form:

$$\lambda = \pi_L \pi_Q (C_1 \pi_T \pi_V \pi_{PT} + (C_2 + C_3)\pi_E) \tag{3}$$

where

π_L: learning factor=10 for new devices or a new process, otherwise=1.0.

π_Q: quality factor, function of device screening. It is equal to 35 for commercial parts.

π_T: temperature acceleration factor, function of device technology, package, case temperature and power dissipation. It is assumed that $\pi_T = 5.0$.

π_V: voltage stress factor=1 for CMOS at $V_{DD} = 5$V.

π_{PT}: PROM programming technique factor, $\pi_{PT} = 1.0$ for all devices except PROM's.

π_E: application environment factor=1 for a ground, benign environment.

C_1, C_2: functions of device complexity. For MOS LSI

$$C_1 = 1.75(10)^{-3} N_G^{0.4} \quad C_2 = 2.52(10)^{-4} N_G^{0.226}$$

where N_G is the number of gates.

C_3: function of package complexity. For nonhermetic packages,

$$C_3 = 2(10)^{-4} N_P^{1.23}$$

where N_P is the number of pins and it is given the value of 200.

Appendix C
Publications

ON THE ELIMINATION OF DATA COMPRESSION IN RESPONSE ANALYSIS FOR BIST

A. Bensouiah, S. Johnson and M. J. Morant
School of Engineering and Applied Science
University of Durham, UK

Abstract–High levels of coverage of classical and non-classical faults require deterministic test sequences. It is suggested, in this paper, that the deterministic test sequences be ordered in such a way that the fault-free output response is a trivial one that can be generated by simple on-chip circuitry, thereby obviating the need for test response compression.

The Built-In Self-Test property is defined as the provision of on-chip test pattern generation (TPG) and/or on-chip response analysis (RA). On the TPG side, besides the use of deterministic test patterns, pseudo-random testing has gained wide acceptance. For test response analysis, and besides the obvious bit-by-bit comparison, signature analysis is the technique most commonly used. The following table summarises the disadvantages of different combinations of TPG and RA approaches.

TPG RA	Pseudo-random	Deterministic
Signature analysis	Test sequence may not detect all faults. Even if all faults are detected, signature analysis introduces aliaising.	The deterministic nature of the test sequence is lost because of aliaising in signature analysis.
Bit-by-Bit Comparison	Test sequence may not detect all faults. On-chip implementation is not feasible.	On-chip implementation not feasible

Pseudo-random test sequences have excessive lengths and they do not cover all stuck-at faults (the so-called 'random-resistant faults'). This prompted some researchers to 'modulate' the pseudo-random sequences in order to increase their fault coverage [1,2]. Others have suggested that the circuit under test should be re-designed so that all random-resistant faults are eliminated [3]. The deficiencies of pseudo-random test sequences are further aggravated when non-classical faults, such as CMOS stuck-open, are taken into consideration.

For test response evaluation, signature analysis uses an n-bit register to compress an m-bit output sequence, where m is usually much larger than n. In this case there are 2^m possible output sequences, only one of which is the correct one. Therefore, there are approximately 2^{m-n} output sequences that get compressed into the same signature, including the correct signature, i.e., there are $2^{m-n} - 1$ faulty output responses that get compressed into the correct signature (if $n = 16$ and $m = 1024$, then $2^{1008} - 1$ faulty responses would escape detection if they occured). This is clearly unacceptable.

To achieve high levels of fault detection, deterministic test pattern generation has to be used, especially for non-classical faults. The deterministic nature of the TPG will be wasted if signature analysis is used to compress the output response. On the other hand, simple bit-by-bit comparison cannot be considered for on-chip implementation because of the large amount of data required.

It is suggested in this paper, that test patterns be generated deterministically, but to avoid signature analysis, and bit-by-bit comparison; the test patterns should be ordered in such a way that the output response is a trivial one (i.e., easily generated by on-chip circuitry). The most trivial signal is a toggling one. Therefore, it is suggested that the test patterns be ordered in such a way that the output of the circuit under test is toggled from one clock cycle to the next one. A similar idea has been suggested in a recent publication [4], where the the quotient output from a signature analyser is made periodic to ease its monitoring in order to reduce aliaising errors.

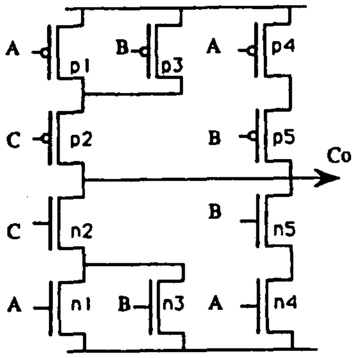
In the next section, we consider the testing of CMOS combinational circuits to show the motivation of the suggested approach, and following this, we address some of its practical issues.

TESTING A CMOS CIRCUIT

A transistor stuck-open fault in a CMOS circuit usually requires a pair of vectors for its detection. For a pFET (nFET) stuck-open fault, the first, or initialisation, vector must set the output of the gate containing the pFET (nFET) to 0 (1), and then the second, or test, vector must charge (discharge) the output throught a path containing the pFET (nFET) under test. If the fault is present, the gate output will remain unchanged, whereas the output of a fault-free circuit will toggle.

If stuck-open faults f_a and f_b are detected by the two-pattern tests (T_{a_0}, T_{a_1}) and (T_{b_0}, T_{b_1}), respectively, and if $T_{a_1} = T_{b_0}$ then the sequence $(T_{a_0}, T_{a_1}, T_{b_1})$ detects both faults. This optimisation, which is possible when the initialisation vector of some stuck-open fault is a test vector for another fault, will be much easier if some care is taken at the test pattern generation stage, since in most cases a stuck-open fault can have a number of initialisation vectors.

Making use of the above observation repeatedly would yield a complete test sequence that produces a toggling fault-free output. Moreover, the presence of any stuck-open fault (or any other detectable fault, for that matter) would prevent, at least, one transition at the output. As an example, consider the testing of the circuit shown in Fig. 1.



Faults	Initialisation vector(s)	Test vector(s)
p1	011, 110, (101), (111)	010
p2	110, 101, 011, (111)	010, 100, (000)
p3	110, 101, (011), (111)	100
p4, p5	011, 101, (110), (111)	001
n1	001, 100, (010), (000)	101
n2	001, 010, 100, (000),	011, 101, (111)
n3	001, 010, (100), (000)	011
n4, n5	010, 100, (001), (000)	110

Figure 1. Example circuit and tests for all its stuck-open faults.

The possible initialisation and test vectors for all stuck-open faults are also shown in Fig. 1. The input vectors that are between brackets should be ignored if the tests are to remain valid under arbitrary circuit delays. The complete test sequence for this circuit is ABC={ 011, 010, 110, 100, 101, 001, 011}. Note that the repetition of vector 011 is necessary since the initial state of the output node is assumed to be unknown.

PRACTICAL CONSIDERATIONS

The first criticism that one might find to the suggested approach is that an ordered deterministic test sequence requires a large area for the test pattern generator, as compared to an LFSR or a counter, for example. However, this may be offset by the simplicity of test response analysis. In a BIST environment, the analysis of the circuit response to a test sequence that toggles the output cannot be made any easier. Simple on-chip circuitry can be used on all nodes that need monitoring to perform bit-by-bit comparison with the fault-free response which, in this case, can be generated by simple on-chip hardware. This fault-free test response is the same for all circuits on the chip. It can be considered as simply an auxillary clock signal, of half the frequency of the chip's clock, that is broadcast to all nodes that need monitoring. Furthermore, the problems of aliaising and information losses in methods that rely on data compression (signature analysis, one's count, etc.) do no exist for the approach suggested in this paper, which makes it particularily suitable in situations where high levels of fault coverage are mandatory.

Another problem arises when considering multiple output circuits: It is impossible for two distinct outputs of the same circuit to be toggled by the same input test sequence (unless they are complements of each other). A solution would be to sequentially test the outputs of a same circuit and, in order to reduce test time, test as many circuits as possible in parallel. In addition, the testing of two, or more, outputs may overlap in time for the portions of the test sequence where these outputs are toggled simultaneously. In the most complex case, when the outputs have some logic in common, the principles of circuit segmentation, as used for pseudo-exhaustive testing [5] can be of great help in reducing the test time.

REFERENCES

1. E. Lindbloom, E. B. Eichelberger and O. P. Florenza "A Method for Generating Weighted Random Test Patterns" *IBM J. Res. and Develop.*, Vol. 33, No. 2, March 1989, pp 149-161

2. F. Brglez, G. Gloster and G. Kedem "Hardware-Based Weighted Random Pattern Generation for Boundary Scan" *Proc. IEEE International Test Conference*, 1989, pp 264-274

3. V. S. Iyengar and D. Brand "Synthesis of Pseudo-Random Testable Designs" *Proc. IEEE International Test Conference*, 1989, pp 501-508

4. S. Gupta, D. Pradhan and S. M. Reddy "Zero Aliaising Compression" *Fault Tolerant Computing Symposium*, 1990, pp 245-263

5. J. G. Udell, Jr. and E. J. McCluskey "Pseudo-Exhaustive Test and Segmentation: Formal Definitions and Extended Fault Coverage Results" *Fault Tolerant Computing Symposium*, 1989, pp 292-298

ACKNOWLEDGMENTS

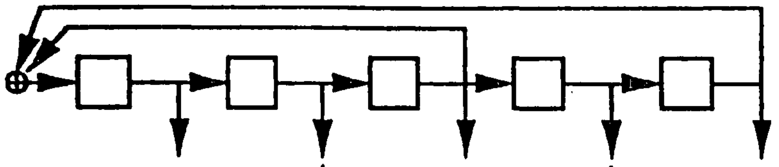
One of the authors (A.B) is sponsored by a grant from the ministry of higher education of Algeria.

ON THE ELIMINATION OF DATA COMPRESSION IN RESPONSE ANALYSIS FOR BIST

A. Bensouiah, S. Johnson, and M. J. Morant
School of Engineering & Applied Science
University of Durham, UK

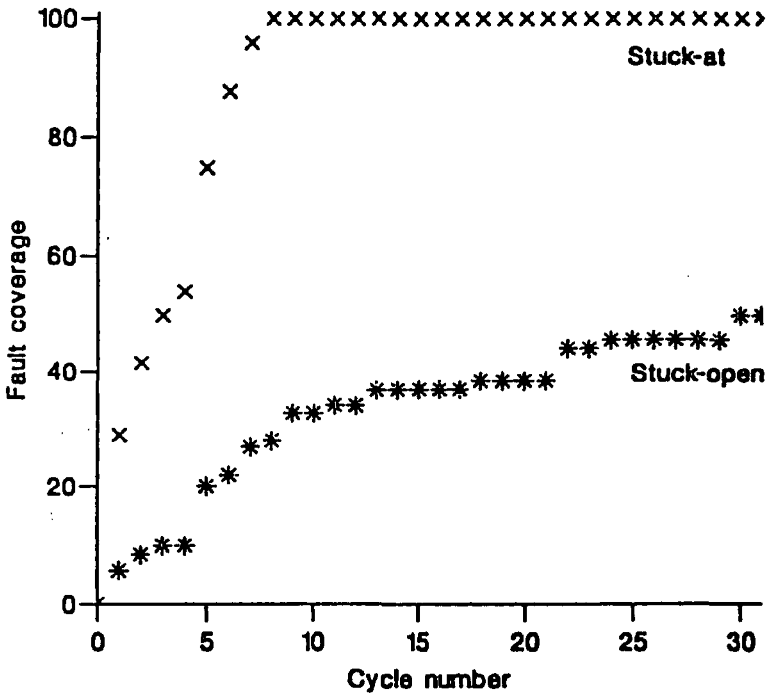
Purpose: Achieving high levels of fault coverage requires deterministic test sequences. It is suggested that the deterministic test sequences be ordered in such a way as to produce a trivial fault-free output response in order to simplify test response analysis and avoid fault masking effects that are common to all test data compression methods.

Consider the testing of the example circuit below by a test sequence produced by the LFSR shown.



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

Pseudo-random sequence fault coverage



- 10 out of the 35 detected stuck-open faults are in the input inverters.
- 18 out of the 31 input vectors do not detect any stuck-open faults.

A Limitation of Test Data Compression

All faults that affect more than one bit in the output response may be masked negating the benefits of having a complete test sequence.

Considering the example circuit and the pseudo-random test sequence, we have

- All stuck-at faults affect more than one cycle.
- 50% of stuck-at faults affect both outputs at the same cycle.
- 15 of the 35 stuck-open faults that are detected by the test sequence affect more than one cycle.

This leaves only 10 out of 60 stuck-open faults that are guaranteed to be detected by the combination of a pseudo-random test sequence and signature analysis, for example.

The sets of test vectors for all stuck-open faults in the pFET and nFET networks, respectively, of complex gate f1 are

$$C_{s_1} = \{00001, 00010, 00100, 11011, 11100\}$$

$$D_{s_1} = \{00011, 01001, 01110, 10100\}$$

If we use the elements of D_{s_1} as initialisation vectors for the elements of C_{s_1}, and vise versa, the resulting test sequence will be the shortest possible sequence that detects all stuck-open faults in the complex gate generating f1.

Note that the output f1 is an alternating signal and that any fault (stuck-at or stuck-open) will prevent at least one of the transitions.

	a	b	c	d	e	f1
1	0	0	0	1	1	0
2	0	0	0	0	1	1
3	0	1	0	0	1	0
4	0	0	0	1	0	1
5	0	1	1	1	0	0
6	0	0	1	0	0	1
7	1	0	1	0	0	0
8	1	1	0	1	1	1
9	0	0	0	1	1	0
10	1	1	1	0	1	1

It is clear that is not possible to toggle more than one circuit output with the same input test sequence (for this to be possible, the outputs must be equal or complement of each others for all the elements of their respective sets C_{s_i} and D_{s_i}).

A first solution would be to test one output at a time. The drawback is a longer test time and the need for separate test sequences for different outputs.

It is possible to order the test sequences for different outputs such that their testing overlap, as shown in the complete test sequence on the right.

Some control logic is required to disable the monitoring of f1 after vector 11 and to enable the monitoring of f2 after cycle 3.

	a	b	c	d	e	f1	f2		
1	0	0	1	0	0	1	0	f1	
2	0	1	1	1	0	0	0		
3	0	0	1	0	0	1	0		
4	0	1	0	0	1	0	1		f2
5	1	1	0	1	1	1	0		
6	1	0	1	0	0	0	1		
7	1	1	1	0	0	1	0		
8	0	0	0	1	1	0	1		
9	0	0	0	1	0	1	0		
10	1	1	1	1	1	0	1		
11	0	0	0	0	1	1	0		
12	1	1	0	1	0	1	1		
13	1	0	1	1	1	0	0		
14	1	1	0	1	0	1	1		
15	0	1	1	1	0	0	0		

V Conclusion

- High levels of fault coverage of classical and non-classical faults require deterministic test sequences.

- The deterministic nature of a test sequence would be waisted if data compression is used for response analysis.

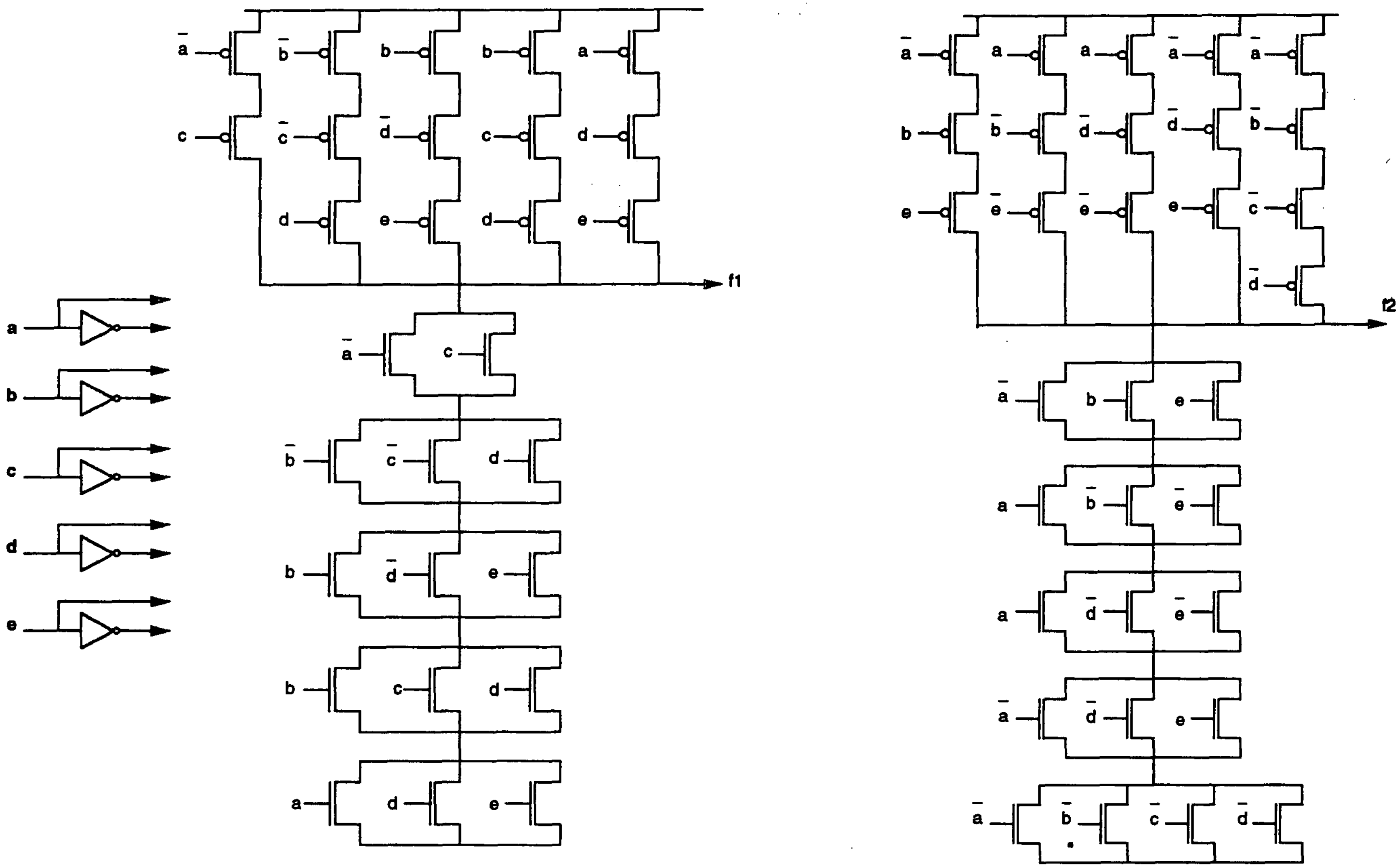
- The derivation of a test sequence that produces a trivial output response offers the following advantages:

    - Guaranteed high level fault coverage

    - Short test times

    - Simple and effective test response analysis

Example Circuit



RELIABILITY OF FAULT-TOLERANT VLSI CIRCUITS

A. Bensouiah, S. Johnson and M. J. Morant
School of Engineering and Applied Science
University of Durham, UK

Abstract–Fault-tolerance at the level of an integrated circuit would have a tremendous impact on the electronics industry. The dependence of system reliability on IC reliability becomes stronger as higher levels of integrations are achieved. Yet, there is no practical method for evaluating the reliability of a fault-tolerant IC. This paper presents such an evaluation method which consists of expressing the failure rate of a section of an IC as a function of its size and a combinatorial model to describe the dependence of the reliability of a fault-tolerant IC on the manufacturing yield.

Designing larger and denser integrated circuits has always been the driving force behind the development of the microelectronics industry. If one looks at dynamic random access memories, the flagship of the industry, increasing densities have been achieved by decreasing feature sizes and improving the control of the manufacturing process. But the reason that made 64K, and subsequently 256K and 1Mbit, DRAMs commercially available is the use of redundancy, in the form of spare rows and/or columns.

The successful use of redundancy in DRAMs has led to numerous investigations of fault-tolerant strategies in other types of ICs [1]. In most of the published literature, yield is used as the figure of merit to assess the suitability, or otherwise, of fault-tolerant strategies. However, it is always emphasised that the use of redundancy also enhances the reliability because larger chips, made possible by higher yields, imply smaller system's chip counts; in other words, the reliability improvment is a by-product of the yield improvment.

The reliability of an electronic system, comprising from a few tens to a few hundreds of ICs, is dominated by the reliability of the solder joints and the interconnections between chips [2]. However, when such a system is re-designed so that it comprises just one or two larger and denser chips, which is the current trend in the electronics industry, the system's reliability would certainly be improved, but at the same time, it becomes strongly dependent on the reliability of the intgrated circuit.

There is a notable lack of publications on reliabilty modelling of fault-tolerant ICs, as compared to the abundant literature on yield modelling. In fact, the subject has been addressed in only one publication [3]. The reason for this lack has been identified as being due to the following two problems:

1) The lack of any method to estimate the failure rate of a section of a chip, and
2) the dependence of reliability on the initial manufacturing yield.

Solutions to both problems are presented in this paper. In the next section, an expression for the evaluation of the failure rate of a section of an IC, as a function of the section's size, is derived. In the following section, the problem of reliability dependence on manufacturing yield is solved using a combinatorial model.

FAILURE RATE OF A SECTION OF AN IC

An expression that is commonly encountered in reliability modelling is the probability that at least one unit out-of-m units is working at time t. This expression is given by

$$\sum_{i=1}^{m} \binom{m}{i} P^i(1 - P)^{m-i} \qquad (1)$$

where P is the probability that a single unit is working at time t. Under the assumption of exponenetial distribution of times to failures, $P = e^{-\lambda t}$ where λ is he failure of a unit. If the units were single chips or assemblies of chips, then the failure rate λ can be estimated using the MIL-HDBK-217 [4]. However, in the case where all m units are parts of a single chip, expression (1) cannot be evaluated, because there is no method to estimate the failure rate λ of a single unit. In the proposed method, the failure rate of a section of a chip is expressed as a fraction of the failure rate of the entire chip. The justification of this approach is given next.

The reliabilty of an IC is given by $R_0 = e^{-\lambda_0 t}$. If the same chip is considered as a partition of n sections each comprising N_0/n gates, then we may compute the reliability of the chip as the product of the probabilities that each section is working, i.e., if we call λ_0' the failure rate of a section containing N_0/n gates, then the chip reliability is

$$R_0' = (e^{-\lambda_0' t})^n = e^{-n\lambda_0' t} \qquad (2)$$

Since R_0 and R_0' must be the same, we have $\lambda_0 = n\lambda_0'$. If we let $L = N_0/n$ then

$$\lambda_0' = \frac{L}{N_0}\lambda_0 \qquad (3)$$

which is the failure rate of a piece of silicon containing L gates that is part of an IC containing N_0 gates. This justifies the assumption made in [3].

DEPENDENCE OF RELIABILITY ON YIELD

The reliability of a non-redundant IC is given by $R_0 = e^{-\lambda_0 t}$ where $\lambda_0 = f(N_0)$ as given in [4]. In the case of a redundant chip, the failure rate is $\lambda_r = f(N_r)$, where N_r is the number of gates in the chip. According to [4], $\lambda_r > \lambda_0$ since $N_r > N_0$. However, the reliability of the fault-tolerant chip $R_r \neq e^{-\lambda_r t}$ since redundancy enables the chip to survive failures.

The reliability R_r of the redundant chip can be expressed as a function of the reliabilities of the different units in the chip. This function is clearly dependent on the way redundancy is introduced and on the topology of the resultant redundant chip. If we assume, for instance, that each functional unit of the chip is replicated m times, so that it can tolerate up to $m-1$ failures, then the probability that the redundant functional unit is working at time t is given by

$$R = \sum_{i=1}^{m} \binom{m}{i} P^i (1-P)^{m-i} = 1-(1-P)^m \qquad (4)$$

with $P = e^{-\lambda t}$ and λ is the failure rate of a unit as given by Eq. (3).

The dependence of the reliability of a fault-tolerant chip on the manufacturing yield is due to the fact that processing defects may render some of the redundant units unusable for tolerating operational failures. This effect can be taken into account by expressing Eq. (4) as

$$R = \sum_{i=1}^{m-k} \binom{m-k}{i} P^i (1-P)^{m-k-i} = 1-(1-P)^{m-k} \qquad (5)$$

where k is the number of units containing manufacturing defects. The above expression is then multiplied by the probability of having k defective units out-of-m units, and since there can be up to $m-1$ defective units. Eq. (5) becomes

$$R = \sum_{k=0}^{m-1} (1 - (1 - P)^{m-k}) \times \Pr\{k \text{ sections were defective}\} \qquad (6)$$

The last term in this expression is clearly a function of the yield. If we call Y the probability that a unit is defect free, then

$$\Pr\{k \text{ sections were defective}\} = \binom{m}{k}(1 - Y)^k Y^{m-k} \qquad (8)$$

and Eq. (6) becomes

$$R_r = \sum_{k=0}^{m-1} \binom{m}{k}(1 - Y)^k Y^{m-k}(1 - (1 - P)^{m-k}) \qquad (9)$$

The reliability is defined as the conditional probabilty that a system is working at time $t > 0$, given that the system was working at time $t = 0$:

$$R(t) = \Pr\{\text{System working at } t > 0 | \text{System working at } t = 0\} \qquad (10)$$

By definition of a conditional probability, we have

$$R(t) = \frac{\Pr\{\text{System working at } t > 0, \text{System working at } t = 0\}}{\Pr\{\text{System working at } t = 0\}} \qquad (11)$$

If the system is working at $t > 0$ then it must have been working at $t = 0$, therefore

$$R(t) = \frac{\Pr\{\text{System working at } t > 0\}}{\Pr\{\text{System working at } t = 0\}} \qquad (12)$$

The probability that a redundant functional unit is in a working state at $t = 0$ is the probability of having, at least, one unit out-of-m units defect free, i.e., the yield of the functinal unit, given by

$$Y_r = \sum_{i=1}^{m} \binom{m}{i} Y^i (1 - Y)^{m-i} = 1 - (1 - Y)^m \qquad (13)$$

The probability that the redundant functional unit is working at $t > 0$ is simply Eq. (9). Therefore, the reliability of the redundant functional unit is

$$R_r(t) = \frac{\sum_{k=0}^{m-1} \binom{m}{k}(1 - Y)^k Y^{m-k}(1 - (1 - P)^{m-k})}{1 - (1 - Y)^m} \qquad (14)$$

The reliability of the entire chip is a product of expressions similar to Eq. (14).

CASE STUDY

In this section, we illustrate the application of the previous expressions in a simple example. We consider a chip containing 1024 identical processing elements and the use of replication as a means of providing fault-tolerance. It is apparent that replication can be done at different levels: we may either replicate the entire chip, every PE, or every group of PEs. The evaluation method presented in this paper allows the selection of the appropriate level for replication, and also the replication factor, in order to attain a certain level of defect-tolerance and/or operational failure tolerance.

The first column of the following table gives the size of the group that is replicated. The first row gives the replication factor.

	Yield							Reliability						
m PEs	2	3	4	5	6	7	8	2	3	4	5	6	7	8
1	0.934	0.914	0.887	0.861	0.835	0.811	0.787	0.988	0.990	0.990	0.989	0.989	0.989	0.988
2	0.925	0.914	0.887	0.861	0.835	0.811	0.787	0.986	0.990	0.990	0.989	0.989	0.989	0.988
4	0.910	0.914	0.887	0.861	0.835	0.811	0.787	0.980	0.990	0.990	0.989	0.989	0.989	0.988
8	0.879	0.912	0.887	0.861	0.835	0.811	0.787	0.970	0.990	0.990	0.989	0.989	0.989	0.988
16	0.824	0.908	0.887	0.861	0.835	0.811	0.787	0.951	0.988	0.990	0.989	0.989	0.989	0.988
32	0.731	0.894	0.885	0.861	0.835	0.811	0.787	0.917	0.983	0.989	0.989	0.989	0.989	0.988
64	0.594	0.846	0.876	0.859	0.835	0.811	0.787	0.865	0.966	0.985	0.989	0.989	0.989	0.988
128	0.432	0.727	0.828	0.843	0.830	0.809	0.787	0.796	0.924	0.968	0.982	0.987	0.988	0.988
256	0.290	0.526	0.678	0.753	0.782	0.784	0.774	0.726	0.855	0.919	0.952	0.970	0.978	0.983
512	0.203	0.338	0.451	0.538	0.599	0.640	0.664	0.673	0.783	0.847	0.888	0.915	0.935	0.949
1024	0.180	0.249	0.306	0.354	0.393	0.425	0.451	0.645	0.739	0.794	0.831	0.858	0.878	0.894

An important observation from these results is that replication of the entire chip, or large sections of it, is not as benefical as replication at lower levels. As for the amount of redundancy, it can be seen that even for the lowest values of m (2 and 3), there is still a substantial improvment in yield and reliability over the non-redundant case, where $Y_0 = 10\%$ and $R_0 = 48\%$.

CONCLUSION

In this paper, we presented a practical method for estimating the failure rate of a section of an IC. This was then used to develop a model describing the dependence of the reliability of a fault-tolerant IC on the manufacturing yield. This allows a quantitative assessment of the effects of redundancy and partitioning on yield and reliability.

REFERENCES

1. G. W. Summerling, G. E. Dixon and A. K. J. Stewart "Assessment of a Non-Regular Cell Based Architecture for ULSI and WSI" *IEE Colloquium on Fault Tolerant Integrated Circuits*, 1986

2. R. Mckirdy and M. Lea "WSI: A Technology for Reliability" *International Workshop on Designing for Yield, Oxford*, 1986, pp 47-56

3. I. Koren and D. K. Pradhan "Modeling the Effect of Redundancy on Yield and Performance of VLSI" *IEEE Trans. on Computers*, 36, No 3, 1987, pp 344-355

4. U.S. Department of Defense *Military Standarization Handbook: Reliability Prediction of Electronic Equipment*, MIL-HDBK-217C, 1980

ACKNOWLEDGMENTS

One of the authors (A.B) is sponsored by the ministry of higher education of Algeria.

RELIABILITY OF FAULT-TOLERANT VLSI CIRCUITS

A. Bensouiah, S. Johnson, and M. J. Morant
School of Engineering & Applied Science
University of Durham, UK

Purpose: Development of reliability models for fault-tolerant integrated circuits that take into account the effect of manufacturing defects on the initial redundancy.

RELIABILITY EXPRESSIONS

FAILURE RATE OF A SECTION OF A CHIP:

N_0: Number of gates in the chip.
λ_0: Failure rate of the chip.
λ_0': Failure rate of a section of L gates.

$$\lambda_0' = \frac{L}{N_0}\lambda_0$$

λ_0 is as given in MIL-HDBK-217.

For this work, fault-tolerance is implemented by module redundancy. Two forms of redundancy are considered: (1) module replication and (2) the provision of spare modules.

Case 1: A module is replicated m times.

$$R = \Pr\{1 - \text{out} - \text{of}{-m} \text{ modules working}\} = \sum_{i=1}^{m} \binom{m}{i} R_0^i (1 - R_0)^{m-i} \tag{1}$$

where $R_0 = \Pr\{\text{a single module is working}\}$

Since manufacturing defects may affect some of the m modules, Eq. (1) becomes

$$R = \sum_{k=0}^{m-1} \sum_{i=1}^{m-k} \binom{m-k}{i} R_0^i (1 - R_0)^{m-k-i} \times \Pr\{k \text{ modules were defective}\}$$

and

$$\Pr\{k \text{ modules were defective}\} = \binom{m}{k} (1 - Y_0)^k Y_0^{m-k}$$

where Y_0 is the yield of a single module.

Hence, the probability that at least one-out-of-m modules is working is

$$R = \sum_{k=0}^{m-1} \binom{m}{k} (1 - Y_0)^k Y_0^{m-k} \sum_{i=1}^{m-k} \binom{m-k}{i} R_0^i (1 - R_0)^{m-k-i}$$

Case 2: S spares are provided for N modules.

$$R = \Pr\{N\text{–out – of–}(N+S)\text{ modules working}\} = \sum_{i=N}^{N+S} \binom{N+S}{i} R_0^i (1-R_0)^{N+S-i} \tag{2}$$

Because of manufacturing defects, Eq. (2) becomes

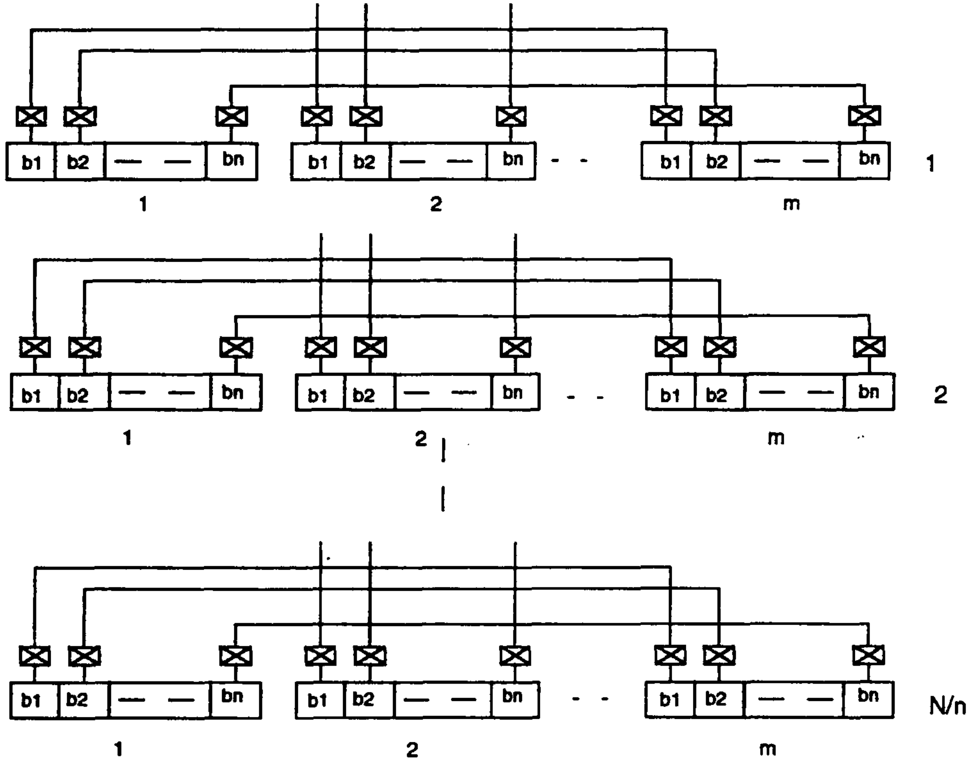
$$R = \sum_{k=0}^{S} \sum_{i=N}^{N+S-k} \binom{N+S-k}{i} R_0^i (1-R_0)^{N+S-k-i} \times \Pr\{k \text{ modules were defective}\}$$

$$R = \sum_{k=0}^{S} \binom{N+S}{k} (1-Y_0)^k Y_0^{N+S-k} \sum_{i=N}^{N+S-k} \binom{N+S-k}{i} R_0^i (1-R_0)^{N+S-k-i}$$

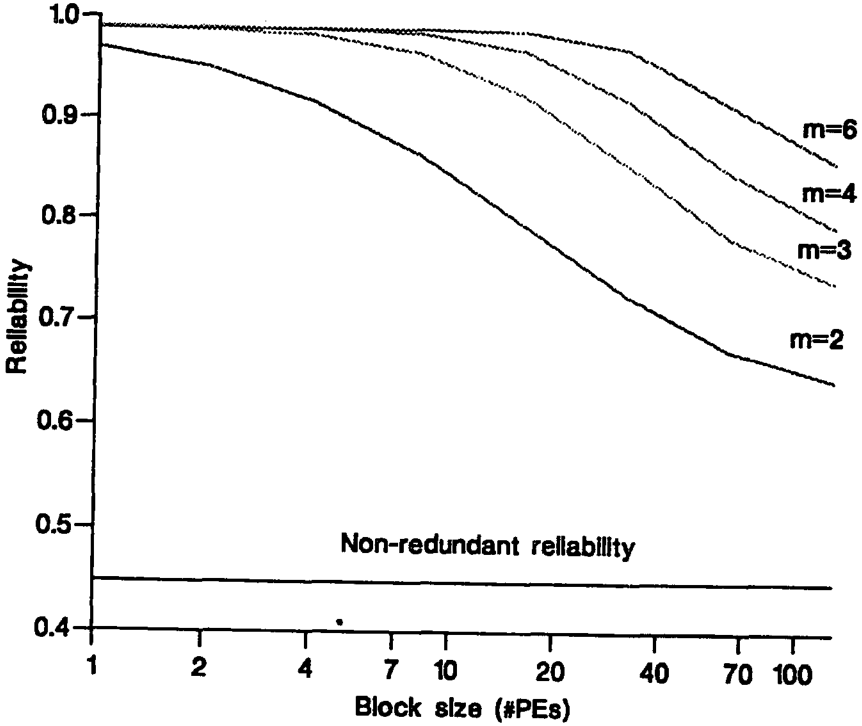
APPLICATION

We consider the introduction of redundancy in a chip of area 3 cm^2 and consisting of 128 identical PEs. The mean defect density is 1 cm^{-2} and the reliability is computed for $t = 0.1$ million hours.

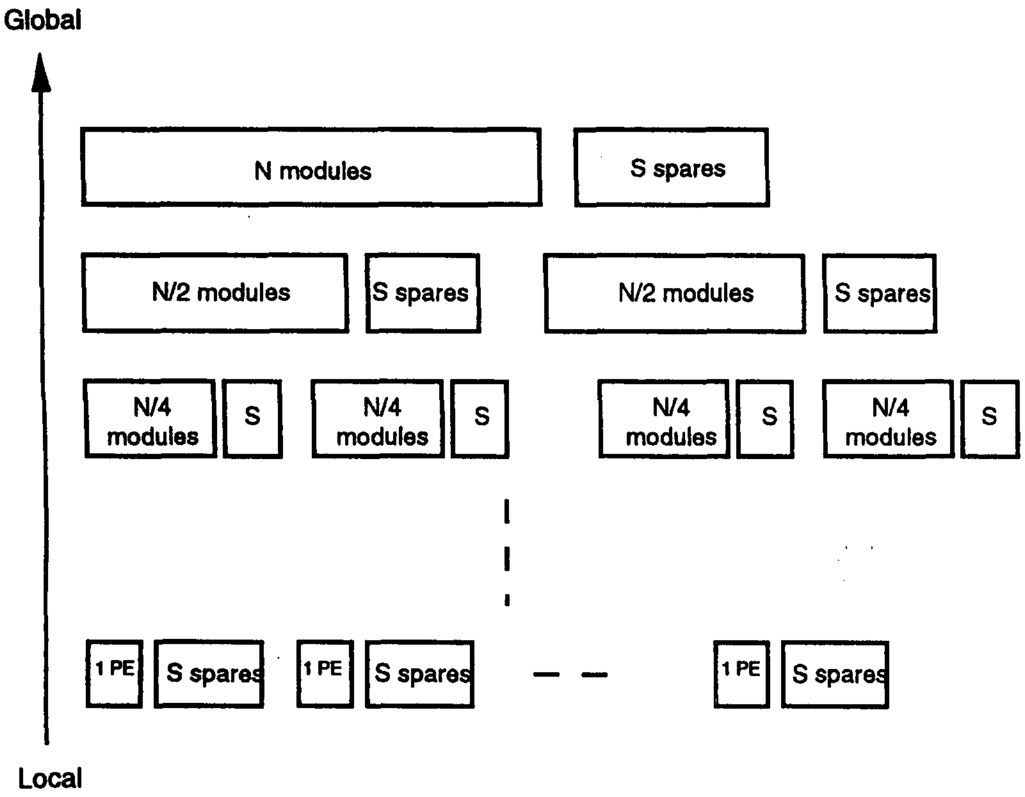
Replication: Each block of $n = 1, 2, 4, 8, 16, 32, 64,$ or 128 PEs is replicated m times and the corresponding chip reliability is evaluated. The maximum reliability is attained when small blocks are replicated as opposed to large blocks.

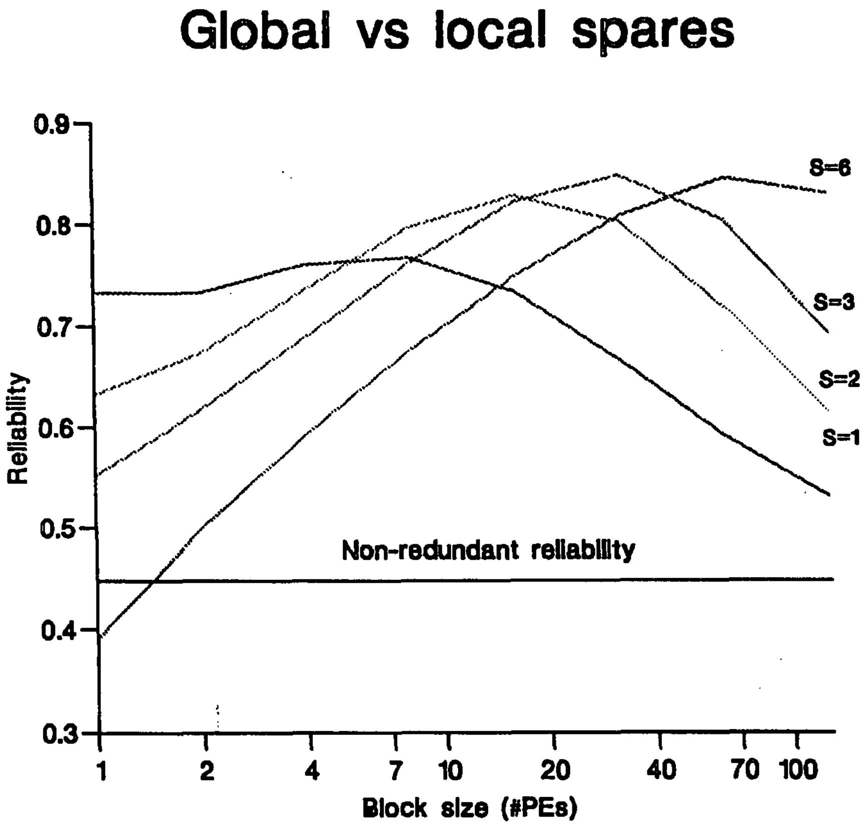


Replication at different levels



Global *vs* **Local Spares:** S spare PEs are provided for a block of $n = 1, 2, 4, 8, 16, 32, 64$, or 128 PEs and the resulting chip reliability is evaluated. If spares are provided to every PE then their number would be large. If spares are provided to the whole chip then the reconfiguration logic, whcih is a hardcore, would be very large.

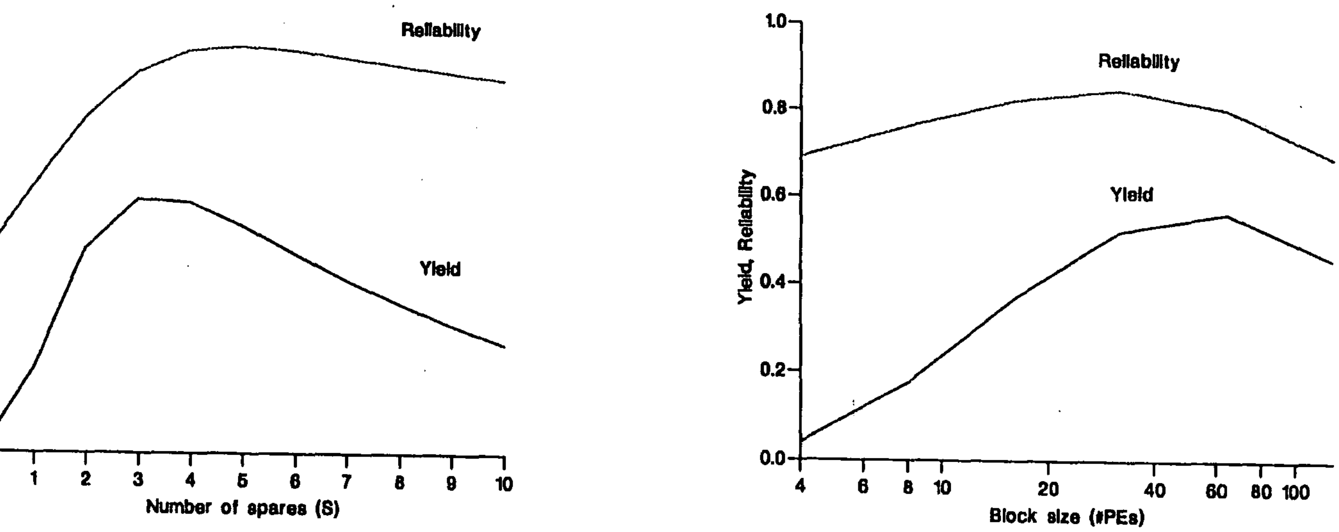




l *vs* Reliability Optimisation:

optimum yield and reliability values are attained for different number of spares and for different tioning.

Yield vs reliability optimisation



ıclusion:

'he fact that manufacturing defects may reduce the amount of redundancy available for tolerating ield failures implies that the reliability of fault-tolerant ICs is dependent on the yield.

[he optimum partitioning (block size) and the optimum amount of redundancy depend not only ın the process parameters and the characteristics of the non-redundant IC, but also depend on the ault-tolerant scheme.

References

[1] C. Mead and L. Conway *Introduction aux Systemes VLSI*, InterEDitions, 1983

[2] W. Maly, W. Moore and A Strojwas "Yield Loss Mechanisms and Defect Tolerance" *International Workshop on Designing for Yield, Oxford*, 1986, pp 3-30

[3] J. M. Pimbley and J. D. Meindl "MOSFET Scaling Limits Determined by Subthreshold Conduction" *IEEE Trans. on Electron Devices*, Vol. 36, No. 9, September 1989, pp 1711-1721

[4] J. D. Meindl "Ultra-Large Scale Integration" *IEEE Trans. on Electron Devices*, Vol. ED-31, No. 11, November 1984, pp 1555-1561

[5] J. R. Pfiester, J. D. Shott and J. D. Meindl "Performance Limits of CMOS ULSI" *IEEE J. Solid State Circuits*, Vol. SC-20, No. 1, February 1985, pp 253-263

[6] B. Eitan and D. Fraham-Benlchkowski "Surface Conduction in Short-Channel MOS Devices as a Limitation to VLSI Scaling" *IEEE Trans. on Electron Devices*, Vol. ED-29, No. 2, February 1982, pp 254-266

[7] D. C. Dorrough "A Methodical Approach to Analyzing and Synthesizing a Self-Repairing Computer" *IEEE Trans. on Computers*, Vol. C-18, No. 1, January 1969, pp 22-42

[8] T. L. Faulkner, C. W. Bartlett and M. Small "Hardware Design Faults: A Classification and some Measurements" *ICL Technical Jour.*, November 1982, pp 218-228

[9] H. G. Borrow "Proving the Correctness of Digital Hardware Designs" *VLSI Design*, Vol. 5, No. 7, July 1984, pp 64-77

[10] Round Table "Formal Verification–Is it Practical for Real World Design?" *IEEE Design & Test of Computers*, December 1989, pp 50-58

[11] M. P. Halbert "Self-Checking Computer Module Based on the VIPER Microprocessor" *Microprocessors and Microsystems*, Vol. 12, No. 5, June 1988, pp 264-270

[12] D. Pountain "Fast Track vs Failsafe" *BYTE*, July 1988, pp 305-309

[13] D. Lewin *Design of Logic Systems*, Van Nostrand Reinhold (UK) Co. Ltd. 1985

[14] T. E. Mangir "Sources of Failures and Yield Improvement for VLSI and Restructurable Interconnects for RVLSI and WSI: Part I–Souces of Failures and Yield Improvements" *Proc. IEEE*, Vol. 72, No. 6, June 1984, pp 690-708

[15] E. A. Amerasekera and D. S. Campbell *Failure Mechanisms in Semiconductor Devices*, John Wiley & Sons, Inc. 1987

[16] J. R. Srous and J. M. McGarrity "Radiation Effects on Microelectronics in Space" *Proc. IEEE*, Vol. 76, No. 11, November 1988, pp 1443-1469

[17] R. L. Pease, A. H. Johnston, and J. L. Azarewicz "Radiation Testing of Semiconductor Devices for Space Electronics" *Proc. IEEE*, Vol. 76, No. 11, November 1988, pp 1510-1526

[18] E. A. Doyle, Jr. "How Parts Fail" *IEEE Spectrum*, Vol. 18, No. 10, October 1981, pp 36-43

[19] M. R. Woods "MOS VLSI Reliability and Yield Trends" *Proc. IEEE*, Vol. 74, 1986, pp 1715-1728

[20] J. P. Hayes "Fault Modeling" *IEEE Design & Test of Computers*, April 1985, pp 8895

[21] W. Maly "Realistic Fault Modeling for VLSI Testing" *24th ACM/IEEE Design Automation Conference*, 1987, pp 173-180

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

[23] N. Burges, R. I. Damper, S. J. Shaw and D. R. J. Wilkins "Faults and Fault Effects in NMOS-Impact on Testability" *IEE Proc. G*, Vol. 132, No. 3, June 1985, pp 82-89

[24] K. C. Y. Mei "Bridging and Stuck-At Faults" *IEEE Trans. on Computers*, Vol. C-23, No. 7, July 1974, pp 720-727

[25] P. Banerjee and J. A. Abraham "Characterizing and Testing of Physical Failures in MOS" *IEEE Design & Test of Computers*, August 1984, pp 76-86

[26] J. Galiay, Y. Crouzet and M. Vergniault "Physical versus Logical Fault Models in MOS LSI Circuits: Impact on their Testability" *IEEE Trans. on Computers*, Vol. C-29, No. 6, June 1980, pp 527-531

[27] N. Burges, R. I. Damper, K. A. Totton and S. T. Shaw "Physical Faults in MOS Circuits and their Coverage by Different Fault Models" *IEE Proc. E*, Vol. 135, No. 1, January 1988, pp 1-9

[28] N. Burges and R. I. Damper "Inadequacy of the Stuck-At Fault Models for Testing MOS LSI Circuits: A Review of MOS Failure Mechanisms and some Implications for Computer-Aided Design and Test of MOS LSI Circuits" *Software & Microsystems*, Vol. 3, No. 2, April 1984, pp 30-36

[29] S. Gai, M. Mezzalma and P. Prinetto "A Review of Fault Models for LSI/VLSI Devices" *Software & Microsystems*, Vol. 2, No.2, April 1983, pp 44-53

[30] K. L. Kadandapani and D. K. Pradhan "Undetectability of Bridging Faults and Validity of Stuck-At Fault Test Sets" *IEEE Trans. on Computers*, Vol. C-29, No. 1, January 1980, pp 55-59

[31] M. Karpovski and S. Y. H. Su "Detection and Location of Input and Feedback Bridging Faults in Combinational Networks" *IEEE Trans. on Computers*, Vol. C-29, No. 6, June 1980, pp 523-527

[32] T. W. Williams and K. P. Parker "Design for Testability–A Survey" *Proc. IEEE*, Vol. 71, No. 1, January 1983, pp 441-455

[33] S. D. Millman and E. J. McCluskey "Detecting Bridging Faults with Stuck-at Test Sets" *Proc. IEEE International Test Conference*, 1988, pp 773-783

[34] D. R. Shertz and G. Metze "A New Representation for Faults in Combinational Digital Circuits" *IEEE Trans. on Computers*, Vol. C-21, August 1972, pp 858-866

[35] E. J. McCluskey and F. W. Clegg "Fault Equivalence in Combinational Logic Networks" *IEEE Trans. on Computers*, Vol. C-20, No. 11, November 1971, pp 1286-1293

[36] D. R. Schertz and G. Metze "On the Design of Multiple Fault Diagnosable Networks" *IEEE Trans. on Computers*, Vol. C-20, No. 11, November 1971, pp 1361-1364

[37] J. Jacob and N. N. Biswas "GTBD Faults and Lower Bounds on Multiple Fault Coverage of Single Fault Test Sets" *Proc. IEEE International Test Conference*, 1987, pp 849-855

[38] V. K. Agarwal and A. S. F. Fung "Multiple Fault Testing of Large Circuits by Single Fault Test Sets" *IEEE Trans. on Computers*, Vol. C-31, No. 11, November 1981, pp 855-865

[39] J. L. A. Huges "Multiple Fault Detection Using Single Stuck-at Test Sets" *IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems*, Vol. 7, No. 1, January 1988, pp 100-108

[40] R. L. Wadsack "Fault Modeling and Logic Simulation of CMOS and MOS Circuits" *The Bell Syst. Tech. Journal*, Vol. 75, No. 5, May-June 1978, pp 1449-1474

[41] S. K. Jain and V. D. Agrawal "Modeling and Test Generation Algorithms for MOS Circuits" *IEEE Trans. on Computers*, Vol. C-34, No. 5, May 1985, pp 426-433

[42] Z. Brazilai et. al. "Efficient Fault Simulation of CMOS Circuits with Accurate Models" *Proc. IEEE International Test Conference*, 1986, pp 520-529

[43] A. Miczo *Digital Logic Testing and Simulation*, John Wiley & Sons, Inc. 1987

[44] R. Dekker, F. Beenker and L. Thijssen "Fault Modeling and Test Algorithm Development for Static Random Access Memories" *Proc. IEEE International Test Conference*, 1988, pp 343-352

[45] S. Somenzi and S. Gai "Fault Detection in Programmable Arrays" *Proc. IEEE*, Vol. 74, No. 5, May 1986, pp 655-668

[46] J. P. Hayes "An Introduction to Switch Level Modeling" *IEEE Design & Test of Computers*, August 1987, pp 18-25

[47] R. E. Bryant "A Survey of Switch Level Algorithms" *IEEE Design & Test of Computers*, August 1987, pp 26-40

[48] R. Rajsuman, Y. K. Malaiya and A. P. Jayasumana "On Accuracy of Switch-Level Modeling of Bridging Faults in Complex Gates" *24th ACM/IEEE Design Automation Conference*, 1987, pp 244-250

[49] R. Rajusman, Y. K. Malaiya and A. P. Jayasumana "Limitations of Switch Level Analysis for Bridging Faults" *IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems*, Vol. 8, No. 7, July 1989, pp 807-811

[50] W. Maly, F. J. Ferguson and J. P. Shen "Systematic Fault Characterisation of Physical Defects for Fault Analysis in MOS IC Cells" *Proc. IEEE International Test Conference*, 1984, pp 390-399

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

[52] F. J. Ferguson and J. P. Shen "Extraction and Simulation of Realistic CMOS Faults Using Inductive Fault Simulation" *Proc. IEEE International Test Conference*, 1988, pp 475-484

[53] J. P. Shen and S. Hirschhorn "Switch-Level Techniques" *IEEE Design & Test of Computers*, August 1987, pp 15-16

[54] M. A. Breuer and A. D. Friedman *Diagnosis and Reliable Design of Digital Systems*, Computer Science Press, Inc. 1976

[55] P. K. Lala *Fault Tolerant and Fault Testable Hardware Design*, Prentice-Hall International, 1985

[56] I. L. Sayers and D. J. Kinniment "Low Cost Residue Codes and their Application to Self-Checking VLSI Systems" *IEE Proc. E*, Vol. 132, No. 4, July 1985, pp 197-202

[57] I. L. Sayers, D. J. Kinniment and E. G. Chester "Design of Reliable and Self-Checking VLSI Data Path Using Residue Coding" *IEE Proc. E*, Vol. 133, No. 3, May 1986, pp 169-179

[58] G. G. Langdon, Jr. and C. K. Tang "Concurrent Error Detection for Group Look Ahead Adders" *IBM J. Res. and Develop.*, September 1970, pp 563-573

[59] I. D. Elliott and I. L. Sayers "Implementation of a 32-Bit RISC Processor Incorporating Hardware Concurrent Error Detection and Correction" *IEE Proc. E*, Vol. 137, No. 1, January 1990, pp 88-102

[60] Y. Crouzet and C. Landrault "Design of Self-Checking MOS-LSI Circuits: Application to a Four-Bit Microprocessor" *IEEE Trans. on Computers*, Vol. C-29, No. 6, June 1980, pp 532-537

[61] M. M. Yen, W. K. Fuchs and J. A. Abraham "Designing for Concurrent Error Detection in VLSI: Application to a Microprogram Control Unit" *IEEE J. Solid State Circuits*, Vol. SC-22, No. 4, August 1987, pp 595-605

[62] A. J. Goode "Design Considerations of a Single Chip Fault-Tolerant Microprocessor" ***Software & Microsystems***, Vol. 4, No. 3, June 1985, pp 53-58

[63] J. H. Patel and L. Y. Fung "Concurrent Error Detection in ALU's by Recomputing with Shifted Operands" *IEEE Trans. on Computers*, Vol. C-31, No. 7, July 1982, pp 589-595

[64] W. Tung and J. H. Patel "Concurrent Error Detection in Iterative Logic Arrays" *Fault Tolerant Computing Symposium*, 1984, pp 10-

[65] J. H. Patel and L. Y. Fung "Concurrent Error Detection in Multiply and Divide Arrays" *IEEE Trans. on Computers*, Vol. C-32, No. 4, April 1983, pp 417-422

[66] B. W. Johnson, J. H. Aylor and H. H. Hana "Efficient Use of Time and Hardware Redundancy for Concurrent Error Detection in a 32-Bit Adder" *IEEE J. Solid State Circuits*, Vol. 23, No. 1, February 1988, pp 208-215

[67] D. A. Reynolds and G. Metze "Fault Detection Capabilities of Alternating Logic" *IEEE Trans. on Computers*, Vol. C-27, No. 12, December 1978, pp 1093-1098

[68] A. Lam, S. Chau and H. Luong "Design of a Class of Self Exercising Combinational Circuits" *Proc. IEEE International Test Conference*, 1985, pp 589-600

[69] L. J. Sigal and C. R. Kime "Concurrent Off-Phase Built-In Self-Test of Dormant Logic" *Proc. IEEE International Test Conference*, 1988, pp 934-941

[70] S. Chau and D. Rennels "Design Techniques for a Self-Checking Self-Exercising Processor" *Defect and Fault Tolerance in VLSI*, 1988, pp 191-202

[71] M. Kameyama and T. Higuchi "Design of Dependent Failure Tolerant Microcomputer System Using TMR" *IEEE Trans. on Computers*, Vol. C-29, No. 2, February 1980, pp 202-206

[72] P. K. Lala "An On-Chip Fault Tolerant Scheme" *Computer Design*, August 1982, pp 143-146

[73] T. Krikland and M. R. Mercer "Algorithms for Automatic Test Pattern Generation" *IEEE Design & Test of Computers*, June 1988, pp 43-55

[74] J. P. Roth "Diagnosis of Automata Failures: A Calculus and a Method" *IBM J. Res. and Develop.*, Vol. 10, No. 4, July 1966, pp 278-291

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

[76] O. H. Ibarra and S. K. Sahni "Polynomially Complete Fault Detection Problems" *IEEE Trans. on Computers*, Vol. C-24, No. 3, March 1975, pp 242-249

[77] S. T. Chakradhar, V. D. Agrawal and M. L. Bushnell "Polynomial Time Solvable Fault Detection Problems" *Fault Tolerant Computing Symposium*, 1990, pp 56-63

[78] K. Totton and S. Shaw "Self-Test: The Solution to the VLSI Test Problem" *IEE Proc. E*, Vol. 135, No. 4, July 1988, pp 190-195

[79] M. J. Bending "Hitest: A Knowledge Based Test Generation System" *IEEE Design & Test of Computers*, May 1984, pp 83-92

[80] M. J. Schofield "Knowledge Based Test Generation" *IEE Proc. G*, Vol. 132, No. 3, June 1985, pp 108-110

[81] E. J. McCluskey and S. Bozorgui-Nesbat "Design for Autonomous Testing" *IEEE Trans. on Computers*, Vol. C-30, No. 11, September 1981, pp 866-875

[82] K. D. Wagner, C. K. Chin and E. J. McCluskey "Pseudo-Random Testing" *IEEE Trans. on Computers*, Vol. C-36, No. 3, March 1987, pp 332-343

[83] I. Shperling and E. J. McCluskey "Circuit Segmentation for Pseudo-Exhaustive Testing via Simulated Annealing" *Proc. IEEE International Test Conference*, 1987, pp 58-65

[84] W. Song, K. C. Smith and W. M. Snelgrove "Partitioning for Pseudo-Exhaustive Testing is NP-Complete" *Electronics Letters*, 24th September 1987, Vol. 23, No. 20, pp 1060-1062

[85] J. Savir, G. S. Ditlow and P. H. Bardell "Random Pattern Testability" *IEEE Trans. on Computers*, Vol. C-33, No. 1, January 1984, pp 79-90

[86] E. B. Eichelberger and E. Lindbloom "Random-Pattern Coverage Enhancement and Diagnosis for LSSD Logic Self-Test" *IBM J. Res. and Develop.*, Vol. 27, No. 3, May 1983, pp 265-272

[87] R. Lisanke, F. Brglez, A. J. de Gauss and D. Gregoy "Testability Driven Random Test Pattern Generation" *IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems*, Vol. CAD-6, No. 11, November 1987, pp 1082-1087

[88] E. Lindbloom, E. B. Eichelberger and O. P. Florenza "A Method for Generating Weighted Random Test Patterns" *IBM J. Res. and Develop.*, Vol. 33, No. 2, March 1989, pp 149-161

[89] P. Thorel, R. David, J. Pulou and J. L. Rainard "Design for Random Testability" *Proc. IEEE International Test Conference*, 1987, pp 923-930

[90] V. S. Iyengar and D. Brand "Synthesis of Pseudo-Random Testable Designs" *Proc. IEEE International Test Conference*, 1989, pp 501-508

[91] P. Olivo, M. Damiani and B. Ricco "On the Design of Multiple Input Shift-Registers for Signature Analysis" *Proc. IEEE International Test Conference*, 1989, pp 936-936

[92] P. Bardell, W. H. McCanney and J. Savir *Built-In Test for VLSI: Pseudorandom Techniques*, John Wiley & Sons, Inc. 1987

[93] R. G. Bennetts *Design of Testable Logic Circuits*, Addison-Wesley Publishers Ltd. 1984

[94] R. G. Bennetts *Introduction to Digital Board Testing*, Crane, Russak & Company, Inc. 1982

[95] P. Goel and P. R. Moorby "Fault Simulation Techniques for VLSI Circuits" *VLSI Design*, Vol. 5, No. 7, July 1984, pp 22-26

[96] E. I. Muehdorf and A. D. Savkar "LSI Logic Testing–An Overview" *IEEE Trans. on Computers*, Vol. C-30, No. 1, January 1981, pp 1-17

[97] W. J. Dally and R. E. Bryant "Hardware Architecture for Switch Level Simulation" *IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems*, Vol. CAD-4, No. 3, July 1985, pp 239-250

[98] J. P. Hayes "Generation of Optimal Transition Count Tests" *IEEE Trans. on Computers*, Vol. C-27, No. 1, January 1978, pp 36-41

[99] J. E. Smith "Measure of the Effectiveness of Fault Signature Analysis" *IEEE Trans. on Computers*, Vol. C-29, No. 6, June 1980, pp 510-514

[100] T. W. Williams, W. Daehn, M. Gruetzner and C. W. Starke "Aliasing Errors in Signature Analysis" *IEEE Design & Test of Computers*, April 1987, pp 39-45

[101] W. C. Carter, H. C. Montgomery, R. J. Preiss and H. J. Reinheimer "Design of Serviceability Features for the IBM System /360" *IBM J. Res. and Develop.*, Vol. 8, NO. 2, 1964, pp 115-126

[102] F. F. Tsui *LSI/VLSI Testability Design*, McGraw-Hill, Inc. 1987

[103] S. Funatsu, M. Kawai and A. Yamada "Scan Design at NEC" *IEEE Design & Test of Computers*, June 1989, pp 50-57

[104] M. J. Ohletz, T. W. Williams and J. P. Mucha "Overhead in Scan and Self-Testing Designs" *Proc. IEEE International Test Conference*, 1987, pp 460-470

[105] S. Bhawmick, M. S. Khaira, P. P. Mishra, A. Das and A. Dasgupta "Threading Multiple Scan Paths in a VLSI Circuit" *Proc. IEEE International Test Conference*, 1988, pp 735-743

[106] V. D. Agrawal, K. T. Cheng, D. D. Johnson and T. Lin "Designing Circuits Using Partial Scan" *IEEE Design & Test of Computers*, April 1988, pp 8-15

[107] D. L. Liu and E. J. McCluskey "CMOS Scan-Path IC Design for Stuck-Open Fault Testability" *IEEE J. Solid State Circuits*, Vol. SC-22, No. 5, October 1987, pp 880-885

[108] R. P. van Riessn, H. G. Kerkhoff and A. Kloppenburg "Designing and Implementing an Architecture with Boundary Scan" *IEEE Design & Test of Computers*, February 1990, pp 9-19

[109] R. G. Bennetts "Integration of Design and Test: A Management Perspective" *IMS/Instrumatic/ Valid Seminar on 'Design for Test'*, October 1989

[110] J. J. LeBlanc "LOCST - A Built-In Self-Test Technique" *IEEE Design & Test of Computers*, November 1984, pp 45-52

[111] F. P. Beuder and M. J. Manner "HILDO: The Highly Integrated Logic Device Observer (modified BILBO)" *VLSI Design*, Vol. 5, No. 7, June 1984, pp 88-96

[112] M. M. Pradhan, E. J. O'Brien, S. L. Lam and J. Beausang "Circular BIST with Partial Scan" *Proc. IEEE International Test Conference*, 1988, pp 719-729

[113] L. T. Wang and E. J. McCluskey "Circuits for Pseudo-Exhaustive Test Pattern Generation" *IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems*, Vol. 7, No. 10, October 1988, pp 1068-1080

[114] H. Fujiwara "Design for High Speed Testability" *Proc. IEEE International Test Conference*, 1987, pp 1132-1133

[115] J. Kuban and J. Salick "Testability Features of the MC68020" *Proc. IEEE International Test Conference*, 1984, pp 821-826

[116] L. A. Basto and J. R. Kuban "Test Features of the MC68881 Floating Point Coprocessor" *Proc. IEEE International Test Conference*, 1985, pp 752-757

[117] G. Giles and K. Sheuer "Testability Features of the MC68851 PMMU" *Proc. IEEE International Test Conference*, 1986, pp 408-411

[118] F. W. Shih, H. H. Chao, S. Ong, J. Y. F. Hang, C. Trempel and A. L. Diamond "Testability Design for Micro/370. A System/370 Single Chip Microprocessor" *Proc. IEEE International Test Conference*, 1986, pp 412

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

[120] W. R. Moore "A Critical Review of Fault-Tolerant Chips and WSI" *Proc. of a Workshop Held at Southampton*, 1985, pp 1-8

[121] D. S. Gardner, J. D. Meindl and K. C. Saraswat "Interconnection and Electromigration Scaling Theory" *IEEE Trans. on Electron Devices*, Vol. ED-34, 1987, pp 633-643

[122] K. A. Sack et.al. "Evolution of the Concept of a Computer on a Slice" *Proc. IEEE*, Vol. 52, No. 12, December 1964, pp 1713-1720

[123] R. I. Petritz "Current Status of LSI Technology" *IEEE J. Solid State Circuits*, SC-2, No. 4, December 1967, pp 130-147

[124] J. W. Lathrop, R. S. Clark, J. E. Hull and R. M. Jennings "A Discretionary Wiring System at the Interface between Design Automation and Semiconductor Array Manufacture" *Proc. IEEE*, Vol. 55, 1967, pp 1988-1997

[125] E. Tammaru and J. B. Angell "Redundancy for LSI Yield Enhancement" *IEEE J. Solid State Circuits*, Vol. SC-2, 1967, pp 172-182

[126] D. F. Calhoun and L. P. McNamee "A Means of Reducing LSI Interconnection Requirements" *IEEE J. Solid State Circuits*, Vol. SC-7, No. 5, October 1972, pp 395-404

[127] I. M. McKintosh and D. Green "Programed Interconnections–A Release from Tyranny" *Proc. IEEE*, Vol. 52, December 1964, pp 1648-1651

[128] J. I. Rafel "The RVLSI Approach to Wafer Scale Integration" *Proc. of a Workshop Held at Southampton*, 1985, pp 199-203

[129] K. Yamashita, A. Kanasugi, S. Hijiya, G. Goto, N. Matsumura and T. Shirato "Wafer Scale 170000 Gate FFT Processor with Built-In Test" *IEEE J. Solid State Circuits*, Vol. 23, No. 2, April 1988, pp 336-342

[130] B. J. Donlan, G. F. Taylor, R. H. Steinvorth, A. S. Bergendahl and J. F. McDonal "Wafer Scale Integration Using Discretionary Microtransmission Line Interconnections" *Proc. of a Workshop Held at Southampton*, 1985, pp 31-45

[131] G. Chevalier and G. Saucier "A Programmable Switch Matrix for the Wafer Scale Integration of a Processor Array" *Proc. of a Workshop Held at Southampton*, 1985, pp 92-100

[132] D. W. Greve "Programming Mechanisms of Polysilicon Resistor Fuses" *IEEE J. Solid State Circuits*, Vol. SC-17, No. 2, April 1982, pp 349-354

[133] T. Mano, M. Wada, N Ieda and M. Tanimoto "A Redundancy Circuit for Fault-Tolerant 256K MOS RAM" *IEEE J. Solid State Circuits*, Vol. SC-1, No. 4, August 1982, pp 726-731

[134] D. C. Shaver, R. W. Mountain and D. J. Silversmith "Electron-Beam Programmable 128K Bit Wafer Scale EEPROM" *IEEE Electron Device Letter*, Vol. EDL-4, No. 5, May 1983, pp 153-155

[135] Y. Ikawa, K. Urui, M. Wada, T. Takada, M. Kawamura, M. Miyata, N. Amano and T. Shibata "A One-Day Chip: An Innovative IC Construction Approach Using Electrically Reconfigurable Logic VLSI with On-Chip Programmable Interconnections" *IEEE J. Solid State Circuits*, Vol. SC-21, No. 2, April 1986, pp 223-227

[136] C. R. Jesshope and L. Bentley "Low-Cost Restructuring Technique for WSI" *Electronics Letters*, 10th April 1986, Vol. 22, No. 8, pp 439-441

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

[138] B. F. Fitzgerald and E. P. Thoma "Circuit Implementation of Fusible Redundant Addresses on Random Access Memories for Productivity Enhancement" *IBM J. Res. and Develop.*, Vol. 24, No. 3, May 1980, pp 291-298

[139] F. J. Aichelmann "Fault Tolerant Design Techniques for Semiconductor Memory Applications" *IBM J. Res. and Develop.*, Vol. 28, No. 2, March 1984, pp 177-183

[140] W. R. Moore "A Review of Fault Tolerant Techniques for the Enhancement of Integrated Circuit Yield" *Proc. IEEE*, Vol. 74, No. 5, May 1986, pp 684-689

[141] W. R. Moore, A. P. H. McCabe and V. Bawa "Fault Tolerance in a Large Bit-Level Systolic Array" *Proc. of a Workshop Held at Southampton*, 1985, pp 259-272

[142] D. P. Siewiorek and R. S. Swarz *The Theory and Practice of Reliable System Design*, Digital Press, 1982

[143] K. Swada, T. Sakurai, Y. Uchino and K. Yamad "Built-In Self-Repair Circuit for High Density ASMIC" *IEEE 1989 Custom Integrated Circuits Conference*, (see 1.51) 1989

[144] R. M. Sedmack and H. L. Licbergot "Fault Tolerance of a General Purpose Computer Implemented in VLSI" *IEEE Trans. on Computers*, Vol. C-29, No. 6, June 1980, pp 492-500

[145] S. T. Temksbury and L. A. Hornak "Wafer Level System Integration: A Review" *IEEE Circuits and Device Magazine*, Vol. 5, No. 5, September 1989, pp 22-30

[146] W. H. Pierce *Failure-Tolerant Computer Design*, Academic Press, 1965

[147] A. E. Barbour and S. S. Wojcik "A General, Constructive Approach to Fault Tolerant Design Using Redundancy" *IEEE Trans. on Computers*, Vol. 38, No. 1, January 1989, pp 15-29

[148] T. F. Shwab and S. S. Yau "An Algebraic Model for Fault-Masking Logic Circuits" *IEEE Trans. on Computers*, Vol. C-32, No.9, September 1983, pp 809-825

[149] T. Leighton and C. E. Leiserson "Wafer Scale Integration of Systolic Arrays" *IEEE Trans. on Computers*, Vol. C-34, No. 5, May 1985, pp 448-461

[150] J. N. Coleman and R. M. Lea "Clock Distribution Techniques for Wafer Scale Integration" *Proc. of a Workshop Held at Southampton*, 1985, pp 46-53

[151] K. D. Warren, M. B. E. Abdelrazik, R. M. Kirdy and R. M. Lea "A Power Distribution Strategy for WSI" *Proc. of a Workshop Held at Southampton*, 1985, pp 54-61

[152] W. Chen, J. Mavor, P. B. Denyer and D. Renshaw "Superchip Architecture for Implementing Large Integrated Circuits" *IEE Proc. E*, Vol. 135, No. 3, May 1988, pp 137-150

[153] W. K. Fuchs and M. F. Chang "Diagnosis and Repair of Large Memories: A Critical Review and Recent Results" *Defect and Fault Tolerance in VLSI*, 1988, pp 213-225

[154] R. A. Evans and J. V. McCanny "A Fault Tolerant Algorithm for High Availability and Wafer Scale Systems" *IEE Colloquium on Fault Tolerant Integrated Circuits*, 1986

[155] B. T. Murphy "Cost-Size Optima for Monolithic Integrated Circuits" *Proc. IEEE*, December 1964, pp 1537-1545

[156] C. H. Stapper, F. M. Armstrong and K. Saji "Integrated Circuit Yield Statistics" *Proc. IEEE*, Vol. 71, No. 4, April 1983, pp 453-470

[157] I. Chen and A. J. Strojwas "RYE: A Realistic Yield Simulator for VLSIC Structural Failures" *Proc. IEEE International Test Conference*, 1987, pp 31-42

[158] C. H. Stapper "Fault Simulation Programs for Integrated Circuit Yield Estimation" *IBM J. Res. and Develop.*, Vol. 33, No. 6, November 1989, pp 647-652

[159] J. L. Gallace "Reliability" *VLSI Handbook, J. Di Giacomo (ed.)*, McGraw-Hill Publishing Company, 1989, pp 27.3-27.40

[160] D. L. Crook "Evolution of VLSI Reliability Engineering" *ESREF 91*, October 1991, Bordeaux, France, pp 293-312

[161] H. A. Shaft, D. A. Baglee and P. E. Kennedy "Building-In-Reliability: Making it Work" *ESREF 91*, October 1991, Bordeaux, France, pp 19-32

[162] S. D. Millman and E. J. McCluskey "Detecting Stuck-Open Faults with Stuck-At Test Sets" *IEEE 1989 Custom Integrated Circuits Conference*, 1989

[163] B. W. Woodhall, B. D. Newman and A, G. Sammuli "Empirical Results on Undetected CMOS Stuck-Open Faults" *Proc. IEEE International Test Conference*, 1987, pp 166-170

[164] C. F. Hawkins and J. M. Soden "Electrical Characteristics and Test Considerations of Gate Oxide Shorts in CMOS Integrated Circuits" *Proc. IEEE International Test Conference*, 1985, pp 544-555

[165] M. Shoji *CMOS Digital Circuit Technology*, Prentice-Hall, Inc., New-Jersey 1988

[166] S. M. Reddy, M. K. Reddy and V. D. Agrawal "Robust Tests for Stuck-Open Faults in CMOS Combinational Logic Circuits" *Fault Tolerant Computing Symposium*, 1984, pp 44-49

[167] S. M. Reddy and M. K. Reddy "Testable Realization of FET Stuck-Open Faults in CMOS Combinational Logic" *IEEE Trans. on Computers*, Vol. C-35, No. 8, August 1986, pp 742-754

[168] B. Gupta, V. K. Malaiya and R. Rajsuman "On Designing Robust Testable CMOS Combinational Circuits" *IEE Proc. E*, Vol. 136, No. 4, July 1989, pp 329-338

[169] N. K. Jha and J. A. Abraham "Design of Testable CMOS Logic Circuits under Arbitrary Delays" *IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems*, Vol. CAD-4, No. 3, July 1985, pp 264-269

[170] R. Rajsuman, V. K. Malaiya and A. P. Jayasumana "CMOS Stuck-Open Fault Testability" *IEEE J. Solid State Circuits*, Vol. 24, No. 1, February 1989, pp 193-194

[171] R. Rajsuman, A. P. Jayasumana and Y. K. Malaiya "CMOS Open-Fault Detection in the Presence of Glitches and Timing Skews" *IEEE J. Solid State Circuits*, Vol. 24, No. 4, August 1989, pp 1055-1061

[172] J. M. Soden and C. F. Hawkins "Test Consideration for Gate Oxide Shorts in CMOS Circuits" *IEEE Design & Test of Computers*, August 1986, pp 56-64

[173] C. Crapuchettes "Testing CMOS I_{DD} on Large Devices" *Proc. IEEE International Test Conference*, 1987, pp 310-315

[174] L. K. Horming, J. M. Soden, R. R. Fritzmeier and C. F. Hawkins "Measurement of Quiescent Power Supply Current for CMOS ICs in Production Testing" *Proc. IEEE International Test Conference*, 1987, pp 300-309

[175] M. Keating and D. Meyer "A New Approach to Dynamic I_{dd} Testing" *Proc. IEEE International Test Conference*, 1987, pp 316-321

[176] P. Nigh and W. Maly "A Self-Testing ALU Using Built-In Current Sensing" ***IEEE 1989 Custom Integrated Circuits Conference***, 1989

[177] P. Nigh and W. Maly "Test Generation for Current Testing" *IEEE Design & Test of Computers*, February 1990, pp 26-38

[178] S. K. Chakravarty "On the Complexity of Computing Tests for CMOS Gates" *IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems*, Vol. Vol. 8, No. 9 September 1989, pp 9973-980

[179] A Bensouiah, S. Johnson and M. J. Morant "On the Elimination of Data Compression in Response Analysis for BIST" 2^{nd} ***European Symposium on Reliability of Electron Devices, Failure Physics and Analysis***, Bordeaux, France, 1991, pp 595-597

[180] B. F. Cockburn and J. A. Brzozanski "Switch Level Testability of CMOS PLA" *Integration, the VLSI Journal*, Vol. 9, No. 1, February 1990, pp 49-80

[181] D. H. Merlino and J. Hadjilogiou "Built-In Test Strategy for the next Generation Military Avionic Hardware" *Proc. IEEE International Test Conference*, 1988, pp 969-975

[182] C. R. Kime, H. H. Kwan, J. K. Lemke, and G. B. Williams "Built-In Self-Test Methodology for VLSI Data Paths" *Proc. IEEE International Test Conference*, 1984, pp 327-337

[183] J. P. Mucha, W. Daehn and J. Gross "Self-Test in Standard Cell Environment" *IEEE Design & Test of Computers*, December 1986, pp 35-41

[184] C. W. Starke "Built in Test for CMOS Circuits" *Proc. IEEE International Test Conference*, 1984, pp 309-314

[185] G. L. Craig and C. R. Kime "Pseudo-Exhaustive Adjacency Testing: A Built in Self Test Approach for Stuck-Open" *Proc. IEEE International Test Conference*, 1985, pp 126-137

[186] J. A. Bate and D. M. Miller "Exhaustive Testing of Stuck-Open Faults in CMOS Combinational Circuits" *IEE Proc. E*, Vol. 135, No.1, January 1988, pp 10-16

[187] M. Cohn and S. Even "Design of Shift Register Generators for Finite Sequences" *IEEE Trans. on Computers*, Vol. C-18, No. 7, July 1969, pp 660-662

[188] W. Daehn and J. Mucha "A Hardware Approach of Self Testing Large Programmable Arrays" *IEEE Trans. on Computers*, Vol. C-30, No. 11, November 1981, pp 829-833

[189] M. Katoozi "Built-In Test of CMOS Structured Logic with Realistic Fault Models" *PhD Thesis*,, University of Washington, 1989

[190] J. E. Price "A New Look at Yields of Integrated Circuits" *Proc. IEEE*, August 1970, pp 1290-1291

[191] B. T. Murphy "Comment on 'A New Look at Yields of Integrated Circuits'" *Proc. IEEE*, July 1971, pp 1128-1128

[192] R. M. Warner "Applying a Composite Model to the IC Yield Problem" *IEEE J. Solid State Circuits*, SC-9(3), June 1974, pp 86-95

[193] C. H. Stapper "On a Composite Model to the IC Yield Problem" *IEEE J. Solid State Circuits*, SC-10, December 1975, pp 537-339

[194] S. M. Hu "Some Consideration in the Formulation of the IC Yield Statistics" *Solid State Electronics*, Vol. 22, 1979, pp 205-211

[195] C. H. Stapper "Comments on 'Some Consideration in the Formulation of IC Yield'" *Solid State Electronics*, 24, 1981, pp 127-132

[196] R. M. Warner "A Note on IC Yield Statistics" *Solid State Electronics*, 24(11), 1981, pp 1045-1047

[197] C. H. Stapper "LSI Yield Modeling and Process Monitoring" *IBM J. Res. and Develop.*, May 1976, pp 228-234

[198] S. M. Sze *VLSI Technology*, McGraw-Hill Book Company, 1988

[199] E. I. Muehldof "Fault Clustering: Modeling and Observation on Experimental Chips" *IEEE J. Solid State Circuits*, Vol. SC-10, August 1975, pp 237-244

[200] A. V. Ferris-Brabhu "Defects, Faults and Semiconductor Device Yield" *Defect and Fault Tolerance in VLSI*, 1988, pp 33-46

[201] A. Gupta and J. W. Lathrop "Yield Analysis of Large Integrated Circuit Chips" *IEEE J. Solid State Circuits*, Vol. SC-7, No. 5, October 1972, pp 389-395

[202] A. V. Ferris-Prabhu "Modeling Critical Area in Yield Forecasts" *IEEE J. Solid State Circuits*, Vol. SC-20(4), August 1985, pp 874-878

[203] A. V. Ferris-Prabhu "Defect-Size Variations and Their Effect on the Critical Area of VLSI Devices" *IEEE J. Solid State Circuits*, Vol. SC-20(4), August 1985, pp 878-882

[204] H. Walker and S. W. Director "VLASIC: A Catastrophic Fault Yield Simulator for Integrated Circuits" *IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems*, Vol. CAD-5, No. 4, October 1986, pp 541-556

[205] W. Maly, A. J. Strojwas and S. Director "VLSI Yield Prediction and Estimation: A Unified Framework" *IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems*, Vol. CAD-5, No. 1, January 1986, pp 114-130

[206] I. Chen and A. Strojwas "Realistic Yield Simulation for VLSI Structural Failures" *IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems*, Vol. CAD-6, No. 6, November 1987, pp 965-980

[207] C. H. Stapper "Simulation of Spacial Fault Distribution for Integrated Circuit Yield Estimation" *IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems*, Vol. 8, No. 12, December 1989, pp 1314-1318

[208] T. E. Mangir and A. Avizienis "Fault Tolerant Design for VLSI: Effect of Interconnect on Yield" *IEEE Trans. on Computers*, Vol. C-31, No. 7, July 1982, pp 487-493

[209] I. Koren and D. K. Pradhan "Introducing Redundancy into VLSI Design for Yield and Performance Enhancement" *Fault Tolerant Computing Symposium*, 1985, pp 330-335

[210] C. Thibeault, Y. Savaria and J. L. Houle "A New Yield Formula for Fault-Tolerant Large-Area Devices" *Defect and Fault Tolerance in VLSI*, 1988, pp 53-64

[211] H. Bolouri and M. Lea "ULSI and WSI Yield Estimation: An Empirical Approach" *International Workshop on Designing for Yield, Oxford*, 1986, pp 205-212

[212] I. Koren and C. H. Stapper "Yield Models for Defect-Tolerant VLSI Circuits: A Review" *Defect and Fault Tolerance in VLSI*, 1988, pp 1-21

[213] U. Ramacher "A Cost Orientated Redundancy Model for Defect-Tolerant VLSI/WSI Systems" *International Workshop on Designing for Yield. Oxford*, 1986, pp 233-245

[214] W. Chen, J. Mavor and D. Renshaw "Yield Estimation for Serial Superchip" *IEE Proc. E*, Vol. 136, No. 3, May 1989, pp 187-196

[215] C. H. Stapper "Productivity Optimization of VLSI DRAM Chips with Redundant Circuits" *International Workshop on Designing for Yield. Oxford*, 1986, pp 41-45

[216] C. Thibeault, Y. Savaria and J. L. Houle "Impact of Reconfiguration Logic on the Optimization of Defect-Tolerant Integrated Circuits" *Fault Tolerant Computing Symposium*, 1990, pp 158-165

[217] R. Mckirdy and M. Lea "WSI: A Technology for Reliability" *International Workshop on Designing for Yield, Oxford*, 1986, pp 47-56

[218] D. I. Heinmann, N. Mittal and K. S. Trivedi "Availability and Reliability Modeling for Computer Systems" ***Advances in Computers***, Vol. 31, M. C. Yovits (ed.), Academic Press, 1990, pp 176-231

[219] U.S. Department of Defense *Military Standardization Handbook: Reliability Prediction of Electronic Equipment*, MIL-HDBK-217C, 1980

[220] V. W. Ng and A. Avizienis "A Unified Reliability Model for Fault Tolerant Computers" *IEEE Trans. on Computers*, Vol. C-29, No. 11, November 1980, pp 1002-1011

[221] K. S. Trivedi *Probability and Statistic with Reliability, Queuing and Computer Science Applications*, Prentice-Hall Inc., Englewood Cliffs, N. J. 07632

[222] R. A. Cliff "Acceptable Testing of VLSI Components which Contain Error Corrector" *IEEE Trans. on Computers*, Vol. C-29, No. 2, February 1980, pp 125-134

[223] T. Haifley and A. Bhatt "Fault-Tolerant ICs: The Reliability of TMR Yield-Enhanced ICs" *IEEE Trans. on Reliability*, Vol. R-36, No. 2, June 1987, pp 224-226

[224] I. Koren and M. A. Breuer "On Area and Yield Consideration for Fault Tolerant Processor Arrays" *IEEE Trans. on Computers*, Vol. C-33, No. 1, January 1984, pp 21-27

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

[226] I. Koren and D. K. Pradhan "Yield and Performance Enhancement through Redundancy in VLSI and WSI Multiprocessor Systems" *Proc. IEEE*, Vol. 74, No. 5, May 1986, pp 699-711

[227] C. H. Stapper "Modeling of Defects in Integrated Circuits Photolithographic Patterns" *IBM J. Res. and Develop.*, Vol. 28, No. 4, July 1984, pp 461-475

[228] G. W. Sumerling "Processing Towards WSI" ***Journal of Semicustom ICs***, Vol. 6, No. 1, 1988, pp 5-17

[229] A Bensouiah, S. Johnson and M. J. Morant "Reliability of Fault-Tolerant VLSI Circuits" 2nd ***European Symposium on Reliability of Electron Devices, Failure Physics and Analysis***, Bordeaux, France, 1991, pp 125-128

[230] T. Nakayama, Y. Miyawaki, K. Kobayashi, Y. Terada, H. Arima, T. Matsukawa and T. Yishikara "A 5-V One-Transistor 256K EEPROM with Page-Mode Erase" *IEEE J. Solid State Circuits*, Vol. 24, No. 4, August 1989, pp 911-915