Automated Fault Diagnosis and Empirical Validation of Fault Models in CMOS VLSI Circuits

Ashish Pancholy

Bachelor of Engineering

Department of Electrical Engineering

McGill University

A thesis submitted to the Faculty of Graduate Studies and Research

in partial fulfillment of the requirements for the degree of

Master of Engineering

Table of Contents

List of Figures

List of Tables

Thought

"The more I study, the more I learn about how little I know."

Acknowledgements

I would like to express my gratitude to my thesis supervisor, Prof. Janusz Rajski, for his guidance and assistance through the course of my graduate work. His enthusiasm, vision and insight are greatly appreciated.

I would like to thank Robert Hum and Phil Wilcox of the CAD group at Bell-Northern Research (BNR) for providing me with the opportunity of using their facilities and equipment for the experimental work.

Thanks are also due to Larry McNaughton of the Design for Testability group at BNR for his involvement, insight and useful suggestions throughout the course of the project.

I would like to thank the staff and students of the VLSI Design Laboratory for making my stay here a truly enjoyable experience. In particular, thanks to Rob Aitken, Henry Cox, Serge Gaiotti, Michael Howells, Michael Moscovitch, Fidel Muradali, Pierre Parent, Jacek Slaboszewicz and Avrum Warshawsky for their help and assistance on numerous occasions, and to François Larochelle for his help with the French version of the abstract.

On a personal level, I would like to thank my parents, Gargi Rani and Maganlal Pancholy, and my sister and brother-in-law, Alka and Rakesh Jain, for their constant encouragement, understanding and support. I would especially like to thank my brother, Rajiv Pancholy, without whose support this work would have been much harder to accomplish.

Abstract

The selection of adequate fault models is crucial to generating tests of high quality for complex digital VLSI circuits. This thesis presents a methodology to perform empirical validation of fault models and to get measures of effectiveness of test sets based on the targeted fault models.

The methodology is based on the automated fault diagnosis of test circuits, representative of the class of circuits being studied and designed to capture the characteristics of the fabrication process, cell libraries and CAD tools used in their development

The methodology is applied to study the faulty behaviour of random logic environments for an experimental VLSI fabrication process. A test circuit is designed, using CMOS technology, and a statistically significant number of samples fabricated. The samples are tested and, subsequently, diagnosed, using a set of software tools developed for the purpose Results of the ensuing analysis are presented.

Résumé

La sélection de modèles adéquats de pannes est cruciale pour la génération de tests de qualité supérieure pour les circuits ITGÉ complexes. Ce mémoire présente une méthodologie pour effectuer la validation empirique de modèles de pannes et mesurer l'efficacité des ensembles de tests basés sur ces modèles

La méthodologie est basée sur le diagnostic automatique de pannes de circuits d'essais, représentatifs de la classe de circuits étudiée et conçus pour avoir les caractéristiques du procédé de fabrication, de la bibliothèque de cellules et des outils de CAO utilisés pour leur développement

La méthodologie est appliquée pour étudier le comportement défectueux de logique aléatoire pour un procédé expérimental de fabrication ITGÉ Un circuit d'essais est conçu en utilisant la technologie CMOS et un nombre statistiquement significatif d'echantillons est fabriqué Les échantillons sont vérifiés et diagnostiqués en utilisant des logiciels spécialement développés à cette fin; les résultats de cette analyse sont présentés

Chapter 1 Introduction

Recent advances in VLSI technology have made possible the fabrication of digital circuits with millions of transistors on single chips With new photolithography techniques supporting even smaller feature-sizes, coupled with other improvements in submicron fabrication technology, circuit densities can be expected to increase further While the number of components within digital integrated circuits (ICs) has increased rapidly, the number of input/output (I/O) pins the logic is *accessed* with has grown only by a modest amount, resulting in an increase in the ratio of logic to I/O pins and a consequent reduction in the controllability and observability of these circuits The complexity of current VLSI chips, coupled with their limited controllability and observability have made their testing a challenging task

In present-day circuits, testing accounts for approximately one-third of the chip's production costs [BhMuHa89]. Furthermore, the cost of testing increases between five and ten times per level of packaging [WilPar83][BaMcSa87] Hence, from an economic viewpoint, it is advantageous to detect faulty devices early and to prevent them from being *shipped out* as "good" parts

The importance of *fault modelling* in developing highly cost-effective test strategies for VLSI circuits is illustrated in figure 1 1 A large number of different physical defects can potentially occur in CMOS circuits These are caused mostly by silicon substrate inhomogeneities, local surface contamination and photolithography related processing [Ravi81] Through a process of abstraction, a fault model *maps* this relatively large number of defects into a small number of modelled faults The task of test pattern generation then reduces to that of devising tests to *cover* all modelled faults The percentage of all possible faults covered by a test set is called its *fault coverage*

Although the single stuck-at fault model has traditionally been used in the industry, there is growing interest in extended fault models representing *transition* [WaLiRI87]-[ShMaFe85] and *stuck-open* [WoNeSa87] faults There is also increasing evidence that layout-related fault models including *bridging* faults account for most non-classical faults [ShMaFe85] *Multiple* fault models have similarly elicited renewed interest since the test sets developed for them stand to cover more physical failures [Maly87] The selection of adequate fault models is crucial to achieving a high quality of testing since faults covered by them are used as targets in tools like fault simulators and automated test pattern generators. Fault models are equally important in BISTed circuits [1], where these faults serve as targets and are used to grade the efficiency of the test scheme

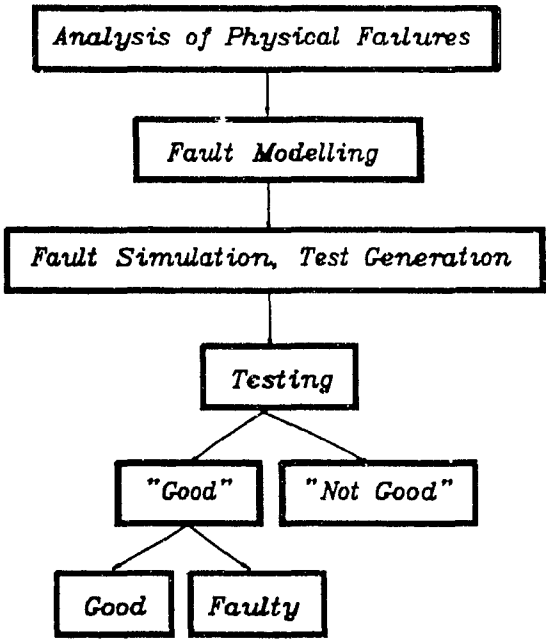


Figure 1.1 The complete process of testing

The process of fault modelling is especially constructive in the framework of scan design [WilPar83], where the increased observability and controllability, with the support of adequate CAD tools, make it possible to achieve arbitrarily high coverage of modelled faults. This, in turn, provides a strategy to make trade-offs between the cost of testing and its *quality*, where test quality is defined as a measure of the effectiveness of the test process to detect real circuit faults — rather than just the modelled ones Clearly, an

[1] Circuits employing Built-In Self Test (BIST)

effective fault model, coupled with a "high" fault coverage results in a correspondingly high test quality

It is essential for any effective test strategy to periodically *validate* its fault models Such validation is important to maintain a high quality of test because of the extreme dependence of fault modelling on design, technology and fabrication processing related parameters which are not always stable For example, an effective way of modelling physical defects in TTL devices may not be suitable for logically identical devices fabricated in CMOS technology [Wadsac78] Even worse, fault models suitable for a given fabrication process may not be effective for similar devices fabricated under a different process

Since fabrication processes are constantly being *tuned* to improve process yields and even replaced to accommodate new technologies — a recent addition being BiCMOS — it is important to periodically monitor the effects of such changes on the validity of fault models

This thesis presents a methodology to perform the validation of fault models using automated fault diagnosis as a key element The methodology is applied to a large number of samples of a test circuit, designed and fabricated specifically for the experiment, and the ensuing results are presented

This chapter presents a brief overview of some existing fault models, looks at state of the art schemes to build effective fault models and validate existing ones, introduces the new methodology for the validation of fault models and describes how the rest of the thesis is organized.

1.1 Overview of Existing Fault Models

This section briefly describes some of the currently used fault models. Circuit "lines" represent interconnects between circuit elements at the same level of abstraction the fault model assumes — for example the gate level

1.1.1 Stuck-At Faults

One of the earliest references to stuck-at faults is made by Eldred [Eldred59] Under the stuck-at fault assumption, any line in a circuit may have either of two kinds of faults

— a stuck-at-one or a stuck-at-zero. A line stuck-at-one remains at a logic one irrespective of the polarity of the driving signal. Similarly, a line stuck-at-zero remains at a logic zero irrespective of the polarity of the driving signal

In a circuit with n lines, there are n different stuck-at fault sites Considering stuck-at faults of all multiplicities, each line in the circuit can be either stuck-at-one, stuck-at-zero or fault-free. Since there is only one state for which the circuit is fault-free — when *all* lines are simultaneously fault-free — there can be $3^n - 1$ stuck-at faults of different multiplicities in the circuit.

1.1.2 Stuck-Open and Stuck-On Faults

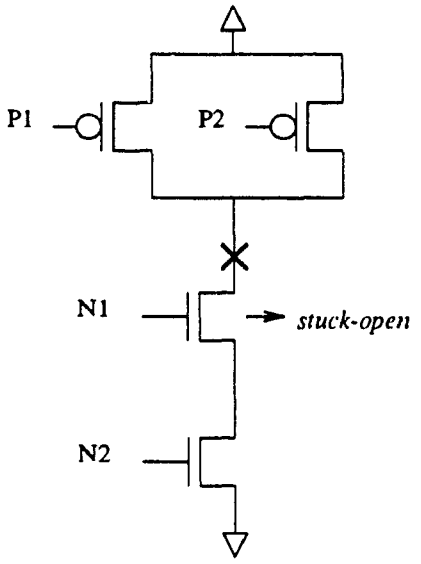


Figure 1.2 Faulty CMOS NAND gate

Stuck-open faults, first proposed by Wadsack [Wadsac78] to handle failure modes unique to MOS circuits, are transistor-level faults. A transistor is said to be stuck-open if its *channel* (between source and drain) behaves like an open circuit Figure 1 2 illustrates a CMOS NAND gate where transistor $N1$ is stuck-open Clearly, the gate cannot be reset to a logic zero unless it already e ists in that state When both inputs are held high, the output of the (faulty) gate goes into a *high impedance* state since the output is not driven by any transistor group Stuck-open faults fall into the class of *sequential* faults since they require a particular *sequence* of vectors to test them — to initialize the output and to exercise the transistor in question to toggle the output value [Wadsac78]

Consequently, an exhaustive test set is not necessarily a complete test set for stuck-open faults. Many methods have been proposed to generate test sets for such faults both at the switch [BasCou84][BKLNPW82] and gate levels [RajCox86][Chandr83][JaiAgr83]-[Elziq82][ElzClo81]. Arbitrary delays and timing skews in circuits can, however, combine to invalidate tests for stuck-open faults by violating the setup conditions required to initialise them. A *robust* test is defined to be one which cannot be so invalidated [ReReAg84].

Stuck-On faults are also transistor faults where the channel of the faulty transistor behaves like an electrical short. Test sets complete for stuck-at and stuck-open faults are not guaranteed to detect all stuck-on faults. In addition, detection of stuck-on transistor faults requires monitoring the I_{dd} steady-state current. Methods have previously been proposed to generate tests to detect such faults [Malary84][ReAgJa84].

1.1.3 Transition and Delay Faults

Under the transition fault assumption, any line in a circuit may have a transition fault of either of two polarities — slow-to-rise or slow-to-fall [BarRos83][WaLiRI87]. A line is slow-to-rise if it takes an unacceptably long time to propagate a logic zero to one transition on it. Conversely, a line is slow-to-fall if it takes an unacceptably long time to propagate a logic one to zero transition on it. Transformations have previously been formulated for modelling stuck-open faults in fully-complementary MOS structures on the basis of transition faults [CoxRaj88b]. Methods to simulate transition faults have also been proposed [LevMen86][WaLiRI87].

Two different delay fault models have been used in earlier literature — the *gate* delay model and the *path* delay model. A gate delay fault is similar to a transition fault where the slow-to-rise and slow-to-fall delay faults are associated with gate inputs and outputs [StoBar77][LesShe80]. In the path delay model, on the other hand, delays are associated with paths through the circuit [Smith85]. If a combinational network fails to propagate data from an input to an output within an acceptable time, and it has no stuck-at faults, it is said to contain a path delay fault.

1.1.4 Bridging Faults

Bridging faults occur when two, or more, lines in a circuit are shorted and, as a result, a wired logic operation performed at the junction [Mei74]. Circuit "lines" in this

case are not restricted to being metal interconnects only For example, bridging faults may occur due to shorts between two different *layers* in the circuit The actual wired logic operation performed as a result of the fault depends on the technology of the circuit *Feedback* bridging faults — involving shorts between lines on the same *path* of a circuit — can also lead to sequential behaviour of combinational circuits [Mei74] If there are n lines in a circuit, there can be $n \cdot (n-1)$, or $O\left(n^2\right)$ different bridging fault sites, assuming only two separate lines get shorted at any site

1.1.5 Specialized Fault Models

In addition to the basic fault models described earlier, a number of specialized fault models exist which address the problem of fault modelling for various functional units These fault models serve the twofold purpose of providing effective modelling of modes of failure unique to the units and of providing *higher-level* fault modelling in functional blocks too complex to model at the gate, or switch, level

Specialized fault models include *crosspoint* faults for PLAs, inter-cell *coupling* faults for RAMs and *instruction decoding* faults in microprocessors

1.2 Techniques to Develop and Validate Fault Models

The best known systematic method to generate a list of faults that can potentially occur in a specific integrated circuit (IC) is Inductive Fault Analysis (IFA) [ShMaFe85] IFA, proposed, and later implemented, by Shen, Maly and Ferguson [ShMaFe85], approaches the problem of fault modelling in a *bottom-up* manner. It determines a list of faults most likely to occur in a given device, based upon its layout topology and the "defect statistics" of the fabrication process used

1.2.1 Inductive Fault Analysis

The authors of IFA believe that most traditional and existing test techniques are inadequate primarily because of three reasons: 1) the use a single-level or flat approach based on a gate level representation of the circuit, 2) the use of technology independent fault models — like the line stuck-at fault model and, 3) the inability to generate a "ranked" fault list

IFA proposes to compile a "ranked fault list" so that the most likely to occur faults may be tested first in a *stop-at-first-fault* production test strategy It is the authors' intention to emphasize "local" or spot defects leading to circuit faults rather than "global" defects at the wafer level since they believe that failures resulting from global defects can be easily identified and the fabrication process tuned to fix them

To develop an appropriate fault modelling approach making use of process-related information, physical failures are viewed at four different levels of abstraction process-level, structure-level, circuit-level and logical-level For example, a point defect caused by a pinhole in the SiO_2 (insulating) layer may be viewed as an open region in the SiO_2 layer, a contact between two conducting layers, a short between two wires at the circuit level, or possibly a stuck-at-one fault at the logical level

Defects are first "generated" for a given layout using statistical information obtained from the fabrication process In an n-layer process, a defect at any spot is assumed to be either one of two basic types an extra layer where there should not be one or a missing layer where there should For any region in the circuit, a single defect assumption is used The generated defects are then "screened" so that only the electrically significant ones are further analyzed Since a number of different physical failures can lead to the same electrical faults, a *K–map* of "electrically equivalent classes" is used to perform the necessary mapping from physical defects to electrical faults

Electrically significant defects are analyzed to extract their faulty behaviours in terms of a "primitive fault list" The fault types in the primitive fault list are 1) shorts between two or more equipotential regions, 2) breaks resulting in the separation of a single equipotential region into two or more, 3) introduction of "new" devices to the circuit and 4) changes in the behaviour of existing devices in the circuit Note that it is possible for two different physical defects to produce identical faulty behaviour in terms of their electrical characteristics The primitive fault list is then interpreted to produce the circuit-level fault list The five types of circuit faults considered are 1) line stuck-at, 2) transistor stuck-at (ON or OFF), 3) floating line, 4) bridging and 5) miscellaneous Miscellaneous faults include all faults not covered by the first four categories These include the creation of parasitic active devices and faults involving power and ground lines

A ranked fault list is then compiled on the basis of the circuit-level faults list, accounting for the number of physical defects which can result in each fault. The flowchart in figure 1 3 illustrates the procedure

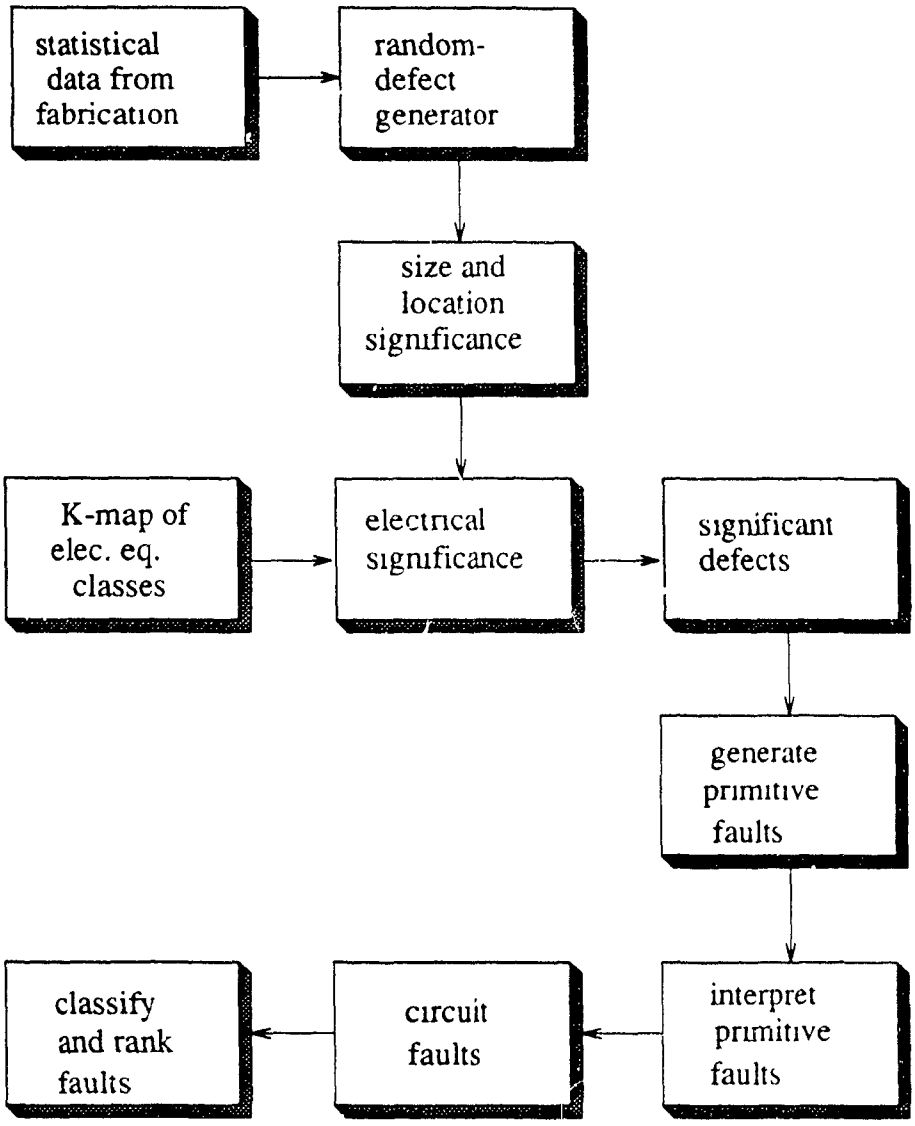


Figure 1.3 Steps involved in the IFA procedure

Fault Types	Number of Defects	Percentage of faults
Line Stuck-ats	132	28%
Transistor stuck-ats	70	15%
Bridging	101	21%
Miscellaneous	29	6%

Table 1.1 Comparison of Fault Types for Example Circuit

The authors have demonstrated the IFA procedure using an example circuit implemented in NMOS technology The circuit is a full-adder cell containing 29 transistors

Their results are summarized in table 1.1. The method has also been applied successfully to analyse regular structures like memories [DeBeTh88]

1.2.2 Behavioral Analysis of Fault Models

While IFA addresses the problem of *generating* adequate fault models for a given circuit, much work has centered around establishing the effectiveness of the stuck-at fault model for CMOS circuits The following two sections describe empirical studies, performed on test chips, to establish the ability of complete stuck-at test sets to detect stuck-open faults While the approach taken by IFA to generate a list of potential faults associated with a given circuit is "bottom-up" in nature, starting at the physical device level, the work described below uses a "top-down" or behavioral approach

1.2.2.1 Empirical Results on Undetected CMOS Stuck-Open Failures

This experimental work, devised and performed by Woodhall, Newman and Sammuli [WoNeSa87] addresses the question of how adequate CMOS stuck-at testing is to detect stuck-open faults

The experimental procedure consists of using a purely combinational ASIC CMOS circuit as a test vehicle and testing it with three separate test patterns — for stuck-open faults, for stuck-at faults (only) and using the manufacturing test The test vehicle used by the authors contained 1854 stuck-open fault sites out of which 616 stuck-open faults were equivalent to stuck-at faults The stuck-open fault set, therefore, consisted of 1238 distinguishable single stuck-open faults

Since any stuck-open test set inherently stands to cover a subset of stuck-at faults, a stuck-at only test set was devised from the stuck-open test set by inserting "de-initialising" vectors between the initialization and test vectors for each vector pair and before each initialization vector This insured that the stuck-at-only test sequence covered all of the stuck-at faults covered by the stuck-open test set but no stuck-opens

Each wafer die clearing an initial parametric test for continuity and short was subjected to the three different test sets All faulty dies were classified according to the procedure outlined in table 1 2

Test Sequence		Failure
Stuck-Open	Stuck-At-Only	Classification
Pass	Pass	No Failure
Pass	Fail	Inconsistency
Fail	Pass	Stuck-Open Fault
Fail	Fail	Stuck-At Fault

Table 1.2 Fault Classification Procedure

A total of 4552 die were tested out of which 1255 had one or more stuck-at faults, 44 had one or more stuck-open faults — possibly in additions to stuck-at faults — and there were 4 die which had stuck-open faults only Each of these 4 die escaped detection with the manufacturing test which provided a 100% coverage of stuck-at faults The observed stuck-open escape rate of stuck-open faults with the manufacturing test given by

$$E_{SOP} = \frac{Stuck-Open\ Escaping\ Die}{Passing\ Die + Stuck-Open\ Escaping\ Die}$$

was 0.121%

Assuming that the number of faults on a device, n, has a Poisson distribution, the authors derived a relation between the stuck-at escape rate, E_{SA}, and the stuck-at fault coverage, f_{SA}, given by:

$$E_{SA}(f_{SA}) = (1 - e^{-a}) \left[\frac{1}{1 - e^{-a}(1 - f_{SA})} \right] - 1$$

where the constant a can be experimentally determined Using experimental results from their test chip, they found that at 99% stuck-at fault coverage, the stuck-at escape rate was greater than the stuck-open escape rate, which was assumed to be the same as with a 100% stuck-at test set

1.2.2.2 Experiments to Study the Effects of CMOS Stuck-Open Faults

Turner, Leet, Prilik and McLean developed a complete test system from test generation to device testing, that supports CMOS stuck-open and short faults [TuLePM85] The test tools work within the framework of LSSD [WilPar82] To allow for I_{dd} (current) monitoring for the detection of transistor short (stuck-on) faults, an attachment was designed and built for the tester which translates the value of the observed current into a voltage level on a tester channel

An experiment was developed using "CMOS masterslice" hardware to determine the nature and frequency of stuck-open and short faults and to establish whether such faults required the use of special tests — other than those for stuck-at faults — to detect them. In addition to several "test circuits", the test hardware contained an implementation of the TI 74181 4-bit ALU and a collection of all "books" (cells) in the masterslice library, each of which was completely controllable and observable from primary inputs and outputs The hardware was developed within an LSSD environment

Essentially, two different tests were applied to the test hardware — one generated by "LSSDSTUCK" for detecting stuck-at faults and the other generated by ETG (Enhanced Test Generator) for stuck-open faults LSSDSTUCK's stuck-at fault coverage was better than ETG's stuck-at coverage ETG's stuck-open fault coverage, however, was 2.5 times that of LSSDSTUCK's

The two tests were applied under different test conditions to measure any power supply and input level sensitivit es Further, some tests were repeated with current monitoring to establish the effectiveness of the I_{dd} monitor (for transistor shorts). Table 1 3 summarizes the nature of the 8 different tests applied

Pattern Source	I_{dd} monitor	Test Conditions
LSSDSTUCK	No	Nominal
LSSDSTUCK	Yes	Nominal
LSSDSTUCK	No	High Conductance
LSSDSTUCK	No	Low Conductance
ETG	No	Nominal
ETG	Yes	Nominal
ETG	Yes	High Conductance
ETG	Yes	Low Conductance

Table 1.3 Logic Test Summary

A "statistically significant" number of chips was tested in a "development environment". The authors established that 93.9% of all chips that passed parametric tests, either passed or failed *all* eight "corners" (stages) of the test By analyzing the corners on which the remaining 6.1% failed, suspected fail mechanisms were determined It was found that

2.9% were I_{dd} monitor fails, 1.5% were "corner sensitive fails" and 0.7% failed the LSSDSTUCK test but passed the ETG test 1.0% were not analyzed since they accounted for only statistically insignificant samples from each category For the 2.9% where the current monitor failed, the failures were attributed to "generally leaky chips" rather than particular transistor shorts since *all* patterns failed the I_{dd} monitor In all chips that passed the ETG but failed the LSSDSTUCK, the stuck-at faults associated with the LSSDSTUCK patterns were faults not tested by the ETG test

It was concluded, by the authors, that: 1) no chip failed on a stuck-open fault, and 2) no chip failed on a stuck-on fault. The "primary" reason proposed by the authors to explain why transistor faults did not affect the yield in the experiments is that the layout of the masterslice provided for a low probability of occurrence of transistor faults

1.3 Proposed Methodology

Since IFA primarily addresses the process of deriving ranked fault models and not the quality of the test sets developed for some faults, it cannot be used to determine the escape rates for tests developed under different fault model assumptions The experimental procedures developed in [WoNeSa87] and [TuLePM85] are well suited for studying the effectiveness of stuck-at test sets to cover stuck-open faults but, again, cannot be used to evaluate the effectiveness or quality of test sets developed under other fault model assumptions

This thesis develops a methodology for the experimental validation of fault models A *top-down* approach is proposed (unlike IFA), where the effectiveness of a fault model is determined by the ability, or lack of it, to diagnose faults, based on the given fault model, in test chips designed specifically to capture the characteristics of a given design process and subsequently tested with a near "ideal" test set

In brief, the key elements of the proposed method are

1) The design and fabrication of an "easily diagnosable" test chip, representative of the class of circuits being studied, the CAD tools used in its design and the fabrication process used in its production.
2) The derivation of an extremely "robust" test set, capable of detecting faults from within a wide range of fault models;

3) The development of a set of diagnostic tools to perform automated diagnosis on faulty circuits,
4) The use of the results to get measures of "effectiveness" of the target fault model(s) and test sets generated to cover all modelled faults, and
5) The validation of the results of the diagnosis using an electron-beam voltage-contrast circuit prober

Since the validity of fault models is very sensitive to the layout topology, design technology and fabrication process of the circuit(s) under consideration, the results of such a study cannot be generalized beyond the confines of a particular set of these parameters which are common only to a particular *class* of circuits Hence, the test circuit is designed to *capture* this parametric information and be representative of the class of circuits being studied Moreover, since the methodology uses automated diagnosis as a key element, the circuit is also designed to be "easily diagnosable"

The test set is designed to cover faults from a number of different fault models Faults are covered in a *robust* fashion, implying their detection even in the presence of other faults, as much as possible This definition of robustness, in the context of a generic test set, remains independent of the definition of robust tests in the context of stuck-open and transition faults In addition, the test set provides a high *diagnostic resolution* so that different faults may be distinguishable from one another, as much as possible A set of software tools is designed to perform automated fault diagnosis on faulty chips, assuming different fault models The results of the diagnosis are interpreted and other experiments performed to get measures of effectiveness of fault models for the class of circuits being studied An electron-beam voltage-contrast prober is then used to validate the results of the diagnosis and, by extension, the methodology The methodology is carried out to study the faulty behaviour of purely combinational random logic circuits for an experimental 1.5μ double layer metal (DLM) CMOS process.

Chapter two of the thesis discusses issues related to the design of the test chip and contains a detailed description of the chip designed to serve as a test vehicle Chapter three describes the theory behind the test set and the steps involved in its generation. The theory behind fault diagnosis, including a historical perspective and some state of the art techniques, is discussed in chapter four, along with a discussion of some implementation issues Chapter five discusses measures of effectiveness of fault models and test sets and

proposes experimental ways to get their estimates. Experimental results, from the analysis of a number of faulty devices, and their subsequent validation are presented in chapter six Chapter seven concludes the thesis.

Chapter 2 The Test Chip

The test chip serves as a means to study, in general, the manifestations of physical defects in terms of the faulty behaviour of the class of circuits it represents. Since the thrust of the work is to study random logic environments in combinational circuits, the chip is required to *realistically* duplicate such an environment The generality of its representation, however, cannot be stretched beyond the confines of its design and process related parameters which characterise the circuit.

This chapter presents the philosophy behind the design of the chip, discusses the parameters which characterize it and describes its design in detail.

2.1 Design Philosophy

The empirical approach to failure analysis, outlined earlier in chapter 1, imposes two basic requirements on the design of the test chip: its structure should be "suitable" for fault diagnosis, providing for a high "resolution" of faults, and its design should be "representative" of the general class of circuits being studied The basic philosophy behind the design, then, is to satisfy each requirement to the maximum extent possible without violating the constraints imposed by the other.

2.1.1 Small Fault Equivalence Classes

Two faults f_1 and f_2 are said to be equivalent if all tests which detect fault f_1 also detect fault f_2 and vice versa .Two equivalent faults are, therefore, indistinguishable from a diagnostic viewpoint For example, a stuck-at-one fault on the output of a 2-input NAND

gate is equivalent to a stuck-at-zero on either input since the only vector which tests for a stuck-at-one at the output (a 11) also tests for a stuck-at-zero at either input. Moreover, the same vector (11) also happens to be the *only* test for a stuck-at-zero on either input Fault equivalence is transitive For example, if fault f_1 is equivalent to fault f_2 and fault f_2 is equivalent to fault f_3, then fault f_1 is also equivalent to fault f_3 When a number of faults are mutually equivalent to each other, the group (of faults) is said to form a *fault equivalence class.* Clearly, it is impossible to distinguish between faults in a given fault equivalence class based upon an analysis of faulty responses since all faults are detected on exactly the same vectors and primary outputs

From the discussion above it is clear that having large fault equivalence classes in a circuit is detrimental to fault diagnosis Determining all fault equivalence classes in a given circuit, however, is an NP-Complete problem [FujToi82]. Nevertheless, simple cases of fault equivalence can be analysed to come up with some basic guidelines on how to keep fault equivalence classes down to a minimum

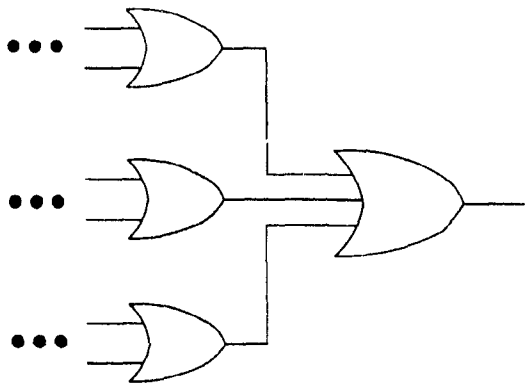


Figure 2.1 Circuit with large fault equivalence classes

The structure shown in figure 2.1 illustrates a circuit with a large fault equivalence class. The structure is a multi-level *fanout-free* region made up of similar non-inverting simple gates. Dominating logic value faults (stuck-at-ones) on the inputs of the OR gates are equivalent to stuck-at-one faults on their own outputs as well as stuck-at-one faults on the inputs of *subsequent* OR gates Due to the transitive property of fault equivalence, a stuck-at-one at the output of the structure is equivalent to a stuck-at-one fault on any other line of the circuit The final 3-input OR gate in the structure *adds* to the fault equivalence class since *all* of its input stuck-at-one faults are equivalent to the output stuck-at-one

The above observations serve as the basis for some general guidelines to limit the fault equivalence classes in a structure· fanout-free regions in a circuit should be kept to a minimum — adding fanout can only reduce fault equivalence, the number of levels of logic, both within fanout-free regions and in the circuit as a whole, should be minimised — again, the more the number of levels of logic, the more the *potential* for larger fault equivalence classes; if non-inverting gates are used, dissimilar gates should be used for successive levels of logic to prevent dominating logic value faults from *propagating* down the circuit, if inverting gates are used, similar gates are recommended for successive logic levels for the same reason, and, since multi-input gates imply a larger number of equivalent faults on their own inputs and output, their use should be minimised.

2.1.2 No Reconvergent Fanout

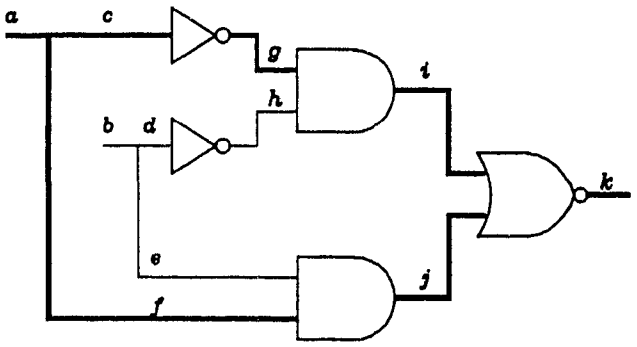


Figure 2.2 Reconvergent fanout in an XOR gate

Reconvergent fanout describes a structure in which more than one directed path may be traced from one line to another in the circuit [SchMet72] Figure 2.2 illustrates a circuit with reconvergent fanout — the circuit implements the two input XOR function Line a, like line b, is said to be a fanout *stem* while lines c, f, d and e are said to be fanout *branches* As can be seen from the figure, two different paths — $acgik$ and $afjk$ (shown with thick lines in the figure) — can be traced from fanout stem a to the output of the circuit Similarly, two separate paths can be traced from the other fanout stem, b, to the output of the circuit

There are two major problems associated with testing for faults associated with circuits with reconvergent fanout regions. Some faults — associated with the reconvergent fanout region itself — may not be testable at all while some stuck-open and transition

faults, if testable, may not be testable in a robust manner. A 2-input NAND gate with its inputs tied together provides an example of a a reconvergent fanout structure with untestable faults. Neither of the two inputs of the gate can be tested for stuck-at-one faults since such tests require complementary logic values on the inputs of the gate at the same time.

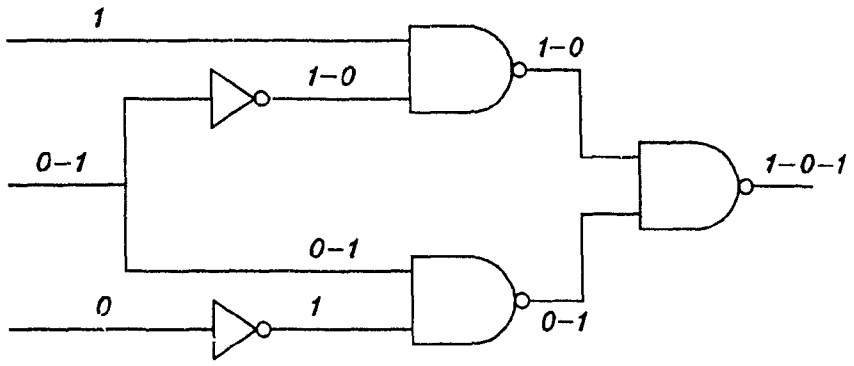


Figure 2.3 Hazards ir structure with reconvergent fanout

Figure 2 3 shows another structure with reconvergent fanout Due to the different number of logic levels present on the two different paths between the fanout stem and the output of the circuit, logic transitions on the fanout stem propagate to the inputs of the final 2-input NAND gate with different delays This leads to hazards within the circuit, due to which some transitions on the fanout stem — even with the other inputs held stable — can lead to *glitches* at the output The figure shows that with the other inputs held stable (at the values shown), a $0 \rightarrow 1$ transition on the fanout stem leads to a glitch at output before it stabilizes again Glitches are detrimental to testing for the twofold reason that they can preclude the generation of robust vectors for faults and their effects can *camouflage* faulty responses if they propagate to the output.

2.1.3 Robust Propagation of Faults

Testing strategies — like fault models — are based on certain fundamental assumptions One key assumption is often regarding the multiplicity of faults In particular, single fault models — like the single stuck-at fault model — assume that faulty circuits contain only single faults Since the object of the present work is to establish the nature of such faults, any presumptions regarding their type and multiplicity are clearly unjustifiable In fact, the approach taken to study the faults is required to be as general as possible

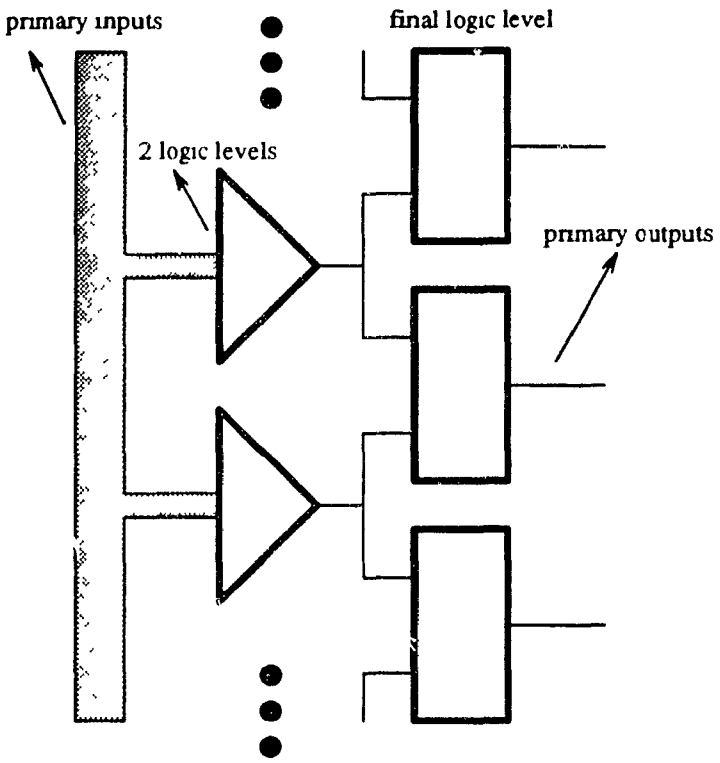


Figure 2.4 Structure supporting a robust propagation of faults

In keeping with the above strategy, the design of the test chip is required to be such that faults may be detected and analysed in the presence of other faults The structure of the chip, therefore, should be designed in a manner such that most faults can propagate to more than one primary output While faults associated with lines in the final level of logic can only propagate to single outputs, suitably designed fanout logic can ensure that all faults, other than those associated with the final level of logic, can propagate to, and be detected on, at least two different primary outputs Figure 2 4 illustrates the idea As can be seen in the figure, all faults associated with the first two levels of logic in the structure can be detected on both primary outputs In the event of a fault, associated with the final level of logic, preventing detection of another fault, it (the latter fault) can still be detected and analysed on the basis of observations made on another output The structure, in this case, is said to support the *robust propagation* of faults

2.1.4 Other Considerations

As was pointed out in the last section, the approach taken to study faults needs to be as general as possible to account for any eventuality One such eventuality could be the

requirement for the application of an exhaustive test to the circuit. The provision for such a requirement has a direct bearing on the total number of input pins in the circuit since the number of vectors in an exhaustive test set, and consequer ·ly their application time, grows exponentially with them. To keep the application time of an exhaustive test set within "reasonable" limits, there should not be a large number of input pins in the circuit

For an accurate representation of the class of circuits being studied, the test chip is expected to duplicate a "realistic environment". The factors affecting the realism of the circuit are its size, — in terms of its gate count, transistor count and silicon area — its functional capability, the average fanout associated with internal lines of the circuit, the number of input/output pins and, in a semi-custom design, the selection of standard cells used. Clearly, the test chip should be "similar" to a real chip with respect to all the properties enumerated above.

2.2 Characterisation of the Test Chip

The nature and frequency of physical defects in devices are intimately related to their layouts [ShMaFe85] Since different cell libraries stand to contain differences in their layouts, even for identical logic blocks, typical defects occurring on devices are characteristic of the cell libraries used in their design Automated design tools — used typically for routing and layout — similarly play a large part in characterising the designs for which they are employed

The fabrication process used dictates, to a very large extent, the defectivity of the devices fabricated with it While "young" processes often produce devices with catas trophic failures resulting from mask misalignments, they can be "tuned" to minimise such gross defects Mature processes, on the other hand, frequently produce defects which are more "local" in nature but very characteristic of the fabrication process itself The characterisation is reflected both in the nature and frequency of physical defects

While the test chip is designed to be representative of a certain class of circuits, it is characterised by the cell libraries, design tools and fabrication process used in its design and production. The experimental results obtained from its use as a test vehicle, therefore, remain valid only within the confines of the above parameters

2.3 The Design and Structure of the Test Chip

The design of the test chip follows a structured hierarchy At the top level, the design is specified in terms of logic *blocks* Each block, in turn, is a collection of *modules* The basic building blocks of the modules, at the lowest level of the design hierarchy, are gates, buffers, tristate drivers and pad cells, all taken from a number of different cell libraries. This section presents a detailed description of the structure and design of the test chip

2.3.1 Top Level Structure

Figure 2 5 illustrates the top level structure of the chip. The chip can be seen as a collection of 10 independently accessible logic blocks The logic blocks share a set of 16 primary inputs and 16 primary outputs Any block can be functionally *selected* at a given time by flagging its corresponding *select* signal The *select* signal for each block controls a set of 16 tristatable buffers on its output lines

In addition to the logic blocks shown in figure 2.5, the test chip contains pull-up/-pull-down resistors on each primary output. Outputs 0–7 contain pull-down resistors — from the output to ground — while outputs 8–15 contain pull-up resistors — from output to V_{dd} When *all* blocks are simultaneously de-selected, these resistors force the outputs into default logic states This property of the outputs — to go into default logic states when not driven — is used to advantage to generate tests to detect faults associated with the tristate drivers as discussed later in chapter 3

2.3.2 The Structure of Blocks

Out of the 10 logic blocks present in the chip, 9, labelled DS1 through DS5 and DZ1 through DZ4, contain combinational logic to implement certain boolean functions The tenth logic block, DEMPTY, contains no logic It merely connects each output to each corresponding input.

While unacceptably large circuit delays can be fairly "local" phenomena, conforming to the gate delay model and restricted to a small section of the circuit, they can equally well be "global" in nature, resulting in an inherently "slow" device or even a slow wafer due to problems associated with the fabrication process. Since the DEMPTY block does

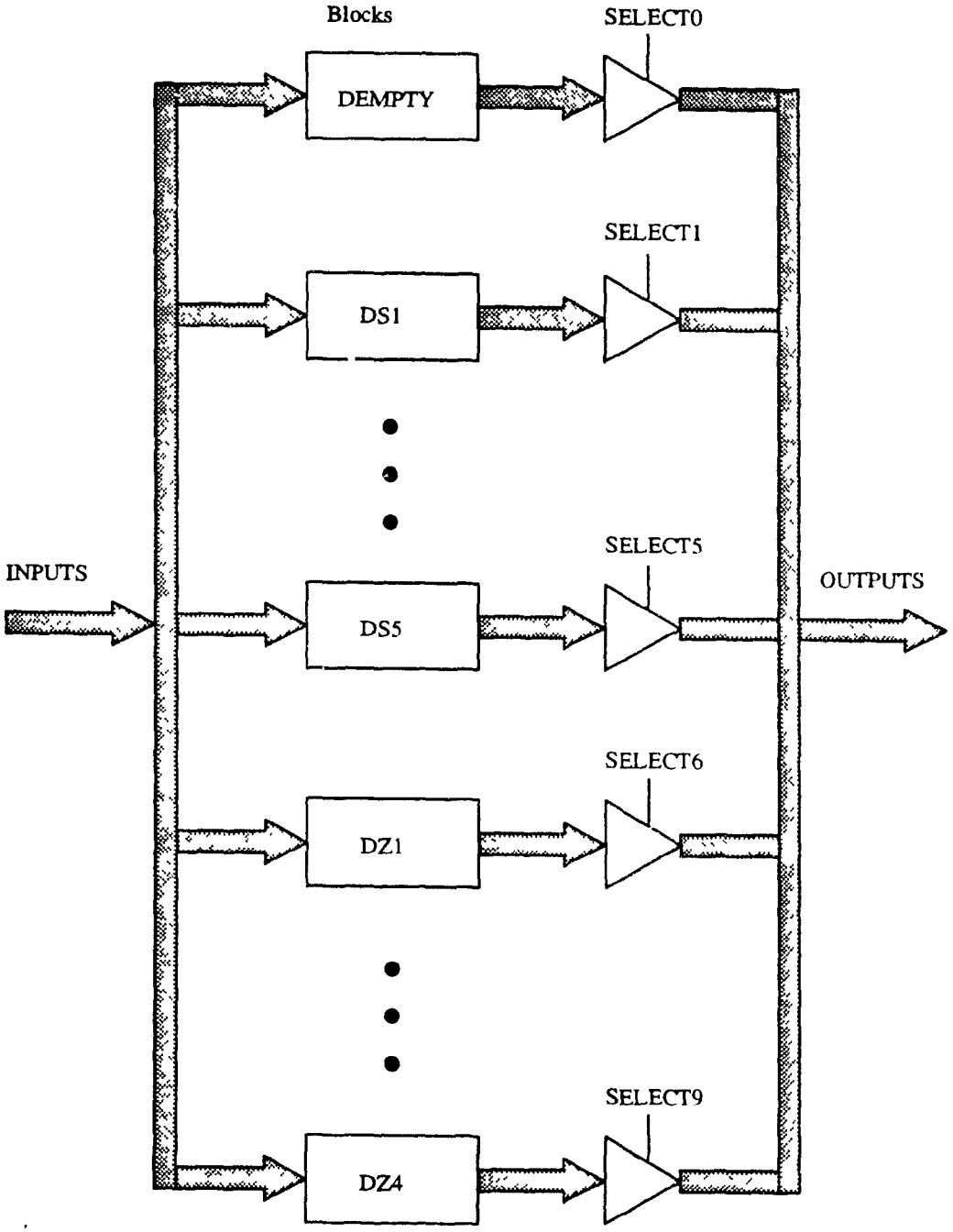


Figure 2.5 Structure of the test chip

not contain any logic, it serves as a test block to get a measure of path delays associated with the logic in its neighbourhood Devices with unacceptably high path delays can, therefore, be easily identified and subjected to further analysis to establish the cause of such problems. Similarly, "normal" devices with acceptable delays can be characterised for

their timing performance.

Figure 2 6 illustrates the structure of each of the other 9 combinational logic blocks Each block has 16 primary inputs and 16 primary outputs The blocks are identical in structure, differing only in their functionality. A collection of 8 different modules — $DXOA$, $DXAO$, $DAXO$, $DOXA$, $DOOO$, $DAAA$, $DOAX$ and $DAOX$ — is used in the design of each block Each module has 16 inputs, 4 outputs and contains 2 levels of logic made up of 2-input NANDs, NORs and XORs The (4) outputs from each module fan out into 2 branches, with each branch feeding a 4-input gate The 4 input gates used are NANDs and NORs for logic blocks $DS1$ through $DS5$ For the 4 DZ blocks, a number of complex gates, implementing the AND-OR-INVERT and OR-AND-INVERT functions are used in addition to the simple 4-input gates The 4-input gates serve as the final level of logic in each block From the structure, it is clear that all faults associated with the macro units (upto and including the second level of logic in the structure of the chip) can propagate to, and be detected on, two separate outputs

The difference between any two successive blocks in the same group — for example between blocks $DZ2$ and $DZ3$ — lies in the *relative positions* of the modules. The modules are *rotated*, relative to the other logic, for each successive block in a manner analogous to an arithmetic rotate operation Figure 2 7 illustrates the relation between two successive blocks in the same group

2.3.3 The Structure of Modules

As described earlier, each module contains 16 inputs and 4 outputs The logic in each module is made up of four *copies* of a 2-level *macro*, each with 4 inputs and 1 output. Each macro contains a combination of 3 2-input gates. The gates used are NANDs, NORs and XORs Figure 2 8 shows the eight different macros used in the design of the chip

Overall, there are 64 2-input gates in the first level of logic, 32 2-input gates in the second level and 16 4-input gates in the third level of logic in each block (other than the DEMPTY) in the circuit The gates used in the circuit are taken from two different CMOS standard cell libraries A total of 12 different standard cells are used in the design of the logic. Each block in the circuit contains approximately 750 transistors The test chip, as a whole, contains about 7500 transistors

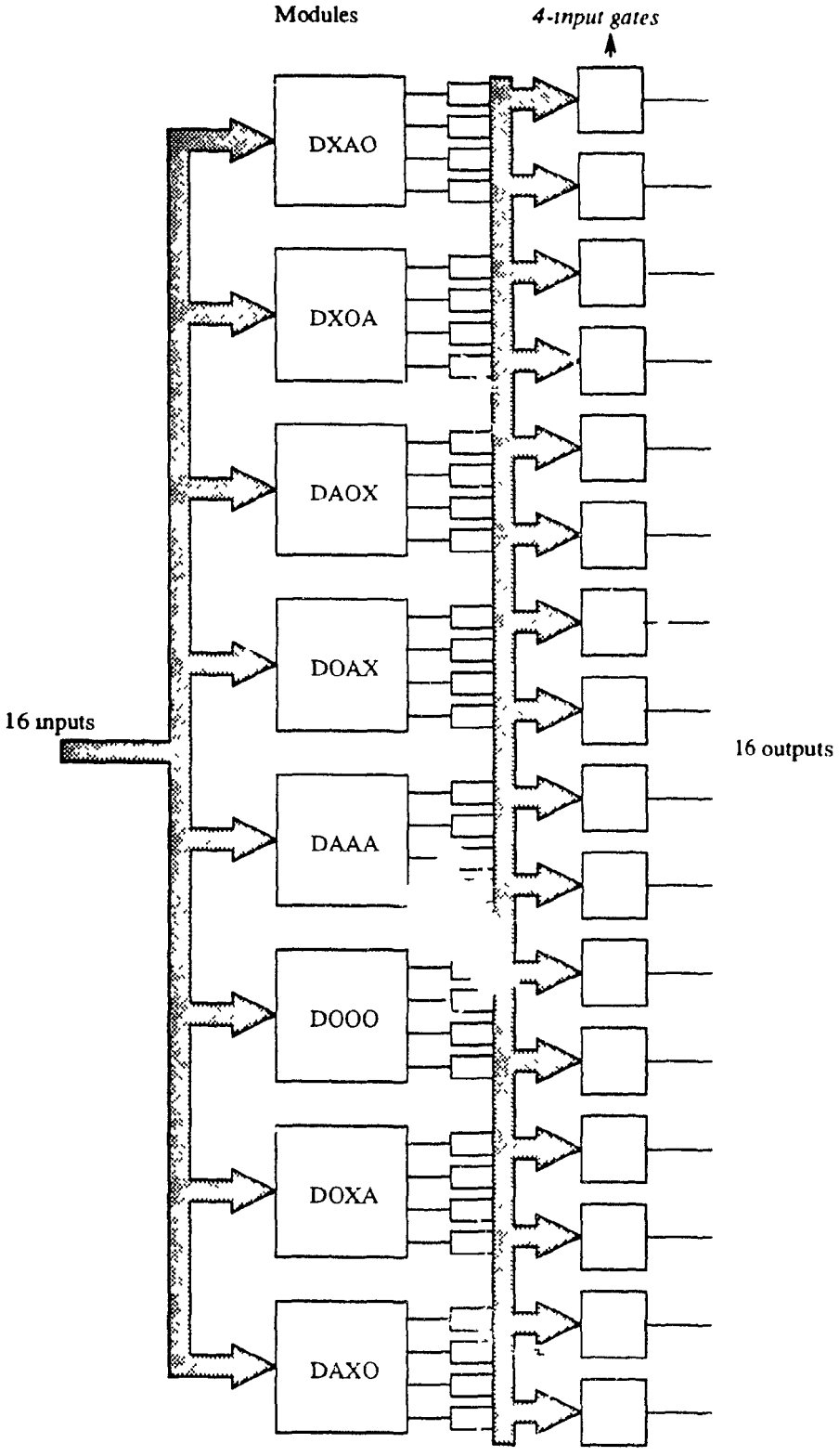


Figure 2.6 Structure of the logic blocks

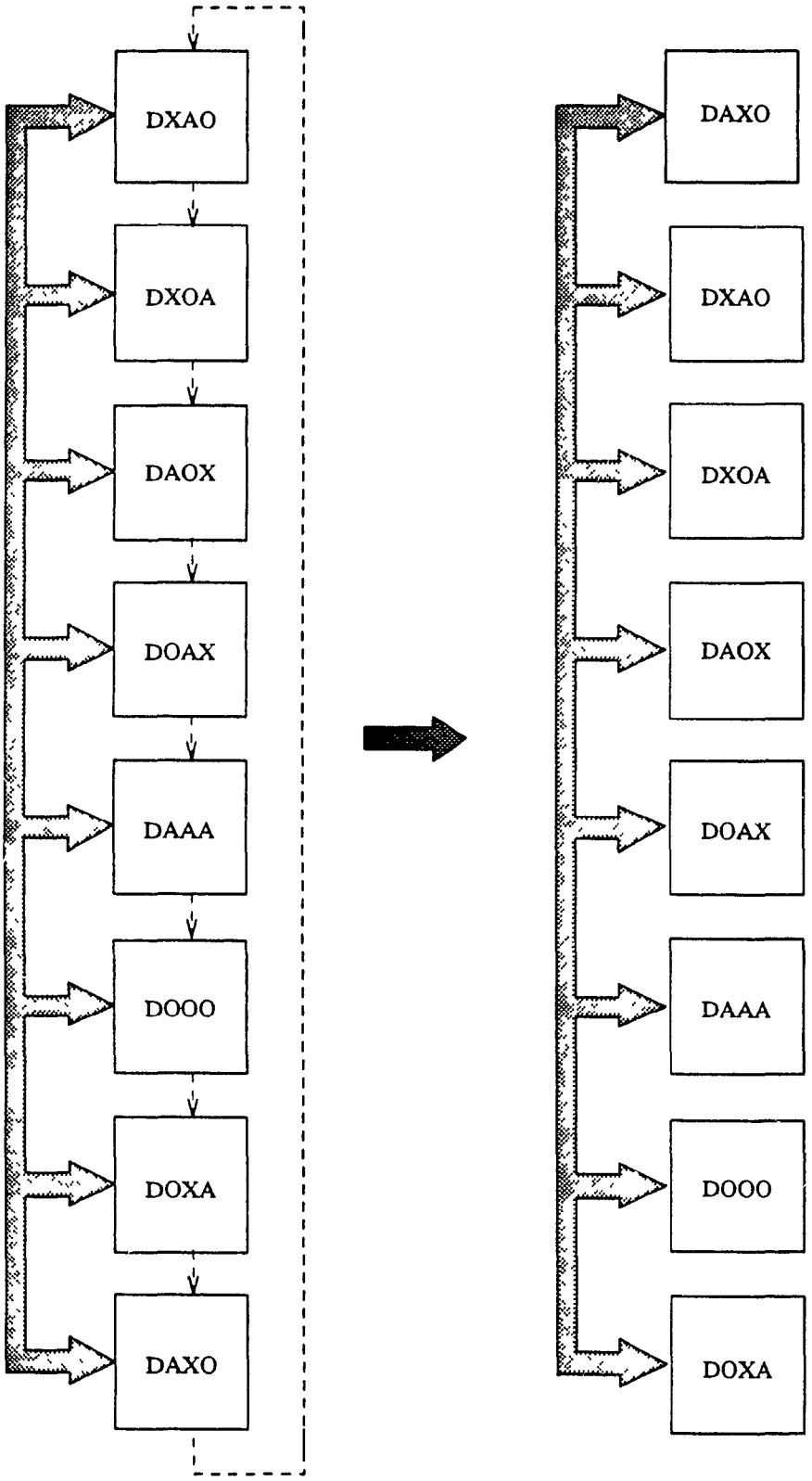


Figure 2.7 Relation between two successive blocks

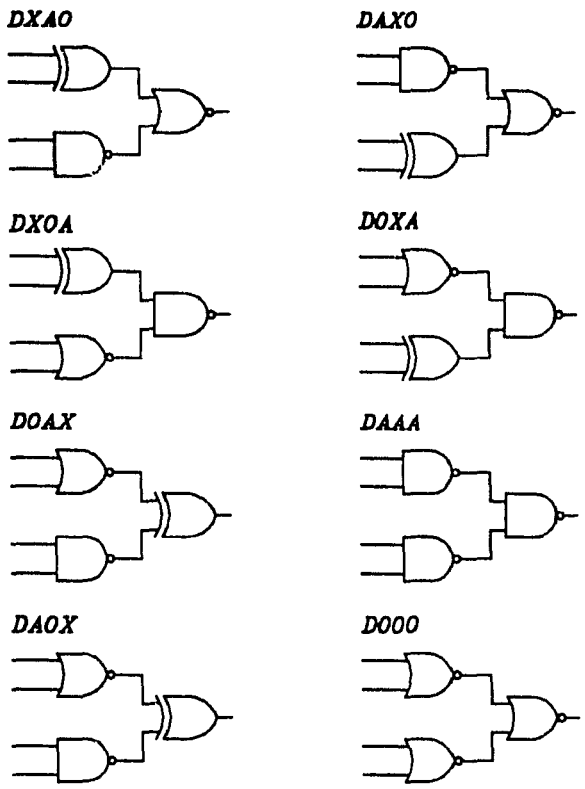


Figure 2.8 Structures of the macros

An analysis of the design of the test chip will reveal that not all the recommendations made in section 2.1 were rigorously implemented. In particular, 4-input gates are used in the design of all logic blocks (except the DEMPTY block) and some macros use dissimilar inverting 2-input simple gates in successive logic stages While this *violation* of guidelines increases the fault equivalence groups to an extent, it makes the design more "realistic" In the final analysis, a delicate balance is maintained between design features supporting ease of diagnosis and others which offer more realism at the cost of some diagnostic properties

Chapter 3 The Test Set

The fundamental requirement of the test set, in the context of the thesis, is to provide an insight into the failures occurring within the test circuit, and to do so in a robust and fault-tolerant manner — in the presence of other failures.

A complete coverage of faults under a number of different fault models is required to maximise the coverage of actual physical failures within the circuit. The fault models considered are.

- Single and multiple stuck-ats,
- Single and multiple delays, and
- Single and multiple stuck-opens

In addition, a good coverage of *shorts* and *bridging* faults is considered essential

In order to provide an insight into actual physical failures in the circuit, the test set is required to support good diagnostic resolution. In particular, it is essential to distinguish between faults — for distinguishable faults under the same fault model as well as for distinguishable faults under different fault models.

Finally, the test set is required to be robust in nature so as to be effective in the presence of multiple faults, as much as possible. More specifically, the test set is required to provide for:

- A robust *initialisation* — so as to be able to initialise internal nodes of the circuit in the presence of other faults;
- A robust *sensitisation* — so as to be able to sensitise faults in the presence of other faults, and

- A robust *propagation* — so as to allow for the detection of faults in the presence of other faults by propagating them to all outputs possible.

Because of the unique requirements of the test set, lines (gates) and switches in the circuit are treated separately for generating appropriate tests The methodology for generating such tests is discussed in the following 2 sections

3.1 Gate-Level Analysis

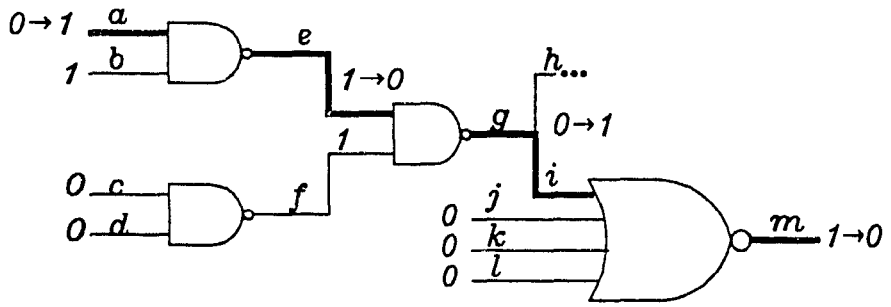


Figure 3.1 Transition propagation on test circuit

Figure 3.1 illustrates a sub structure taken from the test circuit. Line g is a fanout stem with 2 fanout branches h and i. m is a primary output while lines a, b, c and d are the primary inputs of the circuit

Inputs c and d are set to logic 0 while input b is set to logic 1 Other inputs of the circuit, not shown in the sub-structure, are set so as to force 0s on each of the lines j, k and l; note that it is possible to independently control all input values of any gate in the test structure since there are no reconverging paths If, under these conditions, a $0 \rightarrow 1$ transition is applied on input a, the transition will propagate to output m (in a fault-free circuit) along the path $aegim$ If, in response to such a transition on input a, the corresponding $1 \rightarrow 0$ transition is actually observed on output m — given reasonable time for the input transition to propagate to the output — the following conclusions may be drawn about the status of the circuit based on the unconditional fault testing approach, presented in [CoxRaj88a]

- There can be no single or multiple stuck-at faults of any combination or multiplicity along the path $aegim$ since their presence would preclude the occurance of the transition on m. The faults on the path are, therefore, tested unconditionally

- There can be no single or multiple delay faults of one *set* of polarities (corresponding to the input transition) of any combination or multiplicity along the path $aegim$ since their presence would preclude the occurrence of the observed transition on output m within a reasonable amount of time Furthermore, by propagating input transitions of both polarities — the vector trio $0 \rightarrow 1 \rightarrow 0$ serves the purpose for the case at hand — *all* delay faults of all multiplicities along the given path may be covered. Both *line* and *gate* delays are accounted for.
- The observed transition at the output is a consequence of the transition on input a only and could not have occurred as a result of any *hazard* conditions set up in the circuit Hazards are ruled out since the circuit has no reconvergent paths and all other inputs (other than a) are held stable during the transition.

The above analysis illustrates how complete, robust test sets can be effectively generated for multiple stuck-at and delay faults (both gate and path) for suitable structures In the example, the single vector trio $0 \rightarrow 1 \rightarrow 0$ unconditionally covers all stuck-at, transition and delay faults along the path $aegim$ without explicit enumeration Transformations have previously been formulated for modelling stuck-open faults in fully-complementary MOS structures on the basis of transition faults [CoxRaj88b]

The method of test generation covers structures using XOR gates as well by ensuring the propagation of transitions separately through *both* internal paths. For a given input transition, the polarity of the output transition depends on the internal path taken through the XOR

The example also illustrates how a degree of failure tolerance is incorporated into the test set While a logic 0 on either of primary inputs c or d satisfactorily initialises line f to help propagate the transition, *both* c *and* d are held at logic 0 to allow for a failure tolerant propagation

3.2 Switch Level Analysis

Consider the 2 input CMOS NAND gate shown in figure 3.2 with inputs a and b and output c.

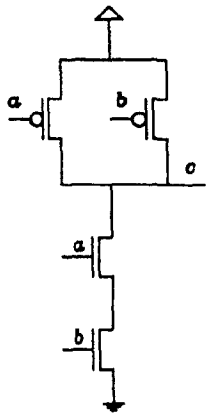


Figure 3.2 2-input NAND gate

a	b	c	stuck-at faults	stuck-open faults
1	1	0	$\{a_0, b_0, c_1\}$	–
0	1	1	$\{a_1, c_0\}$	$\{a_p\}$
1	1	0	$\{a_0, b_0, c_1\}$	$\{a_n, b_n\}$
1	0	1	$\{b_1, c_0\}$	$\{b_p\}$
0	1	1	$\{a_1, c_0\}$	$\{\}$
1	1	0	$\{a_0, b_0, c_1\}$	$\{a_n, b_n\}$
1	0	1	$\{b_1, c_0\}$	$\{b_p\}$
1	1	0	$\{a_0, b_0, c_1\}$	$\{a_n, b_n\}$

Table 3.1 Complete Test Set for stuck-at and transition faults for 2 input NAND gate

A complete test set for single stuck-at and stuck-open faults within the gate is generated by using a sequence of 8 vectors as shown in the first two columns of table 3.1 The stuck-at faults detected by each vector are given in column 4 of the table. For example, the first vector $ab = 11$ detects the faults a stuck-at 0, b stuck-at 0 and c stuck-at 1 ($\{a_0, b_0, c_1\}$) In addition, 6 of the 7 possible vector *pairs* detect stuck-open faults associated with the gate The stuck-open fault coverage of each vector pair is shown in column 5 of the table Each entry corresponds to the stuck-open fault(s) detected for the combination of the current and *previous* vectors For example, the first vector pair (vectors 1 and 2) detects a stuck-open fault on the a pull-up transistor (a_p) The test set shown in the table is not minimal. For example the vector pairs $ab = 01, 11$ and $ab = 10, 11$ cover the same stuck-open faults ($\{a_n, b_n\}$). The redundancy is justified by

the fact that, while the two *stuck-open* faults associated with the pull-down network are equivalent, the two vector pairs can lead to different *delays* observed at the output due to *transition* faults associated with the two transistors. In addition, certain vector pairs (like $ab = 01, 11$) appear twice in the test set. This is because one of their occurances is only incidental to the sequence of the test set While the test set should really be seen as 4 pairs of non-overlapping vectors with each pair testing a transistor (or branch) of the gate, certain stuck-open faults are detected because of the way the vector pairs are *stacked* These vector pairs are repeated to simplify the analysis presented in this section Note also that the vector pair $ab = 10, 01$ is not free from static hazards.

To illustrate the diagnostic resolution offered by this test set, let us assume that the b pull-up transistor is stuck-open The responses expected from the faulty (fy) circuit upon application of the test set are shown in table 3.2 along with the fault-free (ff) responses Minus (-) signs beside faulty output values indicate a *weak* state where the output is not driven (because of the fault) but retains charge from its *previous* state.

a	b	c(ff)	c(fy)	sfl(s-a)	sfl(s-o)
1	1	0	0	$\{\cancel{a_0}, \cancel{b_0}, \cancel{c_1}, a_1, b_1, c_0\}$	$\{a_p, a_n, b_p, b_n\}$
0	1	1	1	$\{\cancel{a_1}, b_1, c_0\}$	$\{\cancel{a_p}, a_n, b_p, b_n\}$
1	1	0	0	$\{b_1\}$	$\{\cancel{a_n}, b_p, \cancel{b_n}\}$
1	0	1	0-	$\{b_1\}$	$\{b_p\}$
0	1	1	1	$\{b_1\}$	$\{b_p\}$
1	1	0	0	$\{b_1\}$	$\{b_p\}$
1	0	1	0-	$\{b_1\}$	$\{b_p\}$
1	1	0	0	$\{b_1\}$	$\{b_p\}$

Table 3.2 Fault Diagnosis on the NAND gate

Based on the results of the test, diagnosis is performed, where the faults detected by vectors (vector-pairs) with *good* responses are eliminated from a *suspect fault-list* (sfl) initially containing all faults. The diagnosis operations on suspect fault-lists are shown for each vector and vector pair in columns 5 and 6 respectively For instance, since the response to the first vector is good, all stuck-at faults detected by it ($\{a_0, b_0, c_1\}$) are removed from the stuck-at fault suspect list as shown in column 5 of the first row Similarly, since the response to the first vector pair (vectors one and two) is also good, the stuck-open fault

detected by it ($\{a_p\}$) is removed from the stuck-open fault suspect list The complete suspect fault list left at the end of the diagnosis operation contains the faults $\{b_1, b_p\}$

Hence, a complete test set guarantees that all targeted faults are detected but does not necessarily provide the required resolution in terms of fault diagnosis

a	b	c	stuck-at faults	stuck-open faults
1	1	0	$\{a_0, b_0, c_1\}$	–
0	1	1	$\{a_1, c_0\}$	$\{a_p\}$
1	1	0	$\{a_0, b_0, c_1\}$	$\{a_n, b_n\}$
1	0	1	$\{b_1, c_0\}$	$\{b_p\}$
0	0	1	$\{c_0\}$	$\{\}$
0	1	1	$\{a_1, c_0\}$	$\{\}$
1	1	0	$\{a_0, b_0, c_1\}$	$\{a_n, b_n\}$
0	0	1	$\{c_0\}$	$\{\}$
1	0	1	$\{b_1, c_0\}$	$\{\}$
1	1	0	$\{a_0, b_0, c_1\}$	$\{a_n, b_n\}$

Table 3.3 Diagnostic Test Set for stuck-at and transition faults for 2 input NAND gate

A modified test set for the 2-input NAND gate is given in table 3.3 along with the fault-free responses and faults covered for each vector/vector-pair The only change from the test set in table 3 1 is that the last two vector pairs testing the pull-down series transistor network are each preceded by the vector $ab = 00$

Once again, we assume a faulty circuit with the b pull-up transistor stuck-open The fault diagnosis operations based on the results of the modified test set are illustrated in table 3 4 The pruned fault list left at the end of the diagnosis operation contains the single fault b_p — the correct one The modified test set, therefore, offers superior diagnostic resolution.

The basic ideas used to generate the diagnostic test for the 2-input NAND gate can be generalised to generate tests for other simple CMOS gates In addition to the traditional approach of generating tests for stuck-open faults [Wadsac78] by testing each transistor in the structure by

a	b	c(ff)	c(fy)	sfl(s-a)	sfl(s-o)
1	1	0	0	$\{\not{a}_0, \not{b}_0, \not{c}_1, a_1, b_1, c_0\}$	$\{a_p, a_n, b_p, b_n\}$
0	1	1	1	$\{\not{a}_1, b_1, \not{c}_0\}$	$\{\not{a}_p, a_n, b_p, b_n\}$
1	1	0	0	$\{b_1\}$	$\{\not{a}_n, b_p, \not{b}_n\}$
1	0	1	0–	$\{b_1\}$	$\{b_p\}$
0	0	1	1	$\{b_1\}$	$\{b_p\}$
0	1	1	1	$\{b_1\}$	$\{b_p\}$
1	1	0	0	$\{b_1\}$	$\{b_p\}$
0	0	1	1	$\{b_1\}$	$\{b_p\}$
1	0	1	1–	$\{\not{b}_1\}$	$\{b_p\}$
1	1	0	0	$\{\}$	$\{b_p\}$

Table 3.4 Fault Diagnosis on the NAND gate for modified test set

- Initialising the output to a value the complement of which the transistor is expected to drive it to: and
- Turning the transistor ON to try and toggle the output value,

the following guidelines are presented for generating robust, diagnostic tests.

- Choose the initialising vector such that it differs from the test vector in only one bit position.
- In case the initialisation can be performed by any one of a group of transistors in parallel, choose any transistor to perform the initialisation, and precede the initialising vector with a robust initialisation using all transistors in the parallel group simultaneously.

3.2.1 Complex Gates

Formal methods exist to transform fully complementary CMOS transistor networks into equivalent logic circuits, consisting of AND, OR and NOT gates, such that tests for stuck-at faults in the equivalent circuit can be used to detect line stuck-at and stuck-open faults in the modeled CMOS circuit [ReAgJa84][JaiAgr85] For the case at hand, however, the guidelines used to generate diagnostic test sets for simple gates are extended to cover

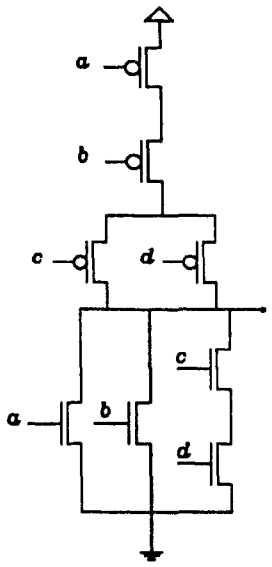


Figure 3.3 1-1-2 AND-OR-INVERT gate

fully complementary implementations of complex gates used in the test circuit because of the special requirements of diagnostic resolution and failure tolerance in the test set

Consider the $1-1-2$ AND-OR-INVERT gate shown in figure 3 3 A test sequence may be generated for the pull-down a transistor by initializing the output node to a logic 1 by using any of 3 vectors — $abcd = 0001,\ 0010\ or\ 0000$ — and then switching the transistor ON To make the test robust, however, the last initializing vector ($abcd = 0000$) is preferred since it guarantees initialization of the node even in the presence of a stuck-open fault on either of pull-up transistors c or d. To generalize the point, the following guideline is appended to the set of guidelines described in the last section

- In case an initialization can be performed by using any one of a number of parallel transistors in a network use *all* of them, provided the test vector does not differ from the initializing vector by more than one bit

Using the guidelines, a test can be generated for the pull-up a transistor by performing a robust initialisation followed by a normal initialisation and the actual test The vector trio $abcd = 1111, 1000, 0000$ serves the purpose Note that both parallel pull-up transistors c and d are switched ON during the actual test so that a failure on either one of them still allows for the detection of faults associated with the transistor under test (the pull-up a transistor). The pull-down a transistor is tested by the vector pair $abcd = 0000, 1000$

The same strategy (used for the a transistors) may be applied to test the pull-up and pull-down b transistors. The network of transistors controlled by inputs c and d, which are still to be tested, forms a 2-input NAND structure which can be *accessed* for testing by keeping $ab = 00$. A point to note, however, is that the pull-down a or b transistors can still be used to provide for a robust initialisation when testing the pull-up c and d transistors. For example, the pull-up c transistor is tested by the trio $abcd = 1111, 0011, 0001$ instead of the pair $abcd = 0011, 0001$ so as to be able to detect a fault in the pull-up c transistor even in the presence of faults in one of the pull-down branches The pull-down c transistor is tested by the trio $abcd = 0000, 0001, 0011$. Tests are similarly generated for the pull-up and pull-down d transistors Table 3.5 shows the complete test for each transistor in the 1-1-2 AND-OR-INVERT gate.

Tests for other, non-XOR complex gates are based on the same guidelines and are included in Appendix A

3.2.2 2-input XOR

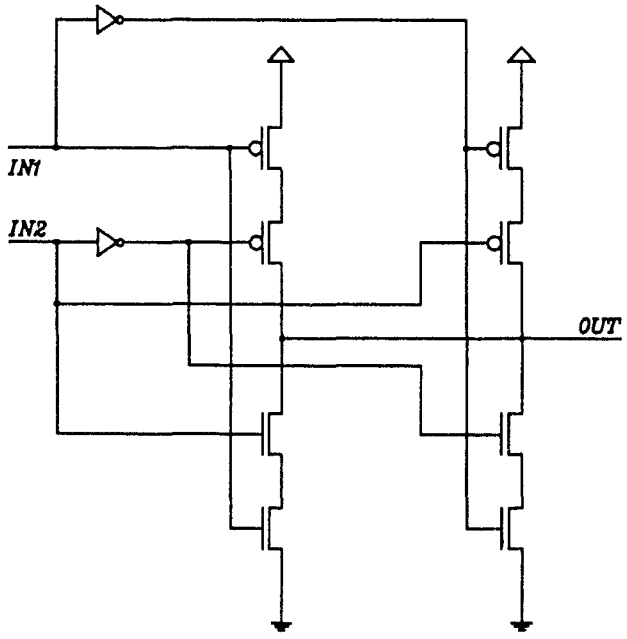


Figure 3.4 2-Input XOR

The transistor level structure of a 2-input CMOS XOR gate is shown in figure 3 4 The ideas used in generating tests for simple and complex gates cannot be used for the

Stuck-Open Test for Transistor		Inputs			
		a	b	c	d
Pullup	a	1	1	1	1
		1	0	0	0
		0	0	0	0
	b	1	1	1	1
		0	1	0	0
		0	0	0	0
	c	1	1	1	1
		0	0	1	1
		0	0	0	1
	d	1	1	1	1
		0	0	1	1
		0	0	1	0
Pulldown	a	0	0	0	0
		1	0	0	0
	b	0	0	0	0
		0	1	0	0
	c	0	0	0	0
		0	0	0	1
		0	0	1	1
	d	0	0	0	0
		0	0	1	0
		0	0	1	1

Table 3.5 Diagnostic test set for 1-1-2 AND-OR-INVERT

XOR since each of the 4 *branches* in its "H" structure are tightly coupled and cannot be independently controlled

The test set for the XOR is generated by testing each transistor in the "H" structure and doing so twice — using both possible branches for initialisation For instance, the vector pair $ab = 11,01$ tests the pull-up a transistor The test is then repeated by using the vector pair $ab = 00,01$ to use the other pull-down branch for initialisation Other transistors in the structure are similarly tested Properties of the test set thus generated

are that it is robust, since vector pairs differ in only a single bit position, and each transistor can be tested, even if there are faults in any one of the two transistor branches which may be used to initialise it

Transistors within the two inverters in the XOR cannot be independently tested because of internal reconvergence within the gate They are, however, implicitly covered by the test described above. Table 3 6 shows the complete test for stuck-at and stuck-open faults for the 2-input XOR gate generated on the basis of the discussion above

a	1	1	1	0	0	1	0	0	0	1	0	0	1	1	1	0
b	1	0	1	1	0	0	0	1	1	1	1	0	0	1	0	0

Table 3.6 Test sequence for 2-input XOR gate

3.3 Complete Test Set Generation

Complete test sets for each logic block are generated by *stacking* tests for each gate within the block Each test vector, for a given gate, is *justified* back to the primary inputs of the block and necessary assignments made to the remaining primary inputs so as to *propagate* the logic value at the output of the gate being tested to a primary output Note that there can be no conflicting assignments to primary inputs due to different sensitization and propagation requirements since the logic blocks are organized as tree structures with no reconvergent fanout In addition, each gate within the first two levels of logic in each block is tested twice, under different conditions of the circuit, so that faults associated with it propagate to, and are detected on *both* possible primary outputs

The gate-level tests devised earlier all contain at least one transition (of both polarities) going from each input (of the gate) to its output When the gates in the first level of logic in the circuits are tested with these test sets, the rest of the circuit is initialized to propagate their responses to a primary output As described previously, however, each gate in the first two levels of logic is tested twice, under different propagation conditions, so as to propagate its responses to both primary outputs possible The complete test set for each block, therefore, contains transitions of both polarities propagating from each input of the first level of logic to each primary output possible, thereby satisfying the requirements of the test set from the point of view of the structural analysis presented in section 3.1

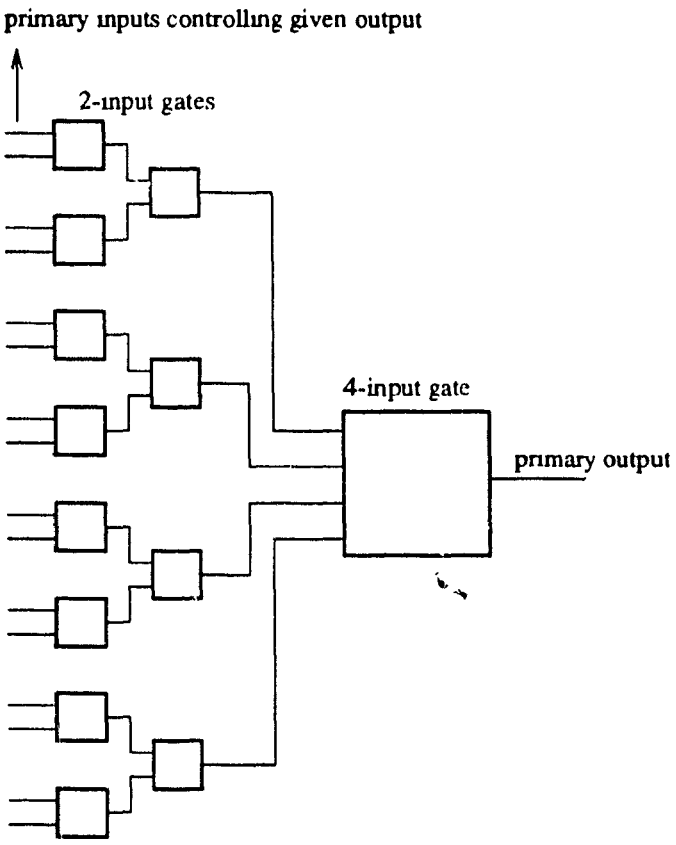


Figure 3.5 Structure of the logic driving each primary output in a logic block

Moreover, by propagating transitions through all paths of the circuit, keeping the nodes in the rest of the circuit static, a significant coverage of bridging faults is achieved

The actual test generation for each block is done in a hierarchical fashion, taking advantage of the regular structure of the logic blocks. Figure 3 5 illustrates the structure of the logic *driving* each primary output Each collection of 3 2-input gates, termed macro, drives an input of a 4-input gate which drives the primary output Three sets of vector *modules* are defined for each macro the first containing concatenated tests for each of the 3 gates within it, the second containing the necessary assignments at the inputs (of the macro) required to get a logic 0 at its output and, the third containing the necessary assignments at the inputs (of the macro) required to get a logic 1 at its output The test of the structure (behind each primary output) is then specified at a higher level in terms of the test of each of the 4 macros driving the 4-input gate and the test of the gate itself Each macro driving the 4-input gate is tested in sequence, using the vector modules defined earlier, by holding the other 3 inputs to the gate to non-controlling values The necessary

assignments to the relevant primary inputs to hold the other 3 inputs to the 4-input gate to non-controlling values are easily made by *looking up* the vector module, for each macro, which results in an output of the desired logic value. The 4-input gate is then tested using its corresponding diagnostic test sequence Again, the test inputs to the gate are backward justified simply by looking up the vector module entries corresponding to the driving macros and their required logic values

The test set for each block is completely specified in terms of a higher level representation. The file containing the specification is then processed automatically by a program which expands the high-level test specification into test vectors, in terms of logic ones and zeros, by reading the appropriate vector modules. The format of the high-level specification is illustrated in table 3 7

The table shows the specification for a structure using a 4-input NOR gate at the output A non-controlling value for the gate is a logic 0 The group of first four *vectors* in the table specifies the test for each of the 4 macros in the structure The header specifies the macros themselves (DXAO, DAXO, DOXA and DAAA) A "tr" entry for a macro implies its test while the outputs of the other macros are held at static non-controlling values for the duration of the test A "s0" specifies a required logic 0 at the output of a macro — the required non-controlling value — while a "s1" specifies a required logic 1 As can be seen from the first four entries in the table, each of the macros is tested in sequence while the other macros drive the 4-input NOR gate with non-dominating values to allow for the propagation of faults within the macro being tested The next set of 20 entries specifies the test for the 4-input NOR gate The test is specified simply in terms of the values required at the outputs of each of the 4 driving macros

The vector modules for each macro are defined manually The pre-defined test for each gate in a macro is backward justified and its response propagated to the output of the macro in a failure-tolerant manner For example, if a logic 1 is required at the output of a NAND gate, *both* inputs of the gate are held at logic 0s so that the required conditions for sensitization/propagation are correctly set up even in the presence of a stuck-at-one fault on one of the inputs

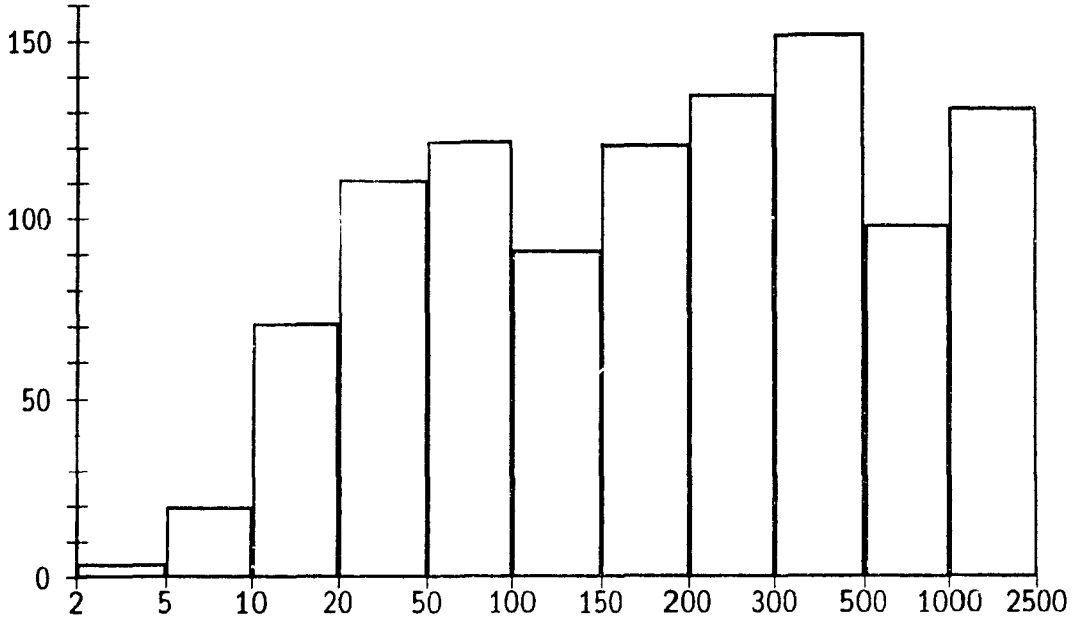
3.4 Characterisation of the Test Set

The test sets generated for each logic block, according to the methodology presented in this chapter, are unusually long because of their qualities of robustness in initialization,

dxao	daxo	doxa	daaa
tr	s0	s0	s0
s0	tr	s0	s0
s0	s0	tr	s0
s0	s0	s0	tr
dxao	daxo	doxa	daaa
s0	s0	s0	s0
s1	s0	s0	s0
s1	s1	s1	s1
s1	s0	s0	s0
s0	s0	s0	s0
s0	s0	s0	s0
s0	s0	s1	s1
s0	s1	s1	s1
s0	s1	s1	s1
s0	s0	s1	s1
s0	s0	s0	s0
s0	s1	s0	s1
s0	s1	s1	s1
s0	s1	s1	s1
s0	s1	s0	s1
s0	s0	s0	s0
s0	s1	s1	s0
s0	s1	s1	s1
s0	s1	s1	s1
s0	s1	s1	s0

Table 3.7 High-level test specification for a structure with a 4-input NOR gate at the output

sensitization and propagation, diagnostic resolution and completeness across several different fault models The test length for each of the DS blocks is 2528 while that for each of the DZ blocks is 2544 The test sets for each logic block are subsequently fault-simulated for stuck-at faults A gate level description (of each block) is used as an input to the simulator. The complex gates in the blocks — 3 implementations of the 4-input AND-OR-



X axis Detection counts of faults

Y axis Number of faults

Note: Faults lying exactly on the boundaries of intervals are placed within boxes to their left

Figure 3.6 Detectability profile of single stuck-at faults with the robust test set

INVERT function, 3 implementations of the 4-input OR-AND-INVERT function and the 2-input XOR gate — are represented as equivalent simple-gate networks using AND, OR and NOT gates The results of the fault simulation confirm complete coverage of single stuck-at faults for each block Moreover, each stuck-at fault in the circuit is covered many times Figure 3 6 shows the distribution of single stuck-at faults against their *detection count* [2] with the diagnostic test set for one of the blocks To get another measure of the stuck-at fault detection redundancy in the test set, it was found that complete stuck-at coverage of the DS1 block was achieved with only 63 randomly generated vectors

Once complete coverage of stuck-at faults in each of the blocks is established using an equivalent gate-level description of all complex gates, the coverage of stuck-open faults in the circuit may be determined Since the algorithm followed to generate the test set propagates transitions through each line in the circuit at least once (including all lines in the equivalent gate-level representation) to a primary output, it follows that all *stuck-open*

[2] Number of times they are detected

faults in the circuit are covered by the test set [ReAgJa84] Furthermore, an analysis of the stuck-at fault simulation results indicates that for each line in the circuit, there is at least one vector pair for which stuck-at faults of either polarity on that line are detected in succession Hence the test set for each block provides complete coverage of all transition faults in the circuit

3.5 Test for the Tristate Drivers

As described earlier, the test chip contains a set of 10 independently accessible blocks, each with its own set of tristate drivers to throttle its access to the common output bus when not *selected*. If, however, a fault associated with a tristate buffer prevents it from being disabled — going into a state of high impedence — it nullifies the results of the tests of all the other blocks by corrupting the data on the shared output bus To test whether each tristate driver can be disabled, use is made of the pull-up/pull-down resistors at the primary outputs of the chip, which, when *all* tristate drivers (of all blocks) are de-selected, force the primary outputs of the chip into *default* logic states Essentially, all tristates are de-selected and then each tristate driven with a logic value opposite to that of the default logic value on its corresponding primary output Clearly, if the tristate still drives the output it will result in a logic state of the output the opposite of its default value If, however, a default value is maintained, it may be assumed that the tristate can be de-selected While a stuck-at fault on a primary output of the same polarity as its default state may render the tristate test ineffective, the stuck-at fault in such a case *overpowers* the faulty tristate driver and the output is simply diagnosed as stuck-at for all logic blocks

3.6 Test for the DEMPTY block

The DEMPTY block does not contain any logic apart from its set of tristate drivers at the output Each input leads, through the tristate, to its corresponding output A sequence of marching ones followed by a sequence of marching zeros is used to test the block The test, therefore, covers all stuck-at and bridging faults between all lines in the block

Chapter 4 Fault Diagnosis

Fault diagnosis refers to the process of determining faults in faulty systems. The resolution of diagnosis, depending on the techniques used and the application requiring the diagnostic information, can range from physical failures on chips to faulty boards in large systems.

Fault location is clearly necessary in any repairable system For example, a faulty module or board can be repaired by replacing faulty chips Similarly, many fault-tolerant systems can be re-configured by bypassing faulty components [CoFGJM87] Fault location often serves as a means to get an insight into process related failures for fabrication processes with unacceptably low yields, with an aim of "tuning" them for improvement [MalNai89] In the context of this thesis, automated fault diagnosis serves as a key element in the experimental validation of fault models

This chapter reviews some of the current state of the art fault diagnosis schemes, discusses the fault dictionary based approach and provides implementation details of an automated diagnosis package using the fault dictionary algorithm

4.1 Overview of Fault Diagnosis Schemes

The fault-dictionary based approach, which will be discussed at length in this chapter, is perhaps the simplest method of fault diagnosis One of the earlier references to the approach can be found in [TsiUlr62] where the fault dictionary is referred to as a "maintenance dictionary"

If the presence of a fault in the circuit causes the response of the circuit to a given vector to differ from its fault-free value, the vector is said to *detect* that fault Further, a

fault is said to be *detected on* a primary output for a given vector if the logic value observed on the output in the presence of the fault is different from its fault-free value

In essence, the fault dictionary contains complete information regarding the detection of all detectable faults of the circuit on each primary output for each test vector (for combinational faults) or each *pair* of test vectors (for sequential faults) Circuit responses are analyzed simply by *looking up* the corresponding entries of the fault dictionary Good responses from the circuit lead to the elimination of those faults as suspects which, if present, would have caused faulty responses The major drawback of the fault dictionary approach is in the computer resources required to generate, typically by fault simulation, and store the large amounts of information required for the fault dictionary

A modified approach has been suggested in [KaShKa89] whereby "non-candidate" faults are successively eliminated by. 1) performing a trace-back through the circuit from the "error output pins", 2) accounting for faults detected by test vectors with no erroneous outputs, 3) performing simulation on the remaining faults to determine those detected on good outputs of faulty vectors, and 4) calculating the "error probability" for each remaining fault.

In another novel approach, suggested by Yano and Okamoto [YanOka87], an electron-beam tester is used in conjunction with a conventional fault dictionary By treating all top level interconnects as "equivalent output pins" or pseudo-primary outputs, a fault dictionary with an unusually high fault resolution can be made

Most of the current state of the art schemes in fault diagnosis deal with diagnosis within the BIST framework In brief, BIST involves circuit test, typically by pseudo-random patterns, and comparison of a compacted response (signature) with a stored fault-free one [Wang88]. Both, pseudo-random pattern generation and response compaction are typically accomplished with linear feedback shift registers (LFSRs)

In intermediate signature collection (ISC) [WaiLin89], circuit signatures are taken every L patterns and the entire output sequence of L vectors for each failing block is stored for analysis Only those faults which could have caused the observed behaviour are retained in the list of suspects and these are then simulated on the set of L patterns within each failing block A variety of heuristics is suggested by the authors to cut down on the size of the potential fault list, and hence on the amount of simulation required

In algebraic analysis techniques, attempt is made to locate a failing pattern before performing any simulation. The method suggested by McAnney and Savir [McASav87] proposes to determine error patterns which could cause a given signature and then equate these to faults. In essence, the signature of each single bit error in a sequence of n vectors is pre-computed — where the compaction LFSR length is at least $log_2 n$ bits — and the faulty signature, assumed to be a single bit error, traced to a faulty pattern. Faults are then simulated to find those detected by the faulty pattern.

4.2 Discussion of the Fault Dictionary Approach

The small size of the present test circuit, coupled with the simplicity of the fault dictionary based diagnosis approach, makes for easy and efficient automation of the algorithm.

For each vector, for combinational faults, and each pair of vectors, for sequential faults, the fault dictionary contains a list of faults detected by the vector/vector-pair on each primary output In other words, for every unique combination of vector number (vector-pair number) and primary output, therefore, there is a corresponding list of faults detected The number of fault lists stored in the fault dictionary is $N \cdot PO$, where N is the number of test vectors in the test set and PO is the number of primary outputs in the circuit Combinations of vector numbers (vector-pair numbers) and primary outputs for which no faults are detected contain *null* fault lists

If the response of a circuit under test is "good" on a given primary output for a given vector (pair of vectors) then, clearly, none of the faults in the list corresponding to the particular combination of vector number and primary output can exist since their presence would, instead, result in a "faulty" response This reasoning is used to incrementally *prune* a list of suspect faults, initially containing all faults, for each good response logged for the circuit under consideration The faults pruned for each good response are *looked up* from the fault dictionary The faults remaining in the suspect fault list (SFL), after all responses have been processed, are the diagnosed faults for the faulty circuit

The basic diagnosis algorithm can be expressed mathematically as a set operation,

$$F \leftarrow F \setminus L_{i,j} \quad \forall\, i,j \in \{good\ response\}$$

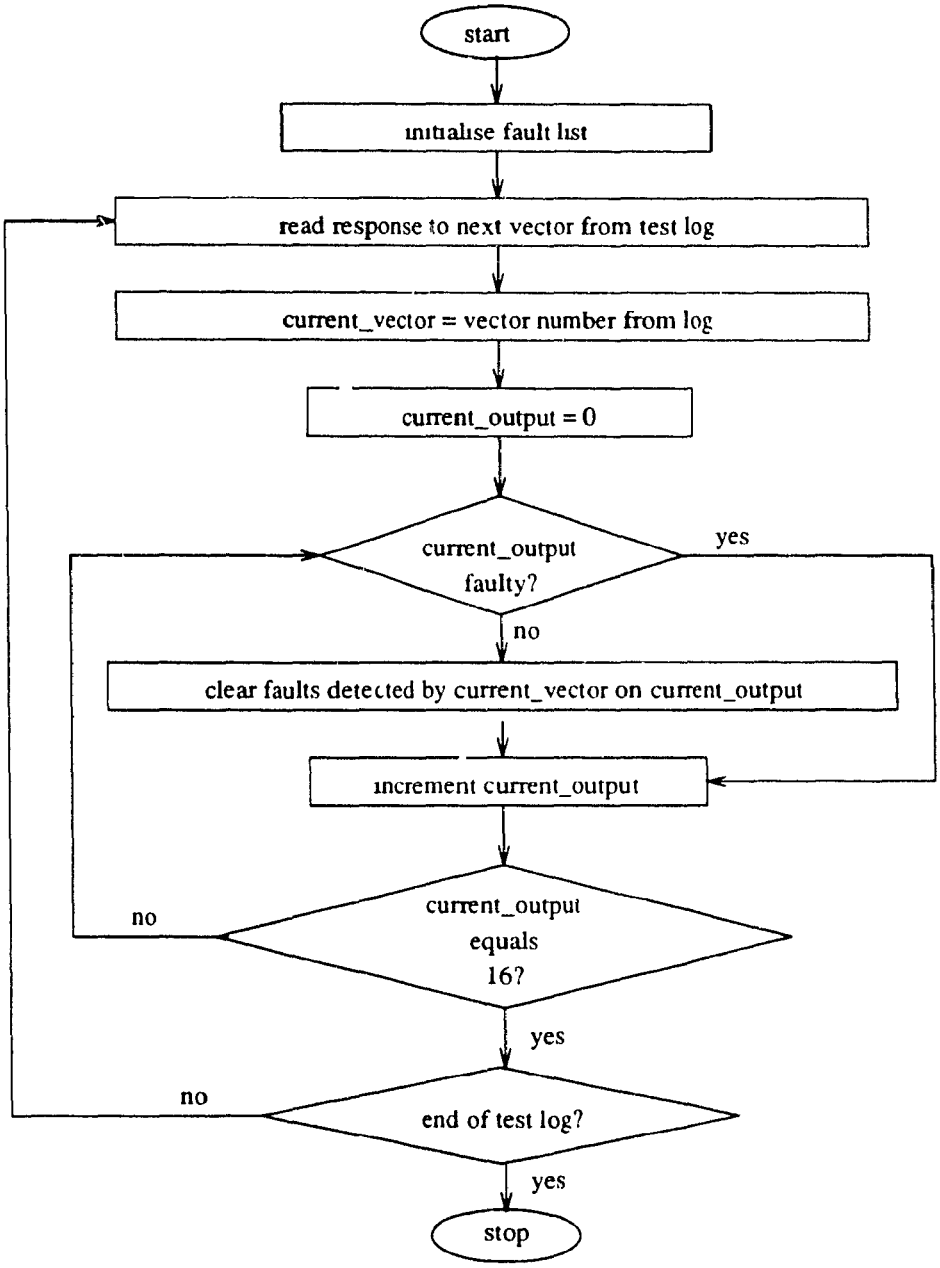


Figure 4.1 Flow of operations during fault diagnosis

where F is the suspect fault set, initially containing all faults, $L_{i,j}$ is the set of faults detected on vector i and primary output j and $\setminus$ is the set difference operator The steps involved in the algorithm are further illustrated by the flowchart of figure 4 1

4.2.1 Dealing with Structural Dominance of Faults

There are conflicting opinions regarding the definition of the term fault dominance

In the original definition given by Poage [Poage62] and subsequently used in text books [BreFri76], a fault f_2 is said to *dominate* fault f_1 if *all* tests which detect f_1 also detect f_2 but only *some* of the tests which detect f_2 also detect f_1 According to this definition, a s-a-0 fault on the output of a NAND gate dominates any input s-a-1. Some authors [Abraha86], however, have used the term to indicate the opposite relation, justifying it by the fact that, in a case such as this, it would only be necessary to consider the fault f_1 for the purpose of test generation.

A subtly different relationship — that of *structural fault dominance* — is used in fault diagnosis to retain only the *most significant* fault in every *group* of suspects.

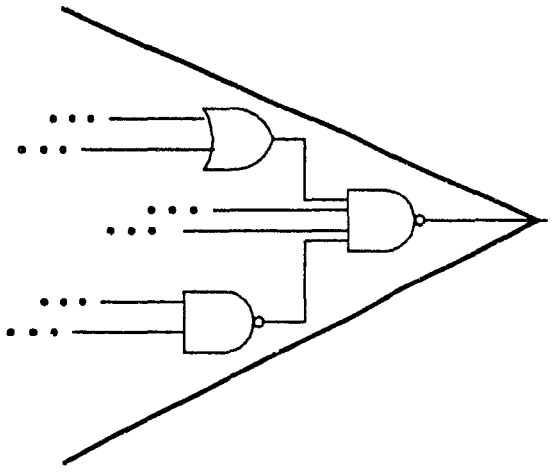


Figure 4.2 Fault Dominance in Logic Cone

Figure 4.2 illustrates the concept. If the output of the logic cone is faulty, we cannot unconditionally detect or diagnose faults on lines within the cone, regardless of their fault types For instance, if the output of the cone is stuck, the results of the diagnosis operation will indicate stuck-at faults of single polarities on *all* lines within the logic cone since values on nodes and lines in the circuit within the cone can no longer propagate through the output of the cone to a primary output. The fault at the output of the cone is said to *structurally dominate* all faults within

Structural fault dominance, in any given circuit, occurs within each fanout-free region. In principle, it is possible to stretch the concept of structural dominance to consider regions with fanout as well For example, if the output of a logic cone fans out into two

branches, each of which leads to a primary output, we cannot unconditionally detect or diagnose faults within the cone (including its output) if there are faults on *both* paths leading from the cone to the primary outputs. Faults on lines on either of the two paths leading from the logic cone to the two primary outputs are said to structurally *half-dominate* all faults associated with the logic cone (including its output)

To prevent an unnecessarily large number of faults from *cluttering up* the list of suspects as a result of structural dominance and half-dominance, the list is pruned after the diagnosis operation to retain only the most structurally dominant faults as suspects While it is possible to consider partial dominance relations between logic cones and multiple fanout paths, it is not deemed necessary to implement it in the diagnosis package because of its large implementation overhead and low marginal gain — the probability of faults occurring on *all* fanout paths leading from a logic cone to a primary output is assumed to be very small

4.2.2 Detecting Unmodelled Fault-Sites

Using "unrealistic" fault models can lead to there being no diagnosed modelled faults in faulty circuits While this reflects on the inability of the fault model to accurately model physical defects, it does not provide a measure of the effectiveness of the *test set* generated to cover all faults under the given fault model due to its windfall coverage of other crucial fault types Ways of getting such measures are discussed in chapter 5

To get a better understanding of the nature of faults, however, a *localising* method is developed to get an indication of the neighbourhood of unmodelled fault sites in faulty circuits in which no fault(s) can be diagnosed Once such sites are determined, specialized instruments, such as electron-beam voltage-contrast probers, can be used to study the exact nature of the unmodelled faults

If two circuit lines are *bridged* together due to a physical defect in the circuit, one would expect the logic value on at least one line — and possibly both — to be dependent on the logic value of the other according to some wired-logic function [Mei74] Further, assuming that both logic values are observed on either line at some point during the application of the test set — consistent with the faulty wired-logic behaviour — no stuck-at faults will be diagnosed for the faulty circuit if diagnosis is performed on the basis of the stuck-at fault model If, however, a list is kept for all faults containing the number of

times each fault could *account* for a faulty response, clearly, all faulty responses could be jointly accounted for by stuck-at faults — possibly of both polarities — on both lines. The number of faulty responses a fault *accounts for* — regardless of whether it is eventually *cleared* from the list of suspects — is termed its *suspect count*. In devices with single unmodelled faults, modelled faults with large suspect counts — close to the total number of faulty responses — can *lead* to the neighbourhood of the actual fault sites.

To perform such an analysis, a *fault suspect count* list is generated, containing numeric entries corresponding to all possible faults The entries are initialised to zero. For each subsequent faulty response, the suspect counts of all faults detected for the combination of faulty vector (pair of vectors) and primary output — looked up from the fault dictionary — are incremented by one. The suspect count for each fault, after all responses have been processed, indicates the total number of faulty responses the fault could *account* for. The flowchart in figure 4 3 illustrates the operations involved in detecting probable unmodelled fault sites

The algorithm can be expressed mathematically as a set operation.

$$s_{fault} \leftarrow s_{fault} + + \ \forall\, fault \in L_{i,j} \ and \ \forall\, i,j \in \{faulty\ response\}$$

where s_{fault1} is the suspect count of fault $fault1$, $L_{i,j}$ is the set of faults detected on vector i and primary output j and $++$ is the arithmetic increment-by-one operator.

4.2.3 Storage Format

As described previously, the fault dictionary contains the list of faults detected on each primary output for each test vector (pair of vectors) Each fault is given a unique numeric ID for reference Since the diagnosis operations performed are 32 bit *vector boolean* in nature (as will be described later), information is stored for groups of 32 vectors each Each line in the fault dictionary contains information in the following format·

$$[vector\_group]\ [primary\_output]\ [fault\_number] = 32\ bit\ decimal\ coded\ binary\ entry$$

For example the line.

$$1 - 32\ pofl\,[7]\,[43] = 8$$

contains the detectability of fault number 43 on primary output 7 for vectors 1 to 32 The information (detectability profile) for each vector, for a given fault and a given primary

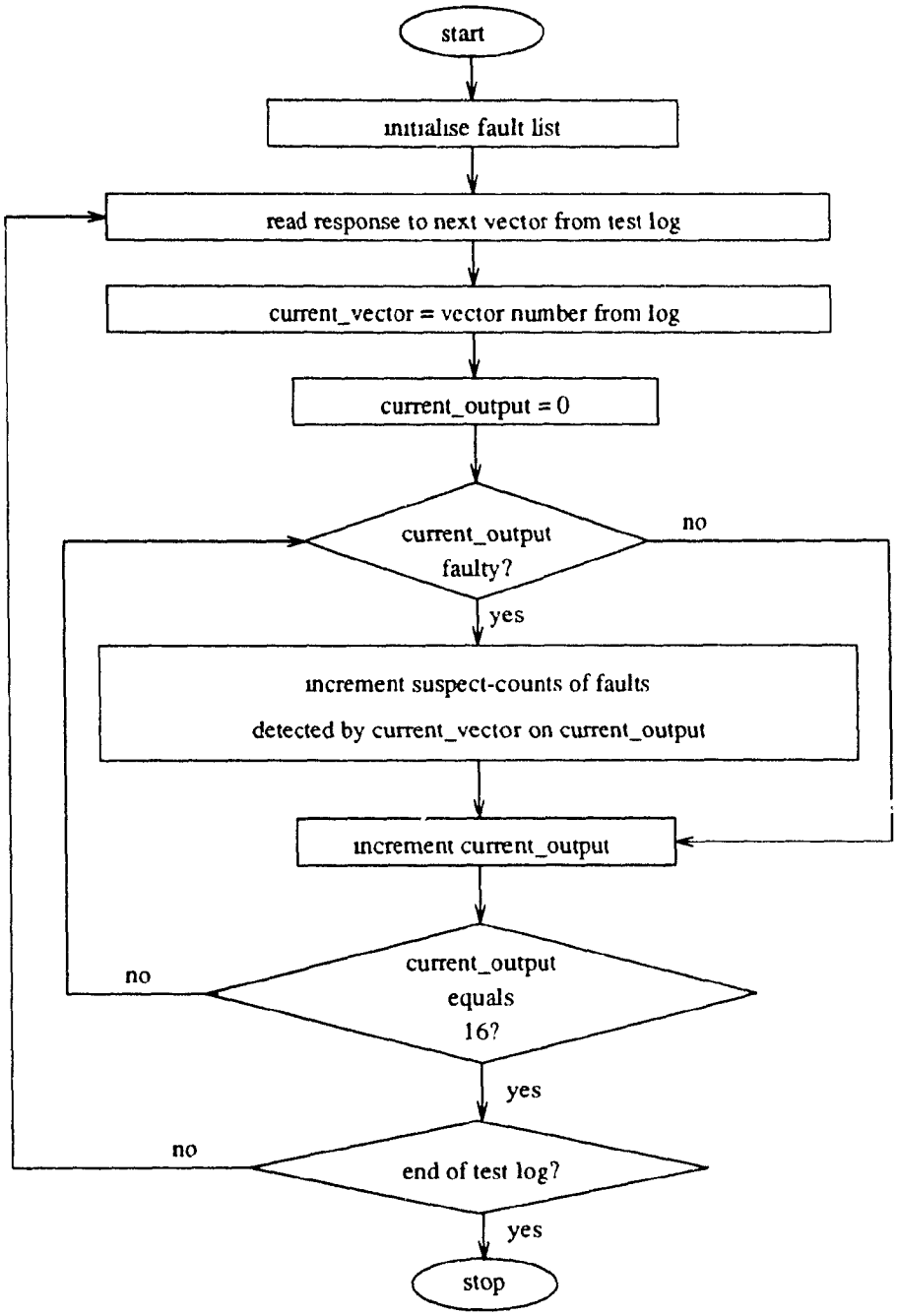


Figure 4.3 Detecting probable unmodelled fault sites

output, is binary in nature with a 1 indicating detection of the fault on the primary output and a 0 indicating otherwise Each 32-bit binary sequence representing the detectability profile of a group of vectors is coded and stored as a decimal number As illustrated in table 4.1, the value 8 in the example above indicates that fault 43 is detected on output 7 on the fourth vector in the first group of 32 Numeric IDs are assigned to circuit faults

such that complementary faults (of opposite polarity) on the same line are represented by a pair of successive integers.

For sequential faults, in which fault detection occurs only for pairs of vectors, the convention followed in coding the fault dictionary is to indicate detection of a fault for a vector number only if the fault is detected for the combination of the present *and previous* vectors

The format of the fault dictionary, following the conventions described in the section, is independent of the fault model used to compile it

Vector Number	32	31	...	4	3	2	1
Detectability (8_{10})	0	0	.	**1**	0	0	0

Table 4.1 Detectability of fault 43 on output 7 for vectors 1-32

4.2.4 Compilation for Different Fault Models

A fault simulator is a tool which, for each vector (pair of vectors) of a given vector set, determines all fault(s) [3] that cause the response of a given circuit to be different from its fault-free value and, for each faulty response, determines all primary outputs affected by each fault Hence, given the fault-free response for a test set, a fault simulator can be used to generate the detectability profile of all faults on all primary outputs — exactly the information required to build a fault dictionary

Fault collapsing, which refers to the representation of only one out of each set of equivalent faults in the circuit fault list, is often used as an option in fault simulators to speed up the processing time required to get fault coverage measures for test sets. It is essential, in order to build a fault dictionary, to have complete detectability information for *all* circuit faults so that all necessary faults, regardless of their possible equivalence, may be pruned from the list of suspects for each good response logged It is essential, therefore, to either perform fault simulation using a non-collapsed fault set or to perform fault simulation using a collapsed fault set and *later* apply fault equivalence relations to generate complete data for a fault dictionary

[3] Based on a given fault model

In summary, a fault dictionary for any given fault model can be created simply by fault-simulating the circuit under consideration for the given test set using a non-collapsed fault set and reformatting the generated information into the desired format.

The pre-eminence of the stuck-at fault model has lead to the development of many automated test and testability tools — like fault simulators and test pattern generators — to be based on the stuck-at fault model. In fact, stuck-at fault simulators and ATPG packages are almost universally used by commercial and research organizations alike. While work has previously been done towards developing transition fault simulators [LevMen86], such simulators are not commonly used, or even easily available for use.

It is possible, however, for the purpose of generating a fault dictionary, to use the results provided by a stuck-at fault simulator and *interpret* them to extract the information required for generating a transition fault dictionary. In addition, it is possible to use a suitably modified circuit netlist to perform bridging fault simulation on a circuit, using a stuck-at fault simulator.

What follows is a discussion of the compilation of the stuck-at and transition fault dictionaries for the test circuit using the stuck-at fault simulator TULIP [MaaRaj88]. The section concludes with a brief overview of how a bridging fault dictionary may be compiled on the basis of stuck-at fault simulation results, using a suitably modified circuit netlist.

4.2.4.1 Stuck-at Faults

In order to compile the fault dictionary for stuck-at faults, the test set is first fault-simulated for the circuit in question using a non-collapsed fault set. The header of the fault simulator output contains a cross-reference of the numeric IDs for all stuck-at faults in the circuit in terms of their circuit netlist line IDs and fault types. For example, the line

$$d30/e2/t1\ sa0 \rightarrow [33]$$

indicates that the stuck-at-0 fault (s-a-0) on line d30/e2/t1 of the circuit is given the numeric ID 33 for subsequent references. The simulator follows the convention described earlier to assign successive numeric IDs to stuck-at faults of opposite polarity on the same circuit line.

The body of the fault simulator output contains all the relevant information for the fault dictionary. Each line contains the detectability profile of a given fault on a given

primary output. For example, the line:

$$[1 - 32][0][646] = 01110011100000000000000000000$$

contains the detectability profile of fault 646 on primary output 0 for vectors 1 to 32, one bit for each vector The bit position corresponding to vector 1 is the right-most. From the line it is clear that fault 646 is detected on vectors 24, 25, 26, 29, 30 and 31. The fault simulator produces output in increasing order of faults, primary outputs and vector numbers respectively

The output produced by the fault simulator is subsequently processed by a program to extract the relevant information, reformat it, and store it as a fault dictionary. As described previously, the detectability information is stored as a decimal coded binary entry for each block of 32 vectors The information is stored in increasing order of vector numbers, faults, and primary outputs, in that order, for more efficient access by the diagnosis programs.

4.2.4.2 Transition Faults

In general, if stuck-at faults of opposite polarities are detected on a given line of a circuit by subsequent vectors, then clearly, a *transition* fault is detected on the same line for the given vector pair This forms the underlying premise for the creation of a transition fault dictionary on the basis of stuck-at fault simulation results

For example, consider the following output lines from the fault simulator.

$$[1 - 32][0][1] = 00000000000000000000000000001$$
$$[1 - 32][0][2] = 00000000000000000000000000010$$

where fault numbers 1 and 2 refer to stuck-at-0 and stuck-at-1 faults on the same line Clearly, a stuck-at-0 is detected, on the line in question, on vector 1 A stuck-at-1 on the same line is detected on vector 2 Since the detection of a stuck at-0 fault on the line on vector 1 *implies* a value of 1 on it in the fault-free case and the detection of a stuck-at-1 on the line on vector 2 similarly *implies* a value of 0 on the line in the fault-free case, the first vector pair (vectors 1 and 2) *detects* the *slow-to-fall transition* fault

A program has been coded to efficiently implement the procedure to generate a transition fault dictionary on the basis of the results of a stuck-at fault simulation. The

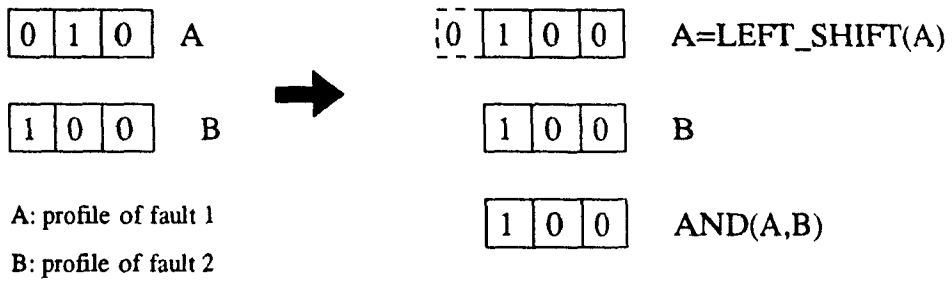


Figure 4.4 Computing the detectability profile of slow-to-rise faults

program extracts information about transitions on circuit lines from the stuck-at fault simulation output through two basic logic operations — left shifts and bitwise ANDs — performed on the stuck-at detectability profiles of the circuit in question

As an example, consider the detectability profiles

$$[1-3][0][1] = 010$$
$$[1-3][0][2] = 100$$

The information — a truncated version of the actual format — indicates that fault 1, stuck-at-0 by convention, on the given line is detected on primary output 0 on vector 2, and fault 2, stuck-at-1 on the same line by convention, is detected on primary output 0 on vector 3. If the detectability profile of fault 1 is *shifted left* one bit and the resulting array ANDed bitwise with the detectability profile of fault 2, we get the detectability profile of the corresponding *slow-to-fall* fault on the same line The operation is illustrated in figure 4.4. The transition fault detectability profile thus created follows the conventions described earlier — detection on a vector implies detection for the combination of the *previous and present* vectors The detectability profile of the slow-to-rise fault can be similarly generated by left shifing the detectability profile of fault 2 and bitwise ANDing the resulting array with the detectability profile of fault 1

4.2.4.3 Bridging Faults

In a circuit with n lines, there can be $n \cdot (n-1)$ or $O\left(n^2\right)$ bridging faults, assuming bridges are limited to those between any *two* circuit lines at a time Since bridging faults are related to the layout topology of a circuit, the list of potential bridging faults within the circuit may be substantially pruned by considering only combinations of lines in close proximity to each other

Consider the circuit lines a and b shown in figure 4 5 The lines are bridged together forming an electrical short, implying, by definition, identical logic values on both lines

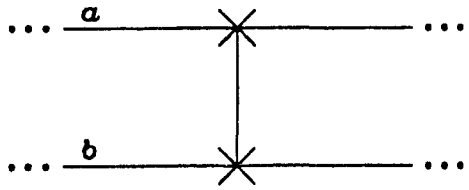


Figure 4.5 Bridge between lines a and b

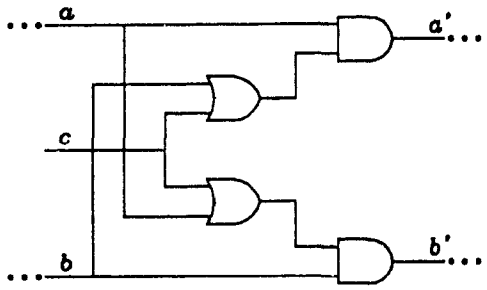


Figure 4.6 Modelling and detecting "AND"-type bridging faults between lines a and b

Hence, a test vector that detects the bridging fault also *implies* complementary logic values on the two lines Depending on the nature of the wired logic formed, the circuit can be suitably modified in a manner such that the detection of a stuck-at fault on a modified circuit line guarantees detection of the bridging fault modelled by the circuit.

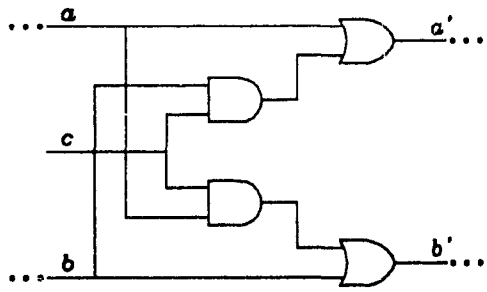


Figure 4.7 Modelling and detecting "OR"-type bridging faults between lines a and b

Figure 4 6 illustrates how the circuit can be modified to *model* an AND-type bridge between lines a and b As can be seen in the figure four extra gates have been added to the original circuit The lines a' and b' carry the same logic values as the lines a and b respectively, provided that the line c, a primary input in the modified circuit, is driven by a logic 1 For modelling "normal" circuit operation, therefore, input c is kept high If the

the stuck-at-one fault on line c (c_{s-a-1}) is detectable, it will be detected under either, and possibly both, of the following conditions. $c = 0$, $a = 0$ and $b = 1$, and $c = 0$, $a = 1$ and $b = 0$. Under the first condition, the stuck-at fault propagates to a primary output through line b' — while line a is held low — indicating an incorrect value of 0 on line b due to a wired-AND bridge between lines a and b [Mei74] Under the second condition, where $a = 1$ and $b = 0$, the same bridging fault is detected through line a', implying a a faulty value on line a due to the wired-AND behaviour between lines a and b

Figure 4 7 illustrates how the circuit may be modified to model and generate tests to detect the OR-type bridging fault between lines a and b Detection of a stuck-at-zero fault on line c (c_{s-a-0}) in the modified circuit implies the detection of the OR-type bridging fault be.ween the lines a and b

By suitably modifying a circuit as described above to account for likely bridging faults, a bridging fault dictionary can be generated on the basis of stuck-at fault simulation

It should be noted, however, that the assumption of an electrical dead-short between two circui. lines is valid only for tracks running in metal Bridges between tracks of other, possibly dissimilar, layers can result in different electrical behaviour because of resistive and capacitive effects

4.3 Implementation Details of Fault Dictionary Based Approach

A fault diagnosis package has been coded in the "C" programming language for a compiler supporting 32-bit vector boolean operations The storage format of the fault dictionary, in fact, was chosen so as to effectively use the vector boolean manipulation capabilities supported by the compiler

The fault dictionary is read in by the program and stored internally as a triply indexed array of long (32-bit) integers The value of each array element indicates the detectability of the specified fault on the specified primary output for the specified group of 32 vectors The indices uniquely represent the vector group (of 32 vectors), primary output, and fault number

The circuit response log from the tester contains the vector numbers of all faulty responses for each primary output The information is read in, reformatted and stored

internally as a doubly indexed array of long (32-bit) integers The value of each array element (response vector) indicates the actual response of the circuit on the specified primary output for the specified group of 32 vectors Each bit in a given response vector corresponds to the response of the circuit to a unique test vector within the group of 32 (on the given output) with a "1" indicating a good response and a "0" indicating a faulty one.

The fault list is stored as a boolean array indexed by the fault number. A "1" corresponding to a fault in the list indicates that the fault is suspected while a "0" indicates that it is cleared The list is initialized to all 1s to indicate all faults as suspects

The actual diagnosis procedure, while being the same in principle, warrants different implementations for handling stuck-at and transition faults — while each logged response is processed as an entity for diagnosing stuck-at faults, responses are processed as pairs for diagnosing transition faults

4.3.1 Stuck-at Fault Diagnosis

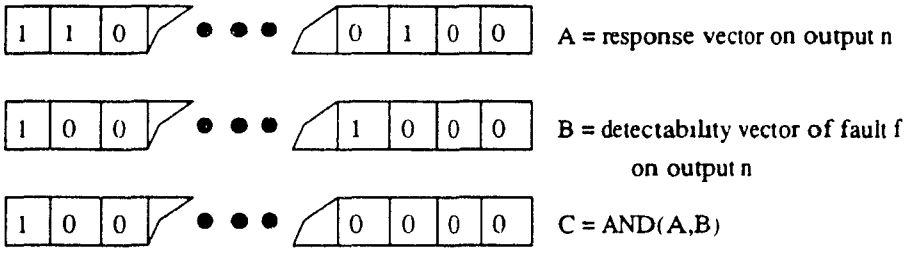


Figure 4.8 Bitwise AND operations in stuck-at fault diagnosis

A stuck-at fault needs to be removed from the list of suspects if it is detected on a primary output for a given vector *and* the response of the circuit on the same primary output for the same vector is good This condition is established by performing bitwise AND operations between response vectors and their corresponding detectability profile vectors for each fault For instance, if the result of a bitwise AND operation between a response vector — for a particular vector group and primary output — and the detectability vector of a fault f — for the same vector group and primary output — is non-zero, then clearly, there is *at least* one vector in the group on which the fault f is detected *and* the actual response of the circuit is good As illustrated in figure 4 8, if vector C has a non-zero value, the fault f may be cleared from the list of suspects

For each response vector the bitwise AND operation is performed with the corresponding detectability vectors of all faults which are still suspected (*all* faults are initially suspected). When a fault needs to be cleared from the list of suspects, the fault number serves as an index to its entry in the fault list The entry is subsequently reset to 0 to indicate a "cleared" fault.

4.3.2 Transition Fault Diagnosis

As described earlier, responses need to be processed in pairs for transition fault diagnosis In particular, a transition fault may be removed from the list of suspects only if it is detected by a pair of vectors **and** the observed responses to **both** vectors (of the pair) are good At any time during the diagnosis process, therefore, information is required not only about the circuit response for the *current* vector but also that for the *previous* vector In order to efficiently automate such processing, *delayed-response vectors* are generated for each vector group (of 32 vectors) which have the property that for each vector bit position, they contain the response of the circuit (on the given output) to the *previous* vector. Delayed response vectors are generated from their corresponding response vectors simply by performing a specialized one bit left shift operation such that the bit *shifted in* — the least-significant bit of each delayed response vector thus created — is identical to the most-significant bit of the *previous* response vector Figure 4 9 illustrates the operation

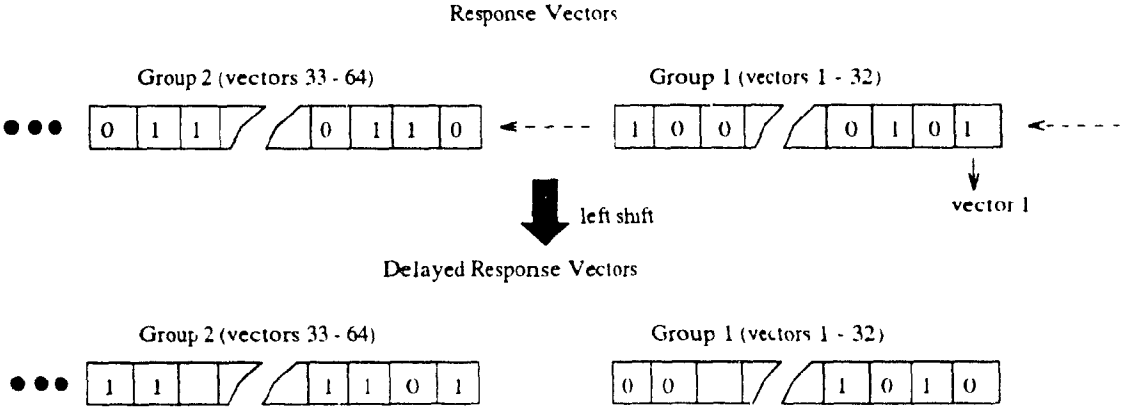


Figure 4.9 Creation of delayed response vectors

Once the delayed-response vectors have been created, the actual diagnosis procedure is implemented in much the same way as it is for stuck-at faults For each fault, primary output and vector group, a bitwise AND operation is performed on *three* operands — the

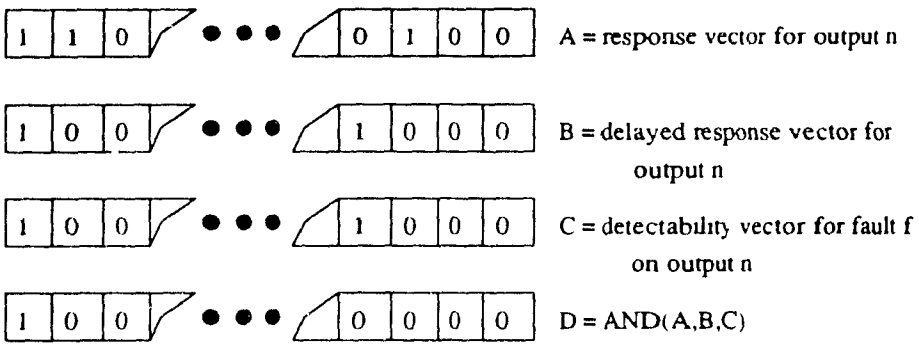


Figure 4.10 Bitwise AND operations in transition fault diagnosis

response vector, the delayed-response vector and the detectability vector for the fault in question, for the given vector group and primary output If the result of such a bitwise AND operation for a fault f — for a given vector group and primary output — is non-zero, then clearly, there is *at least* one vector pair in the vector group for which the fault f is detected *and* the circuit response to both vectors in the pair is good The fault f, therefore, may be deleted from the suspected fault list Figure 4 10 illustrates the operation. Since the vector C in the example is non-zero, fault f may be cleared from the list of suspects

4.3.3 Locating Probable Unmodelled Fault Sites

Suspect-counts for each fault are stored in an array of integers indexed by the fault number All counts are initialized to 0 The response of the circuit is stored in response arrays similar to those used in the diagnosis routines The convention followed, however, is different in that a "1" in any bit position within a response vector, corresponding to the response of a particular vector on a particular output, indicates a faulty response and a "0" indicates a good one Delayed-response vectors are created, using the same format, for handling transition faults The algorithm itself is implemented in a manner similar to the diagnosis routines For stuck-at faults, each response vector is bitwise ANDed, in turn, with the corresponding detectability vector for each fault If the result of such an operation for a fault f is non-zero, the number of $1s$ (ones-count) in the result indicates the number of faulty responses in the group of 32 the fault could *account for* The suspect-count for the fault in question is subsequently incremented by that amount (ones-count) For transition faults, the bitwise AND is performed on three operands — the response vector, the delayed-response vector and the corresponding detectability vector for each fault — and the suspect-count incremented by the ones-count of the result

Chapter 5 Measures of Effectiveness of Fault Models and Test Sets

As described earlier in chapter 1, effective fault modelling is essential to a high quality of testing In reality, however, it is very unlikely for any single fault model to accurately account for *all* kinds of physical defects that are likely to occur in a circuit While an inability to locate faults in known faulty circuits on the basis of a particular fault model is an indication of its (the model's) ineffectiveness, it does not provide any clear indication of the effectiveness of the *test set* generated to cover all faults — or a large percentage of them — under the given fault model The effectiveness of any deterministically generated test set depends not only on the fault model used as its basis but also on the CAD tools used to generate it, since a particular algorithm may, inherently, stand to cover more crucial non-modelled faults than another

Broadly speaking, there are two ways to measure the effectiveness of test sets the bottom-up approach where actual physical defects on chips are analysed and the test set evaluated for its ability to detect them and, the top-down approach where the "performance" of the given test set may be evaluated by comparing results of actual device tests with those of a known "good" reference test set

The first approach clearly focusses on the investigative study of actual physical failures While Shen, Maly and Ferguson [ShMaFe85] provide a method to account for such failures to come up with a ranked fault list, additional work is required to use the fault lists to evaluate what percentage of such faults any given test set stands to cover

The second approach requires the generation of a reference test which ideally, provides coverage of faults from a variety of different fault models The test set generated according to the methodology presented earlier in chapter 3 can effectively serve as the required reference since its qualities of redundancy, robustness and a complete coverage of

faults under a number of different fault models make it very close to being an *ideal* test set. While it may be tedious to generate such a test set for large circuits, the actual test vehicle for which the test set is required to be generated need not be a large circuit — it need only be *representative* of the class of circuits being studied. The results can be *scaled*, if required, to account for larger areas

This chapter looks at the implications of using ineffective test sets, outlines methods to measure the effectiveness of test sets and fault models and discusses how the results may be scaled to account for larger circuits

5.1 Defect Level and Escape Rate

The ultimate aim of any production test strategy is to keep the number of defective parts shipped out within "reasonable" limits By extension, then, a crucial requirement of the test set used for the purpose is for it to be able to *detect* all faulty devices or, certainly, a very large fraction of them

The *defect level* of a test process serves as a measure of its effectiveness It is defined as a relative measure of the number of "bad" chips classified "good" on the basis of the test [WilBro81] In mathematical terms, the defect level, DL, is expressed as:

$$DL = \frac{F_G}{G + F_G}$$

where G is the number of good chips and F_G is the number of faulty chips classified good Some authors have previously used the term *reject ratio* [SetAgr84] to indicate the same measure

Faulty chips that go undetected due to the ineffectiveness of the test set are said to *escape* the test The *escape rate* of a test is defined as a measure of the number of escaping chips relative to the total number tested

Williams and Brown have shown [WilBro81] that the defect level of a test can be determined in terms of the process yield, Y, and the fault coverage, $\frac{m}{n}$ according to the relation

$$DL = 1 - Y^{(1-\frac{m}{n})}$$

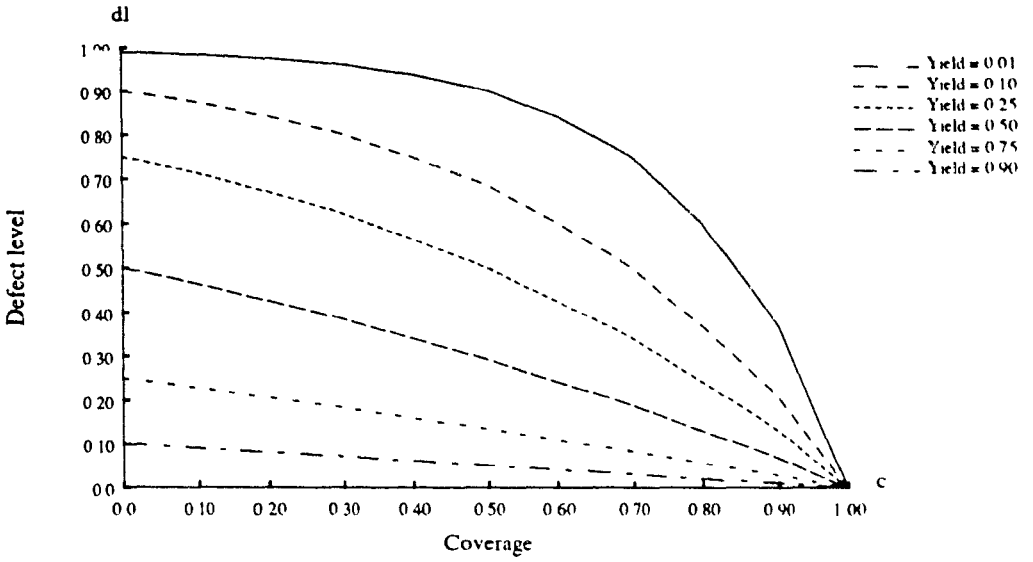


Figure 5.1 Defect level as a function of fault coverage

under the assumptions that a given chip has exactly n faults out of which m are actually tested, the probability of a fault occurring is independent of the occurrence of other faults and all faults are equally likely to occur

Figure 5 1 shows the variation of defect level as a function of fault coverage for different process yields. For high yield processes the decrease in defect level is almost linear with an increase in fault coverage For lower yields, however, increase in fault coverage beyond a certain *threshold*, corresponding to the elbow in the curve, leads to a rapid reduction in the corresponding defect level

5.2 Components of Defect Level

Chips escaping detection by a given test set need to be analyzed for their failures so as to help generate better tests to detect them Each *class* of such failures (faults) on escaping chips is said to form a *component* of the defect level

What follows is a discussion of the different components of the defect level of a test for a given *single* fault model (like the single stuck-at fault model) A single-fault model is used as a basis for the discussion since most, if not all, current ATPG tools generate test patterns based on the single-fault assumption

Faulty chips escaping a test set designed to cover single faults can be classified to fall into one of 3 categories — chips containing non-modelled faults only, chips containing single modelled faults, possibly in addition to other non-modelled faults, and chips containing multiple occurrences of modelled faults, possibly in addition to other non-modelled faults The defectlevel of a single fault test set is given by

$$DL = \frac{(F_0 + F_1 + F_{\geq 2})}{(G + F_0 + F_1 + F_{\geq 2})}$$

where F_0, F_1 and $F_{\geq 2}$ are, respectively, the number of escaping chips falling into each the 3 categories described above and G is the number of good chips. For chips containing non-modelled faults only, we rely completely on the *windfall* coverage of the test set to detect them.

5.2.1 Single Faults

The defect level due to incomplete coverage of single faults is given by.

$$DL_{single\_faults} = \frac{(1 - Y)(1 - Q_1) k_{sf}}{\{Y + (1 - Y)(1 - Q_1) k_{sf}\}}$$

where Y is the yield of the process, Q_1 is the coverage of single modelled faults ($Q_1 \leq$ 100%) and k_{sf} is the fraction of faulty chips containing single modelled faults only Clearly, the term $(1 - Y)(1 - Q_1) k_{sf}$ amounts to the fraction of all bad chips fabricated containing only single modelled faults likely to go undetected due to incomplete coverage of such faults Figure 5 2 shows the variation of this component of the defect level with single fault test coverages for a fixed value of k_{sf}, assumed to be 30%

5.2.2 Multiple Faults

If there are n lines in a circuit, each of which can be in one of three possible states — fault-free, stuck-at-0 and stuck-at-1 (assuming the stuck-at fault model) — there are 3^n possible states for the circuit Since there is only 1 fault-free state, there are $3^n - 1$ possible multiple faults in the circuit

While the definition of multiple faults is clear, authors have previously used a number of different definitions to get measures of coverage of multiple faults [CoxRaj88a] [JacBis87] [RajTys85] It has been shown that measures of absolute coverage of multiple faults can

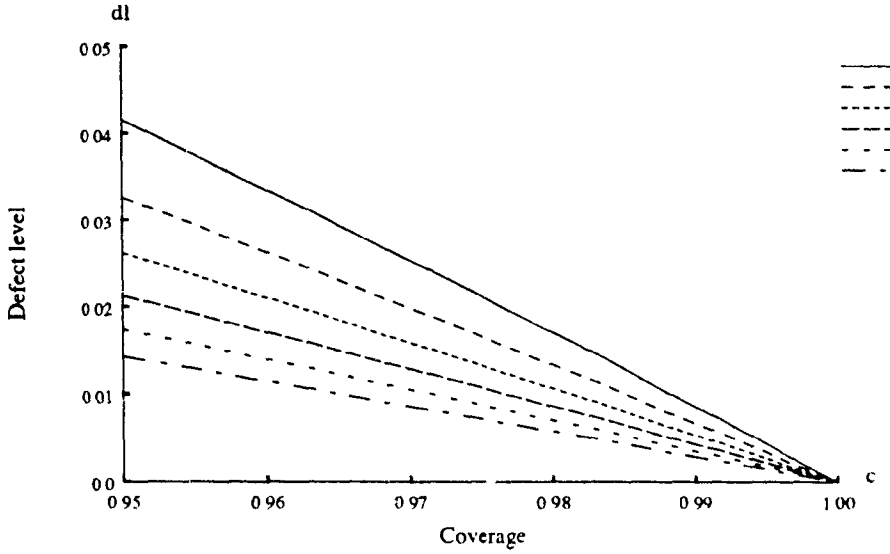


Figure 5.2 Variation of defect level with fault coverage for single faults

be misleading [ColvAR88] while still being mathematically sound. In fact, it is shown in [ColvAR88] that in a circuit with n lines and k primary outputs on which faults are *guaranteed to be detected* (GTBD), regardless of the presence of other faults, a lower bound on multiple fault coverage is given by:

$$Coverage \geq 1 - \frac{1}{3^k}.$$

The expression, interestingly, is independent of n which leads one to believe, albeit erroneously, that increasing the number of primary outputs of any circuit, regardless of its size, should result in an increase in fault coverage

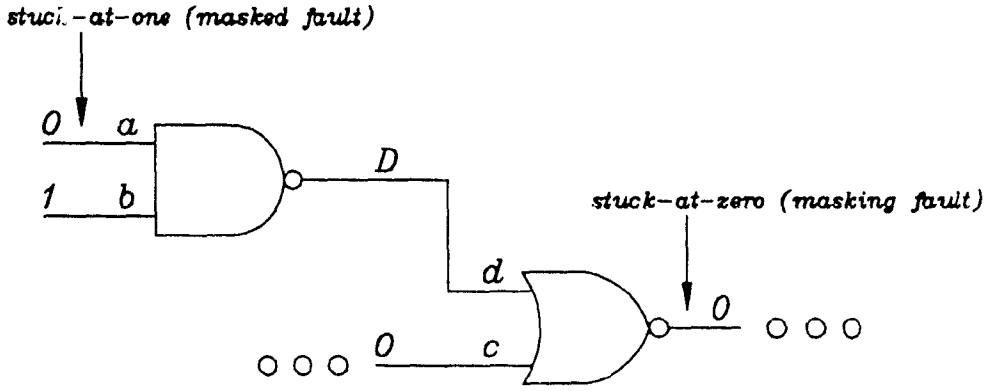


Figure 5.3 Fault Masking

In any case, a complete test set for single faults may fail to detect a small fraction of multiple faults due to the phenomenon of *fault masking* Fault masking occurs when the fault effect produced by a fault is nullified or *masked* by the presence of another set of faults. There are three different possibilities of fault masking [Pradha86]

1 Fault f_2 may mask fault f_1 in such a manner that the combined effect of faults (f_1, f_2) makes the circuit behave as if there were no faults in the circuit at all.
2 Fault f_2 may mask fault f_1 in such a manner that a particular test vector or set of test vectors expected to detect f_1 does not detect f_1 in the combined presence of (f_1, f_2)
3 Faults f_1, f_2, f_3, f_4 or any combination of up to three of these faults may be detectable, but the combined effect (f_1, f_2, f_3, f_4) is undetectable.

Figure 5 3 illustrates a simple case of fault masking. As can be seen from the figure, in order to detect a stuck-at-one fault on line a, line b is held at logic one, line c is held at logic zero and a logic zero is applied to the line in question (line a) If the stuck-at-one fault on line a is the only fault in the structure, then clearly, we should expect a value of one on line d but should actually observe a zero on it, which should propagate to the output as a logic 1 Since the output itself is stuck-at-zero, we observe the "correct" response for the test for line a stuck-at-one The stuck-at-one fault on line a is said to be masked by the stuck-at-zero fault on the output

Lower bounds on the coverage of multiple faults with single fault test sets — after accounting for such masking — have previously been derived [AgaFun81] It has been shown on the basis of an experimental study on the 74LS181 4-bit ALU that single stuck-at fault test sets can provide extremely high multiple fault coverage for "practical" circuits [HugMcC86] In an experiment conducted in [HugMcC86], it was found that 16 different, complete single stuck-at fault tests each detected more than 99 96% of the double stuck-at faults (faults of multiplicity two) in the ALU It has also been analytically shown [JacBis87] that at least 99.67% of all multiple faults in any circuit are detected by a single fault test set if the number of primary outputs in the circuit is three or more

On the other hand, if fault masking does not occur, which is a very unrealistic assumption, the detection probability of multiple faults is much higher than the corresponding single fault coverage For example if the coverage of single faults in a given circuit is Q_1, the likelihood that a single fault is not covered by the test is $1 - Q_1$ By extension, the

probability that a double fault exists such that neither of its components is detected by the test set is $(1 - Q_1)^2$. Therefore, the likelihood of detection of any double fault by the single fault test is $1 - \{(1 - Q_1)^2\}$. Hence, given a single fault coverage of Q_1, the detection probability of faults of multiplicity k, Q_k, assuming fault independence and an equally likely occurrence of faults, is given by

$$Q_k = 1 - \{(1 - Q_1)^k\}.$$

As an example, for $k = 2$ and $Q_1 = 0.99$, $Q_k = 0.9999$

In general, a test set providing a sufficiently high coverage of single faults should result in only a negligibly small proportion of faulty chips escaping detection due to the presence of multiple faults.

5.3 Evaluation of Fault Models and Test Sets

As described earlier, a reference test set can be used to evaluate the effectiveness of a given fault model and a test generated to cover all faults under the model

In the first phase of the experiment, a batch of (sample) chips are tested with the extended fault test set (reference test) All faulty responses on all chips detected faulty are logged

For each faulty chip, a *modified* test set is generated from the original extended fault test set (EFTS) by *deleting* all vectors on which faulty responses were logged The modified test set is then simulated for faults using the fault model under consideration Clearly, if the coverage of modelled faults remains complete (100%) with the modified test set, a complete test (for modelled faults) can be constructed which would fail to detect the faulty circuit On the other hand, if the resulting coverage of the modified test for modelled faults falls to less than 100%, we can conclude that there is at least one modelled fault in the circuit which would be detected by a test set with complete coverage of modelled faults.

The experiment, therefore, provides a measure of the effectiveness of a fault model and a worst-case measure of a test set generated to cover all faults Whether faulty chips actually get detected by a given test set generated to cover all faults depends on its windfall coverage of "crucial" faults

The effectiveness of a given test set can be estimated simply by re-testing known faulty chips, tested earlier with the EFTS, with the test set under evaluation. The relative number of faulty chips escaping detection by the test set under evaluation then gives a lower bound on the escape rate of the test. Note that the method provides only a lower bound on the escape rate since the EFTS itself may not detect *all* faulty devices

5.4 Scaling of Results

The above methods of estimating the effectiveness of fault models and test sets require an extended fault model test which may be time-consuming to generate for large circuits with irregular structures It is easier, therefore, to use a smaller circuit as a test vehicle to represent the class of circuits being studied The results of effectiveness of fault models and test sets, in terms of defect levels and escape rates, then need to be *scaled* to be applicable to larger circuits utilizing larger silicon areas

It has previously been shown [SetAgr84] that the necessary fault coverage required for a given defect level is given by.

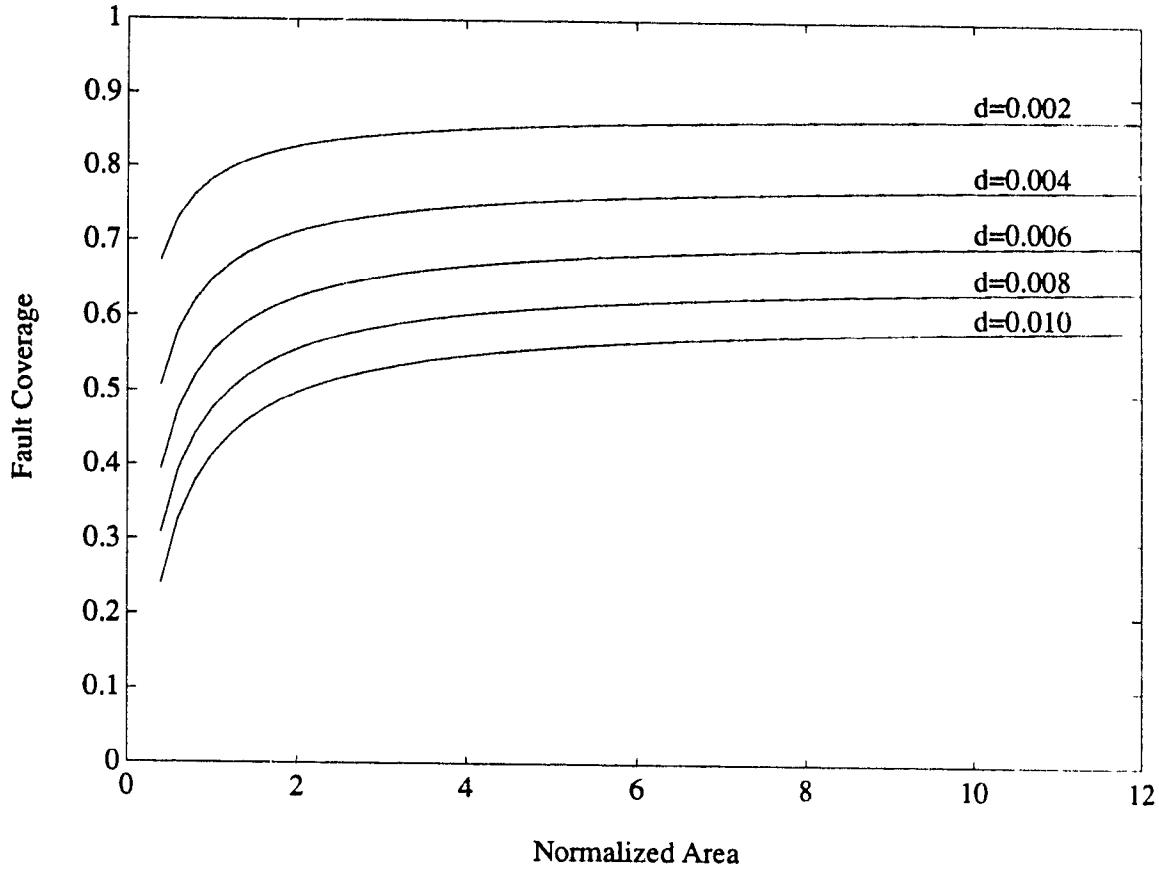
$$f = -\frac{1}{c} ln \left[1 + \frac{1 - (1-r)^{\frac{1}{a}} \{ 1 + Ab\,(1 - e^{-c}) \}}{Ab} \right] \tag{5.1}$$

where a, Ab and c are the parameters of the yield equation, $y = [1 + Ab\,(1 - e^{-c})]^{-a}$. The yield equation assumes that physical defects on a chip conform to the negative binomial distribution and the number of faults caused by each defect are independent A comparison with Stapper's yield equation [Stappe75] gives the expression:

$$Ab\,(1 - e^{-c})\,a = A\overline{D} \tag{5.2}$$

where A (not to be mistaken with Ab) is the chip area and $\overline{D}$ is the average defect density

Substituting equation parameters estimated from actual wafer test data, for a given fabrication process and design style, and using equations 5 1 and 5 2, it is shown [SetAgr84] that as chip area increases the required fault coverage for a given reject ratio (defect level) converges to a fixed value Figure 5 4 shows the variation of the required fault coverage against normalized circuit area for given defect levels, ranging from 2% to 10%, based on experimentally determined parameters [SetAgr84] The normalizing area is the area of the chip whose test results were analyzed to estimate parameters of the yield equation. It


1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
2
4
6
8
10
12
Fault Coverage
Normalized Area
d=0.002
d=0.004
d=0.006
d=0.008
d=0.010

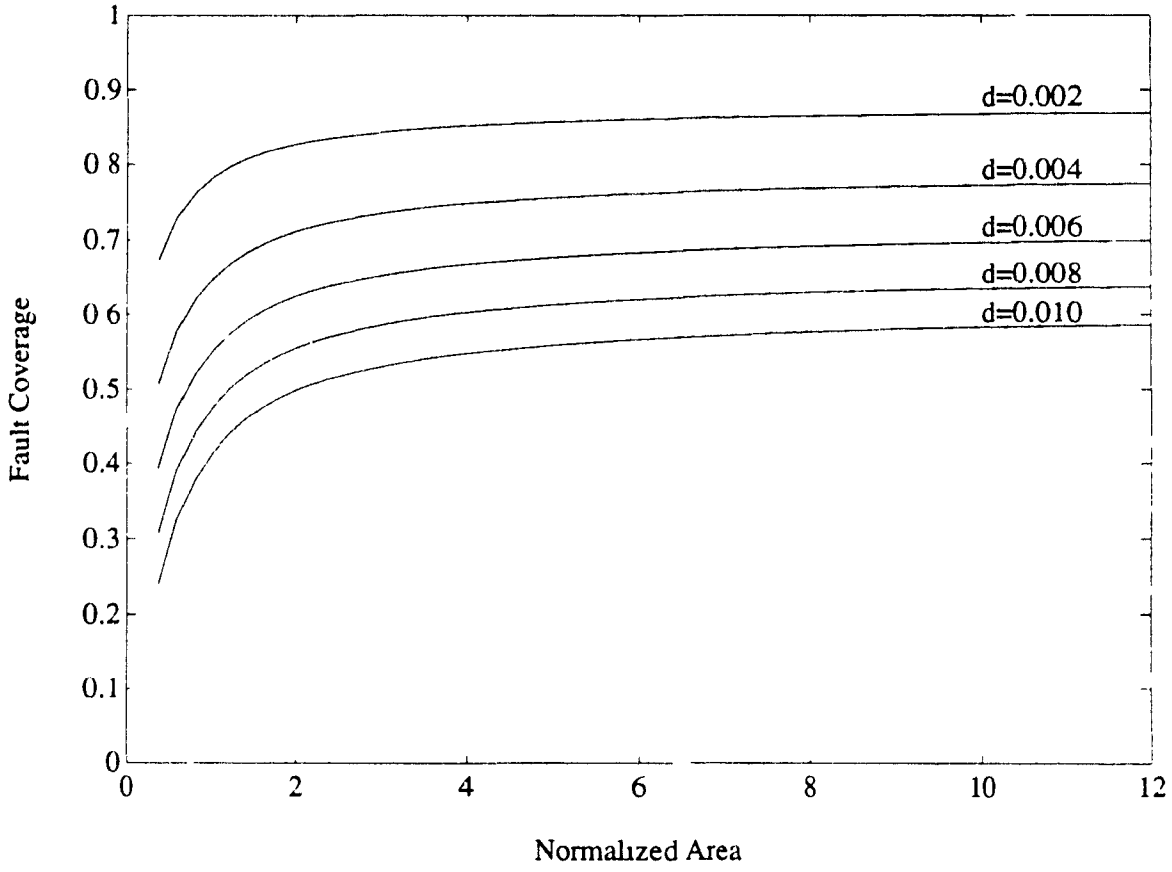


Figure 5.4 Fault coverage versus normalized area for different defect levels

contained approximately 2700 transistors The chip designed to be used as a test vehicle for the work in this thesis contains approximately 7500 transistors

Looking at the above results another way, for a given fault coverage, the defect level of a test set converges to a fixed value as the circuit area increases. For circuit sizes larger than a particular *threshold*, therefore, the defect levels of test sets providing the same fault coverage remain almost the same, irrespective of the area of the chip.

Chapter 6 Results

The test chip described in chapter 2 was designed and fabricated using an experimental CMOS 1.5μ double-layer metal (DLM) process A number of wafers were fabricated yielding a statistically significant sample of test devices

All fabricated devices were initially tested at "slow speed" using the extended fault test set (EFTS) and those which failed the test were subsequently analyzed The analysis consisted of:

- automated diagnosis based on the stuck-at and transition fault models,
- validation of the results of diagnosis of selected chips representative of major failure classes, and
- experiments to evaluate the effectiveness of:
    - the stuck-at fault model, and
    - test sets generated to cover stuck-at faults

In addition, a smaller sample of devices was tested at a higher speed (using the EFTS) and subjected to transition fault diagnosis

This chapter details the actual testing, diagnosis and validation operations performed on the batch of fabricated devices and presents the results collected during the course of the study

6.1 Sequence of Tests

An ASIX [4] desk-top tester was used to perform tests on all fabricated devices While some devices were packaged and tested as such, an automatic wafer-prober, serving as a front-end to the tester, was used to probe devices on unpackaged wafers

Each chip was initially subjected to a *continuity* test to verify the protection diodes on all pins of the device Only chips passing the continuity test were tested further Subsequent testing on each chip consisted of applying a sequence of 11 different test sets — one for verification of the set of tristate drivers (section 3 5) in the chip and one each for each of the 10 different logic blocks

As described in chapter 3, the EFTSs for each of the DS blocks (DS1–DS5) consist of 2528 vectors Those for each of the DZ blocks (DZ1–DZ4) consist of 2544 vectors The ASIX tester, however, has a limitation in that it can only log a maximum of 256 faulty responses for each "stage" of a test Since fault diagnosis requires a data-log of *all* faulty responses from the circuit under test, the test set for each logic block was split up into ten stages so that, accounting for the worst case, all faulty responses could be logged The original test sets were divided (into stages) in a manner such that *initializing* and *testing* vectors (for pairs or trios) did not stretch across boundaries of test stages to prevent inadvertent insertion of de-initializing vectors in going from one stage of the test to the next It also ensured a uniform *time* between application of initializing and testing vectors

All chips in the initial batch of testing were tested with the EFTS at a rate of $2MHz$ The sampling strobe for the output data was set at $350ns$, or 70% of the test speed The strobe placement, at 350ns, was approximately 10 times slower than the worst-case delay expected through the circuit. Chips from one wafer were later tested *at-speed* with the sampling strobe placed close to the mean delay of the circuit

All faulty responses of all faulty chips (other than those with continuity failures) were logged by the tester, their files translated to ASCII formats, from the default tester format, and the ASCII files transferred to a network of Apollo workstations on which the automated diagnosis was performed Each log file thus generated — typically one per

---

[4] ASIX Systems Corporation

wafer — contains information in the form of a header, clearly identifying the faulty block(s) (within a faulty chip) and all their faulty responses in the format.

$$< faulty\ vector\ number > \quad < expected\ response > < observed\ response$$

The data-log files for all nine wafers took approximately 180 megabytes of storage space

6.2 Diagnosis Package

A package of automated diagnostic tools was developed based on the methodology discussed in chapter 4 The package includes programs to

- Perform fault diagnosis,
- Determine the neighbourhood of unmodelled fault sites in cases where no faults can be diagnosed,
- "Trace" specific faults to determine which vectors (vector pairs) they are *cleared* on from the list of suspected faults for a given faulty block,
- Generate the stuck-at fault dictionary from the output of the fault simulator,
- Generate the transition fault dictionary from the stuck-at fault dictionary, and
- Generate the database used by the diagnosis program to implement fault list pruning based on structural dominance relationships

The entire package of diagnostic tools is written in the "C" programming language and contains approximately 4500 lines of code

Programs in the fault diagnosis package work directly on the data generated by the tester As described previously, this data consists of a listing of all *failing* vectors with the expected and observed responses on each primary output Results of the diagnosis are produced in two separate files The format of the information produced in each file is described in the following two sections

6.2.1 Exact Fault Location

The first file produced by the diagnosis program contains a list of diagnosed faults for every faulty block (of every faulty chip) The location of each suspected fault is specified

exactly by its numeric ID which translates into a unique circuit line and fault type In addition, the file contains a list of all primary outputs of the circuit on which faulty responses were observed For example, the excerpt

Results of chip number 23
****
Block DS1
List of primary output(s) with faulty responses
1
List of stuck-at faults not cleared·
204 1s12 s-a-1

taken from a file produced by the diagnosis program indicates that faulty responses were observed only on primary output number 1 for block $DS1$ of chip number 23 and the diagnosed fault for the block is number 204 which is a stuck-at-one fault on line $1s12$

6.2.2 Compressed Report

For each faulty block, the second file produced by the diagnosis program contains the following information

- The total number of diagnosed faults,
- The total number of faulty responses,
- The total number of asymmetric faults (for transition faults),
- The number of diagnosed faults propagating to each primary output,
- The number of faulty responses per output, and
- The number of asymmetric faults propagating to each primary output (for transition faults)

Chip number 23, Block DS1

FRO	*0*	*178*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*
SFO	*0*	*2*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*
AFO	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*

TFV 178 TSF 2 TAF 0

Table 6.1 Excerpt from file produced by diagnosis program

Term	Description
FRO	Faulty Output
SFO	Suspected Faults per Output
AFO	Asymmetric Faults per Output
TFV	Total number of Faulty Vectors
TSF	Total number of Suspected Faults
TAF	Total number of Asymmetric Faults

Table 6.2 Description of terms used in file produced by diagnosis program

Table 6 1 shows an excerpt taken from a file produced by the diagnosis program Each column in in the information produced in the file corresponds to a specific primary output, starting with output 0 on the left Table 6 2 describes the terms used in the file The example indicates that primary output number 1 accounted for all 178 faulty responses and that there are two transition faults suspected on the same output (1), neither of which is asymmetric.

The format of the data in the compressed report makes it easy for further parsing for the compilation of gross statistics Faulty blocks (or chips) can be sorted on the basis of any combination of parameters on which information is stored For instance a small utility program can easily extract the IDs of all faulty blocks on which the number of suspected faults falls within a certain range In addition, a general purpose parser was developed which can sort faulty blocks and faulty chips on the basis of ranges of a parameter (or combination of parameters) listed in table 6 2 The parser can sift through the huge amounts of information produced as a result of the diagnosis and extract relevant portions on the basis of user defined *queries*

6.3 Diversity of Results

The compressed diagnostic report generated by the diagnosis program for faulty blocks can be easily interpreted to extract basic information regarding the nature and multiplicity of faults

The examples shown in tables 6 3 through 6 8 illustrate the diagnostics of some typical faulty blocks As may be inferred from the tables the results are quite varied and imply a large variety of modes of failure

Chip number 23, Block DS1
FRO. 0 178 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SFO. 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TFV. 178 TSF 1

Table 6.3 Example of single stuck-at fault

Chip number 6, Block DS3
FRO 0 0 0 0 8 352 0 0 0 0 0 2 74 0 0 0
SFO 0 0 0 0 1 2 0 0 0 0 0 1 2 0 0 0
TFV 360 TSF 3

Table 6.4 Example of multiple stuck-at faults

Chip number 8, Block DS1
FRO. 112 460 0 0 386 1361 176 1170 96 30 72 76 462 518 728 152
SFO 0 5 0 0 0 2 0 4 1 2 1 0 2 2 2 2
TFV 2455 TSF 13

Table 6.5 Example of a very large number of faults (catastrophic failures)

Chip number 21, Block DS3
FRO 0 28 0 0 0 8 0 0 22 0 0 0 2 0 0 0
SFO 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0
TFV 54 TSF. 1

Table 6.6 Example of non-modelled faults with stuck-ats

Chip number 26, Block DS2
FRO 0 0 0 0 29 0 0 0 0 14 0 0 0 0 0 0
SFO 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
TFV 43 TSF. 0

Table 6.7 Example of non-modelled faults

6.4 Validation

While the results of diagnosis performed on faulty chips demonstrated the ability of the method to provide a high diagnostic resolution -- for example in cases where single faults, localised to single circuit lines, were diagnosed — their validation is essential to developing any degree of confidence in the results and in the methodology on which the

Chip number 65, Block DS3

FRO.	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*176*	*0*	*0*	*0*	*0*	*70*	*0*	*0*	*0*
SFO	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*11*	*0*	*0*	*0*	*0*	*11*	*0*	*0*	*0*
AFO.	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*3*	*0*	*0*	*0*	*0*	*1*	*0*	*0*	*0*

TFV. 246 TSF 17 TAF 3

Table 6.8 Example of asymmetric delay faults

experimental work is based

It is clearly not practical, if at all possible, to validate the results (of diagnosis) of *all* faulty blocks because of the time-consuming nature of the process For the purpose of validation, therefore, faulty blocks were classified into two major categories — blocks with a very large number of diagnosed faults and blocks with exactly one diagnosed stuck-at fault — and a number of faulty blocks picked at random from either category to be validated

Seven faulty devices were thus subjected to verification For the 3 devices falling under the first category, verification was easily performed using an optical microscope Massive failures could actually be seen at the sights suspected The failures appeared to have resulted both from substrate impurities, appearing as dark spots, and surface deformities Some of the surface deformities appeared to have been caused by post-fabrication handling — as a result of the bonding and packaging processes — that resulted in long scratches across the surface of the chip

The optical microscope was not effective in verifying results of the diagnosis where only single stuck-at faults were suspected Four faulty devices were subsequently de passivated and probed with an electron-beam voltage-contrast prober (EBVCP)

The EBVCP works on the same principle as a scanning electron microscope (SEM) Essentially, a beam of primary electrons, accelerated through a potential of about $1kV$, *scans* the area to be probed The angle of reflection of the secondary electrons produced serves as a measure of the relative potential of the area on which the primary beam of electrons was incident Tracks with high voltage levels (logic 1s) show up dark in contrast to other tracks with low voltages (logic 0s)

The electron-beam probing was performed with the faulty devices being stimulated with the test set in a tight loop In all 4 cases, results of the diagnosis exactly matched the results of the probe In one case, for example, it could be seen that one of the fanout

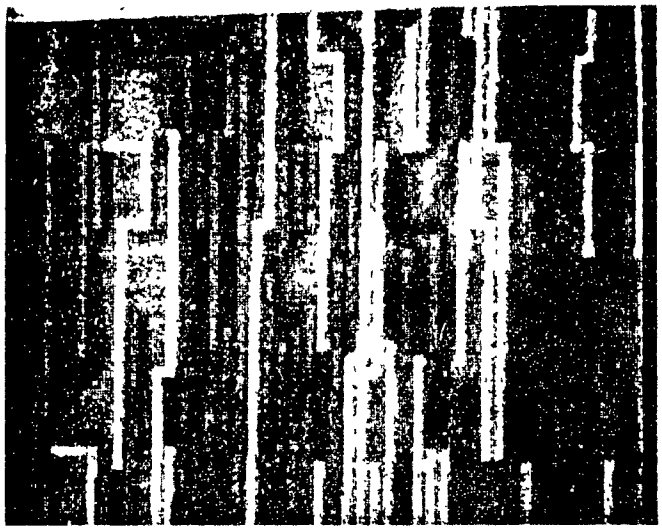


Figure 6.1 Picture taken through EBVCP

branches of a stem was stuck at a fixed value even though the stem itself and the other fanout branch was functioning properly The fault, in this case, was traced to a missing *via* connection between the 2 metal layers Figure 6 1 is a reproduction of a picture taken through the EBVCP The picture shows an instantaneous snap-shot of the voltage levels on metal tracks forming the top layer of the test chip Tracks appearing bright indicate low voltage levels (logic 0s) Given a layout of the chip, the tracks can be correlated with the circuit netlist lines

6.5 Sequence of Experiments

Chips which failed the tristate test were discarded and only those with functional tristate drivers, determined on the basis of the tristate test, were subjected to further analysis The results were compiled on a *per-block* basis so as to provide a platform to compare the behaviour of circuits (blocks) designed with different cell libraries In the results included in this section, however, all blocks are treated as similar entities As described earlier, each logic block consists of combinational logic containing approximately 750 transistors The test chip contains approximately 7500 transistors

The experimental work was divided into two phases The first phase consisted of the analysis of results of blocks detected faulty with the "slow" EFTS The focus of the analysis was to perform automated diagnosis on faulty blocks, on the basis of the stuck-at fault model, and to get measures of its (the fault model's) effectiveness In addition, faulty

blocks were subjected to diagnosis on the basis of the transition fault model even though a slow strobe was used during the test The second phase of experimentation dealt with transition fault analysis on devices found faulty with the *at-speed* test

6.6 Analysis of "slow" EFTS results

This section presents experimental results from the automated diagnosis of 970 faulty blocks. The results are broadly classified into two groups — those obtained from stuck-at fault analysis and others obtained from transition fault analysis

6.6.1 Stuck-at fault anal' 's

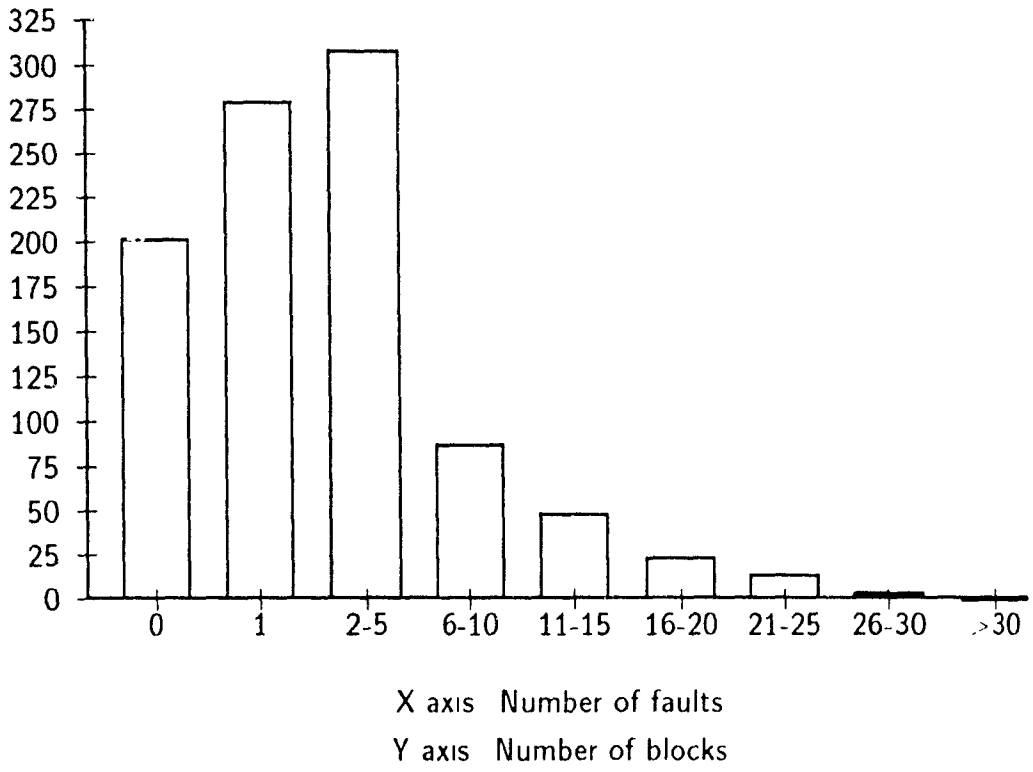
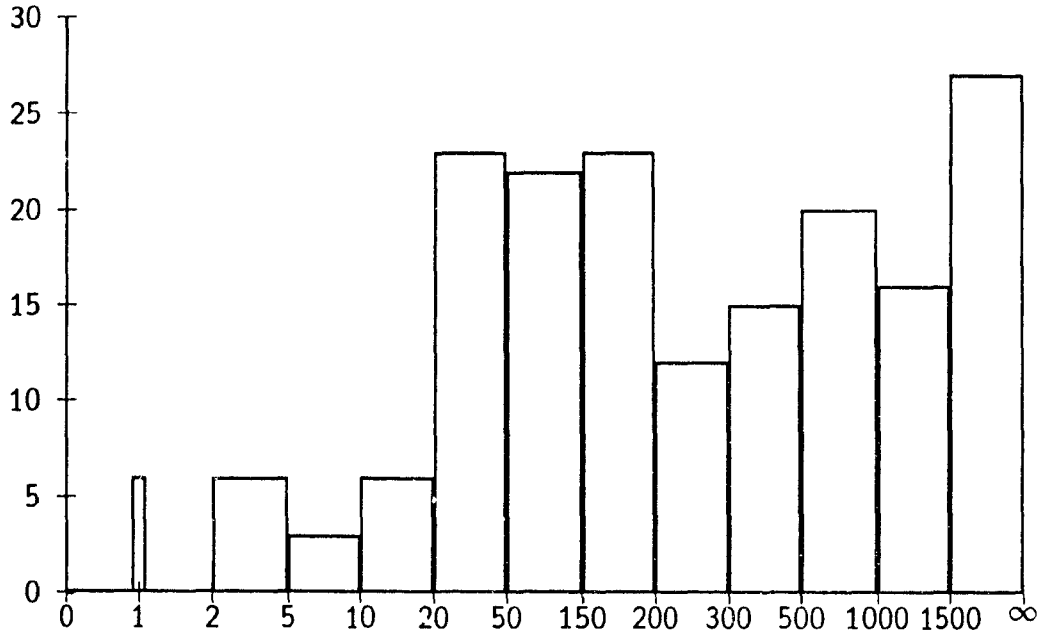


Figure 6.2 Number of stuck-at faults per faulty block

Figure 6 2 shows the distribution of faulty blocks, sorted according to the number of stuck-at faults diagnosed on them As can be seen from the bar graph, a relatively large number (202 out of 970 or 20.8%) of faulty blocks have no stuck-at faults diagnosed on them Looking at it another way, there are *no* faults on any of these faulty blocks,

consistent with the single stuck-at fault model. The remaining 79.2% of faulty blocks have at least one stuck-at fault suspected even though they may contain non-modelled faults in addition to the stuck-at faults diagnosed on them. 28.9% of faulty blocks have exactly one suspected stuck-at fault while another 31 9% have between 2 and 5 (both inclusive) stuck-at faults diagnosed The number of faulty blocks with a larger number of diagnosed faults is lower, only 18 4% having between 6 and 30 (both inclusive) faults There are no blocks with over 30 stuck-at faults diagnosed This, however, cannot be interpreted to project that most blocks only contain spot defects since only the *most* structurally dominant of a set of originally suspected faults are finally listed, as described earlier in section 4.2.1



X axis. Number of faulty vectors
Y axis· Number of faulty blocks
Note Numbers lying exactly on the boundaries of intervals are placed within boxes to their left

Figure 6.3 Blocks with no stuck-ats sorted by number of faulty vectors

Figure 6 3 shows the distribution of faulty blocks with no suspected stuck-at faults, sorted according to the number of faulty vectors observed on them Clearly, only these blocks are likely to contribute to the defect level of a complete single stuck-at test set since all other faulty blocks have at least one diagnosed stuck-at fault

As shown in the figure, there are 6 faulty blocks with only single failing vectors

Since all single stuck-at faults are detected at least twice for all blocks (section 3.4), test sets can be constructed for these 6 cases which would provide a 100% coverage of single stuck-at faults but still "pass" the faulty circuits

It is also possible, though not guaranteed, to construct test sets complete for single stuck-at faults which would "pass" other faulty blocks with no diagnosed stuck-at faults which had a relatively small number of failing vectors in the complete diagnostic test

Clearly, the larger the number of faulty responses recorded in the complete test, the lesser the likelihood of "other" crucial faults *escaping* detection by a test set with complete single stuck-at fault coverage.

6.6.1.1 Effectiveness of the stuck-at fault model and stuck-at test sets

It can be seen from the results presented in the previous section that the single stuck-at fault model is clearly inadequate in terms of defect modelling since there is an unacceptably large number of faulty circuits (20 8% of all faulty blocks) where no stuck-at faults could be diagnosed. It is of more interest, however, to get a measure of the defect level of a test set designed to cover all single stuck-at faults As described in chapter 5, this depends not only on the adequacy of the fault model (in this case of the single stuck-at fault model) but also on the automated tools used to generate the test set since the sequence and redundancy of the test set would tend to affect its windfall coverage of other faults.

Using the approach described in section 5 3 the stuck-at fault model and a test set generated to cover all stuck-at faults were evaluated

In the first experiment, the failing vectors for each faulty block tested with the EFTS were *deleted* from the test set The block in question was then fault simulated with the modified test It was found that out of the 202 faulty blocks for which no fault(s) could be *diagnosed* on the basis of the single stuck-at fault model, test sets providing a 100% coverage of single stuck-at faults could be constructed for 132 (or 65 3%) which would not even *detect* their faulty behaviour Looking at the results another way, for 13.58% of *all* faulty blocks (20.8% · 65 3%), test sets can be devised which would provide complete coverage of stuck-at faults and still not detect them

Wafer	Good Blocks (G)	Escaping Blocks (F)	Defect Level $(\frac{F}{F+G})\%$
1	718	2	0.278
2	806	2	0.248
3	895	7	0.776
4	1020	1	0.098
5	978	0	0 000
6	913	0	0.000
7	973	0	0.000
8	994	3	0.301
9	859	7	0 808

Table 6.9 Experimentally determined defect level

In the second experiment, all blocks [5] were subjected to a sequence of two tests using the EFTS, and using a complete test for single stuck-at faults, generated with the help of a random pattern-generator and fault simulator. It was found that in addition to the 970 blocks originally found faulty with the application of the EFTS, 5 other blocks were found faulty with the re-application of the same test, at the same speed, during the present experiment, making for a total of 975 faulty blocks The 5 additional blocks found faulty, therefore, contained *intermittent* faults Out of the 975 established faulty blocks, 22 (or 2 26%) passed the shorter, but complete, test for single stuck-at faults Interestingly, it was also found that 1 additional block passed the EFTS but failed the shorter test Table 6.9 shows the results of the experiment on a wafer-by-wafer basis. The column labelled "good blocks" refers to the number of blocks found functional as a result of the initial diagnostic test while the column labelled "escaped blocks" refers to the number of blocks found faulty with the EFTS but which were not subsequently detected by the stuck-at fault test The mean defect level, of the complete stuck-at test set generated randomly, across all wafers, $\overline{DL}$, $(\frac{\sum DL_i}{n})$ was determined to be 0.279% while its standard deviation $(\frac{\sum (DL_i - \overline{DL})^2}{n})$ was determined to be 0.297

Assuming the defect level to be normally distributed, bounds for its variation can be determined for a given confidence level [Kreysz71] Using the data at hand, the defect

[5] On chips which passed their *tristate test*

level was determined to be bounded by:

$$0.085 \leq DL(\%) \leq 0.473$$

with a confidence level of 95%

6.6.1.2 Nature of faults on escaping blocks

Out of the 22 blocks that escaped the shorter stuck-at test, 4 blocks contained at least one transition fault, determined as a result of transition fault diagnosis, even though the initial diagnostic test was not conducted at *circuit-speed*), 5 others contained intermittent faults, as described previously, and the remaining 13 contained other non-modelled faults only

6.6.2 Transition fault analysis on the basis of *slow* test results

Automated transition fault diagnosis was performed on all faulty blocks even though the sampling strobe was deemed to be excessively slow for this purpose. The results revealed that in addition to the 79.2% of all faulty blocks which contained at least one stuck-at fault, 6.9% of all faulty blocks were diagnosed as containing asymmetric transition faults only In other words 13.9% of all faulty blocks could not be diagnosed on the basis of the transition and single stuck-at fault models Table 6 10 summarizes the results

	Number	Percent
Faulty	970	100.00
Stuck-at faults	768	79 2
Transition faults	67	6 9
Non-modelled faults	135	13.9

Table 6.10 Comparative analysis of faults with *slow* diagnostic test

In table 6 10, the "Stuck-at fault" row lists the number of blocks which were diagnosed as having at least one stuck-at fault, the "Transition fault" row lists those which have no stuck-at faults but at least one transition fault and, the "Non-modelled fault" row lists those blocks which only contain faults not consistent with either model

6.7 Transition Fault Analysis based on "At-Speed" Test

In addition to the testing and diagnosis performed for stuck-at fault analysis, one packaged wafer was re-tested at a higher speed and diagnosed for transition faults All devices on the wafer were initially characterized for their worst-case delays Information was logged separately for DS and DZ blocks since two different cell libraries, with different characteristics were used in their design The mean and standard deviation of the delays for both types of blocks were calculated and devices on the wafer re-tested with the sampling strobe set at a value of 3 standard deviations beyond the mean delay of the block type under test

In addition to the blocks already determined faulty on the basis of the initial test using the EFTS with a "slower" strobe placement, 10 other blocks were found faulty All newly failing blocks were subjected to automated diagnosis They were also subsequently characterised (on the tester) for their worst-case delays It was found, as expected, that no stuck-at faults could be diagnosed on the newly detected faulty blocks Out of the ten such blocks, however, transition faults could be diagnosed only on five In all five cases, asymmetric transition faults were diagnosed

Block	Failing Vectors	Faulty Outputs	Transition Faults	Delay (times mean)
1	393	1	1	1.35
2	61	1	0	2.71
3	1	1	0	1 32
4	11	4	0	1 02
5	1	1	0	4.57
6	126	10	5	1 61
7	16	1	0	1.33
8	78	1	1	2.42
9	78	1	1	2.55
10	200	1	1	2.25

Table 6.11 Results of blocks found faulty with high-speed test

The results of the transition fault analysis are presented in table 6 11 As can be seen from the table, no transition faults could be diagnosed on blocks which failed

on a relatively few number of vectors — blocks 2, 3, 4, 5 and 7. Single, asymmetric transition faults were diagnosed on four other blocks while 5 asymmetric transition faults were diagnosed on block number 6 The last column in the table shows the experimentally determined worst-case delay through the faulty blocks as a multiplication factor of the sampling strobe delay[6] for the high-speed test Interestingly, while block number 5 logged only one faulty response with the high-speed test, the worst-case delay through it was observed to be extraordinarily high

[6] Experimentally determined *average* worst-case delay across similar block-types initially characterised plus three standard deviations

Chapter 7 Conclusions

A methodology for the experimental evaluation of fault models, using fault diagnosis as the basic approach, has been developed The methodology includes a way of determining the defect level of test sets in addition to determining the adequacy of the fault models used to generate them The operations of diagnosis have been implemented in a package of automated tools The tools can be used for periodically monitoring any given fabrication process by performing automated diagnosis on faulty devices They can also be used to help locate unmodelled fault sites, leading to the generation of more adequate fault models

A purely combinational test chip was designed and fabricated specially to capture the characteristics of the CAD tools, cell libraries and fabrication process used in its development. The results of the automated analysis, performed on the test chip are not inconsistent with those published in [ShMaFe85] Results of experiments with the test chips indicate that 20 8% of all faulty blocks had no stuck-at faults According to results given in [ShMaFe85], 36% of all *faults* are of the non stuck-at variety The results, however, can be compared only in the context of a common denominator For instance, the percentage of all faulty blocks with no stuck-at faults do not account for *all* unmodelled faults present on all faulty blocks There were still other faulty blocks with unmodelled faults which, *in addition*, also had stuck-at faults. The number of non stuck-at faults in faulty blocks as a percentage of the total number of faults, therefore, can be expected to be larger than 20.8%

In the future, information on the location of unmodelled fault sites, generated by the diagnosis package, can be used to determine the exact nature of such defects using an EBVC prober The information can then be used to define algorithms to generate appropriate tests for such faults and/or to suggest modifications to current (stuck-at) ATPG algorithms to better cover them

Appendix A Test Sets for Complex Gates

A.1 2-2 OR-AND-INVERT Gate

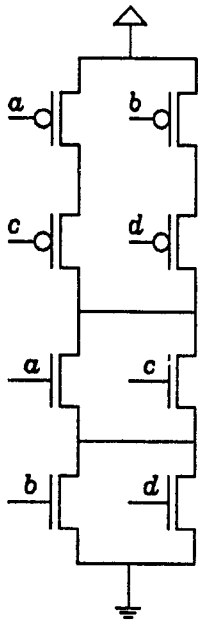


Figure A.1 2-2 OR-AND-INVERT Gate

Stuck-Open Test for Transistor		Inputs			
		a	b	c	d
Pullup	a	1	1	1	1
		1	0	1	1
		0	0	1	1
	b	1	1	1	1
		1	1	1	0
		1	1	0	0
	c	1	1	1	1
		0	1	1	1
		0	0	1	1
	d	1	1	1	1
		1	1	0	1
		1	1	0	0
Pulldown	a	0	0	0	0
		0	0	1	1
		1	0	1	1
	b	0	0	0	0
			1	0	0
		1	1	1	0
	c	0	0	0	0
		0	0	1	1
		0	1	1	1
	d	0	0	0	0
		1	1	0	0
		1	1	0	1

Table A.1 Diagnostic test set for 2-2 OR-AND-INVERT

A.2 1-3 OR-AND-INVERT Gate

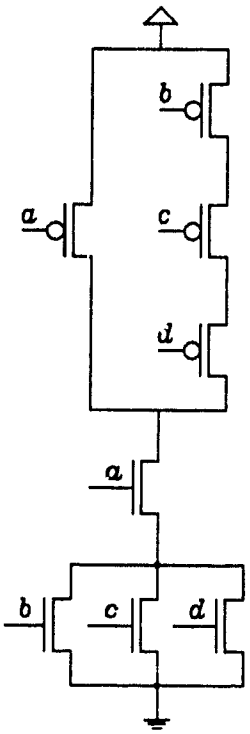


Figure A.2 1-3 OR-AND-INVERT Gate

Stuck-Open Test for Transistor		Inputs			
		a	b	c	d
Pullup	a	1	1	1	1
		0	1	1	1
	b	1	1	1	1
		1	1	0	0
		1	0	0	0
	c	1	1	1	1
		1	0	1	0
		1	0	0	0
	d	1	1	1	1
		1	0	0	1
		1	0	0	0
Pulldown	a	0	0	0	0
		0	1	1	1
		1	1	1	1
	b	0	0	0	0
		1	0	0	0
		1	1	0	0
	c	0	0	0	0
		1	0	1	1
		1	0	1	0
	d	0	0	0	0
		1	0	0	0
		1	0	0	1

Table A.2 Diagnostic test set for 1-3 OR-AND-INVERT

A.3 1-1-2 OR-AND-INVERT Gate

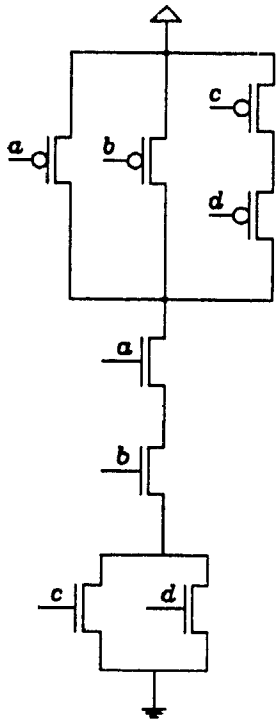


Figure A.3 1-1-2 OR-AND-INVERT Gate

Stuck-Open Test for Transistor		Inputs			
		a	b	c	d
	a	1	1	1	1
		0	1	1	1
	b	1	1	1	1
		1	0	1	1
	c	1	1	1	1
		1	1	1	0
		1	1	0	0
	d	1	1	1	1
		1	1	0	1
		1	1	0	0
Pulldown	a	0	0	0	0
		0	1	1	1
		1	1	1	1
	b	0	0	0	0
		1	0	1	1
		1	1	1	1
	c	0	0	0	0
		1	1	0	0
		1	1	1	0
	d	0	0	0	0
		1	1	0	0
		1	1	0	1

Table A.3 Diagnostic test set for 1-1-2 OR-AND-INVERT

A.4 2-2 AND-OR-INVERT Gate

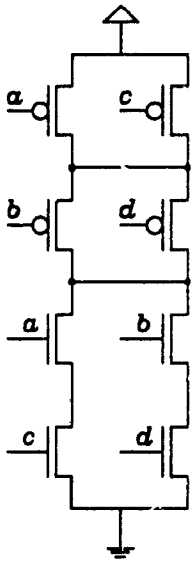


Figure A.4 2-2 AND-OR-INVERT Gate

Stuck-Open Test for Transistor		Inputs			
		a	b	c	d
Pullup	a	1	1	1	1
		1	1	0	0
		0	1	0	0
	b	1	1	1	1
		0	0	1	1
		0	0	0	1
	c	1	1	1	1
		1	1	0	0
		1	0	0	0
	d	1	1	1	1
		0	0	1	1
		0	0	1	0
Pulldown	a	0	0	0	0
		0	1	0	0
		1	1	0	0
	b	0	0	0	0
		0	0	0	1
		0	0	1	1
	c	0	0	0	0
		1	0	0	0
		1	1	0	0
	d	0	0	0	0
		0	0	1	0
		0	0	1	1

Table A.4 Diagnostic test set for 2-2 AND-OR-INVERT

A.5 1-3 AND-OR-INVERT Gate

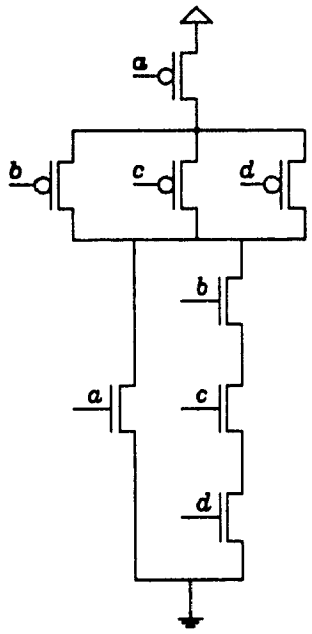


Figure A.5 1-3 AND-OR-INVERT Gate

Stuck-Open Test for Transistor		Inputs			
		a	b	c	d
Pullup	a	1	1	1	1
		1	0	0	0
		0	0	0	0
	b	1	1	1	1
		0	1	1	1
		0	0	1	1
	c	1	1	1	1
		0	1	1	1
		0	1	0	1
	d	1	1	1	1
		0	1	1	1
		0	1	1	0
Pulldown	a	0	0	0	0
		1	0	0	0
	b	0	0	0	0
		0	0	1	1
		0	1	1	1
	c	0	0	0	0
		0	1	0	1
		0	1	1	1
	d	0	0	0	0
		0	1	1	0
		0	1	1	1

Table A.5 Diagnostic test set for 1-3 AND-OR-INVERT

References

[Abraha86] Jacob A Abraham, "Fault Modelling in VLSI", *VLSI Testing, Editor: T.W. Williams, North-Holland Press*, 1986.

[AgaFun81] V.K Agarwal, A S. Fung, "Multiple Fault Testing of Large Circuits by Single Fault Test Sets", *IEEE Trans. On Computers, Vol. C-30*, pp. 855–865, Nov. 1981.

[BasCou84] D. Baschiera, B Courtois, "Testing CMOS: A Challenge", *VLSI Design, Vol. No. 10*, pp 58–62, Oct. 1984.

[BaMcSa87] P H Bardell, W H McAnney, J Savir, *Built-In Self-Test for VLSI*, Wiley-Interscience, New York, 1987

[BhMuHa89] D. Bhattacharya, B T Murray, J P. Hayes, "High-Level Test Generation for VLSI," *IEEE Computer*, Vol. 22, No. 4, pp. 16–24, April 1989.

[BKLNPW82] A K. Bose, P. Kozak, C Y. Lo, H.N. Nham, E Pacas-Skewes, K. Wu, "A Fault Simulator for MOS LSI Circuits", *Proc* 19^{th} *Design Automation Conference*, pp. 400–409, June 1982

[BreFri76] M.A Breuer, A.D Friedman, "Diagnosis and Reliable Design of Digital Systems", *Computer Science Press, Rockville, MD*, 1976.

[Chandr83] R Chandramouli, "On Testing Stuck-Open Faults", *Digest of Papers*, 13^{th} *FTCS*, pp 258–265, June 1983.

[CoFGJM87] H Cox, K. Fadlallah, S. Gaiotti, A. Jain, M. Malowany, R. Tio, B. Mandava, J. Rajski, N C. Rumin, "A Processing Element for a Reconfigurable Massively-Parallel Processor", *Proc Canadian Conference on VLSI*, pp. 241–246, Oct. 1987

[ColvAR88] H. Cox, A Ivanov, V.K. Agarwal, J. Rajski, "On Multiple Fault Coverage and Aliasing Probability Measures," *Proc ITC-88*, pp 314–321, Washington DC, Sept 1988.

[CoxRaj88a] H. Cox, J Rajski, "A Method of Test Generation and Fault Diagnosis" *IEEE Transactions on CAD, vol 7, No 7* pp 813-833, July 1988

[CoxRaj88b] H Cox, J Rajski, "Stuck-Open and Transition Fault Testing in CMOS Complex Gates" *Proc International Test Conference* pp 688-694, Sep. 1988

[DeBeTh88] R. Dekker, F. Beenker, L. Thijssen, "Fault Modelling and Test Algorithm Development for static Random Access Memories" *Proc. International Test Conference* pp 343-352, Sep 1988

[Eldred59] R.D Eldred, "Test Routines Based on Symbolic Logical Statements," *Journal of the ACM*, Vol 6, pp 33-36, 1959

[Elziq82] Y.M. Elziq, "Automatic Test Generation for Stuck-Open Faults in CMOS VLSI", *Proc. 18th Design Automation Conference*, pp 347–354, June 1982

[[ElzClo81] Y.M Elziq, R J Cloutier, "Functional Level Test Generation for Stuck-Open Faults in CMOS VLSI", *Proc International Test Conference*, pp 536–546, Oct 1981

[FujToi82] H. Fujiwara, S Toida, "The Complexity of Fault Detection Problems in Combinational Logic Circuits," *IEEE Trans Comp* , Vol C-31, pp 555-560, June 1982

[HugMcC86] J.L.A. Hughes, E.J McCluskey, "Multiple Stuck-At Coverage of Single Stuck-At Fault Test Sets", *Proc. International Test Conference*, pp 368–374, Sep 1986

[JacBis87] J Jacob, N N Biswas, "GTBD Faults and Lower Bounds on Multiple Fault Coverage of Single Fault Test Sets", *Proc International Test Conference*, pp. 849–855, Sep. 1986

[JaiAgr83] S.K. Jain, V D Agrawal, "Test Generation for MOS Circuits Using D-Algorithm", *Proc. 20th Design Automation Conference*, pp 64–70, June, 1983

[JaiAgr85] S.K Jain, V.D. Agrawal, "Modeling and Test Generation Algorithms for MOS Circuits", *IEEE Transactions on Computers, Vol. C-34, Number 5*, pp 426–433, May 1985.

[KaShKa89] J. Kato, T. Shimono, M. Kawal, "Fault Diagnosis Based on Post-Test Fault Dictionary Generation", *Proc International Test Conference*, pp 940 (poster session), Aug. 1989

[Kreysz71] E Kreyszig, *Advanced Engineering Mathematics*, Wiley Eastern Private Limited, 2nd edition, 1971

[LesShe80] J.D Lesser, J J Shedletsky, "An Experimental Delay Test Generator for LSI Logic", *IEEE Trans on Computers*, pp. 235–246 March 1980

[LevMen86] Y Levendal, P R Menon, "Transition Faults in Combinational Circuits Input Transition Test Generation and Fault Simulation", *Digest of papers*, 16th *FTCS*, pp

278–283, July, 1986.

[MaaRaj88] F. Maamari, J. Rajski, "A Fault Simulation Method Based on Stem Regions," *Proc. ICCAD 1988*, pp. 170-173, Nov. 1988.

[Malaiy84] Y.K Malaiya, "Testing Stuck-On Faults in CMOS Integrated Circuits", *Proc. IC-CAD*, pp 248–250, 1984

[MalNai89] W. Maly, S.B Naik, "Process Monitoring Oriented IC Testing", *Proc. International Test Conference*, pp 527–532, Aug 1989.

[Maly87] W. Maly, "Realistic Fault Modeling for VLSI Testing " 24^{th} *Design Automation Conference Proceedings* pp. 173-180, June 1987.

[McASav87] W.H. McAnney, J Savir, "There is Information inFaulty Signatures," *Proc International Test Conference*, pp 630-636, Washington DC, Sep 1987

[Mei74] K.C.Y. Mei, "Bridging and Stuck-At Faults", *IEEE Trans. on Computers, Vol. C-23*, pp 720–727, July 1974.

[Poage62] J.F. Poage, "Derivation of Optimum Tests to Detect Faults in Combinational Circuits", *Proc. Symposium on Mathematical Theory of Automata*, pp. 483–528, April 1962 (Polytechnic Press 1963)

[Pradha86] D.K. Pradhan, (Editor), J.A. Abraham, V.K Agarwal, B. Bose, Y. Levendal, E.J. McCluskey, P.R Menon, J. Metzner, Yoshihiro Tohma (Contributors), "Fault-Tolerant Computing, Volume 1", *Prentice-Hall*, 1986.

[RajCox86] J. Rajski, H Cox, "Stuck-Open Fault Testing in Large CMOS Networks by Dynamic Path Tracing", *Proc International Conference on Computer Design*, pp 252–255, Oct. 1986

[RajTys85] J. Rajski, J. Tyszer, "Combinatorial Approach to Multiple Contact Faults Coverage in Programmable Logic Arrays", *IEEE Transactions on Computers Vol C-34, Number 6*, pp 549–553, June, 1985.

[Ravi81] K. W Ravi, "Imperfections and Impurities in Semiconductor Silicon", *John Wiley and Sons, New York*, 1981

[ReReAg84] S M Reddy, M K Reddy, V D Agrawal, "Robust Tests for Stuck-Open Faults in CMOS Combinational Logic Circuits", *Digest of Papers*, 14^{th} *FTCS*, pp 44–49, June, 1984.

[ReAgJa84] S.K. Reddy, V.D. Agrawal, S.K. Jain, "A Gate-Level Model for CMOS Combinational Logic Circuits With Application to Fault Detection", *Proc 21st Design Automation Conference* pp. 504–509, June 1984

[SchMet72] D.R. Schertz, G Metze, "A new Representation for Faults in Combinational Digital Circuits", *IEEE Trans on Comp Vol C-21, Number 8*, pp 858–866, Aug 72

[SetAgr84] S.C Seth, V.D. Agrawal, "Characterizing the LSI Yield Equation from Wafer Test Data" *IEEE Transactions on CAD, Vol CAD-3* pp 123–126, April 1984

[ShMaFe85] J P Shen, W. Maly, F.J. Ferguson, "Inductive Fault Analysis of MOS Integrated Circuits" *Design and Test of Computers* pp 13-26, Dec 1985

[Smith85] G.L. Smith, "Model for Delay Faults Based Upon Paths", *Proc International Test Conference*, pp. 342–349, Nov 1985

[Stappe75] C.H. Stapper, "On a Composite Model to the IC Yield Problem", *IEEE Journal of Solid-State Circuits, Vol DC-10*, pp. 537–539, Dec 1975

[StoBar77] T.M. Storey, J.W Barry, "Delay Test Simulation", *Proc 14th Design Automation Conference*, pp. 492–494, June 1977

[TsiUlr62] S.H Tsiang, W Ulrich, "Automatic Trouble Diagnosis of Complex Logic Circuits," *Bell Sys. Tech Jour*, Vol 41, p 1177, July 1962

[TuLePM85] M E Turner, D G Leet, R J Prilik, D.J McLean, "Testing CMOS VLSI Tools, Concepts, and Experimental Results", *Proc International Test Conference*, pp 322–328, Sep. 1985

[WaLiRI87] J. A. Waicukauski, E Lindbloom, B.K. Rosen, V S Iyengar, "Transition Fault Simulation" *IEEE Design and Test of Computers*, Vol 4, pp 32-38, April 1987

[Wadsac78] R. L. Wadsack, "Fault Modelling and Logic Simulation of CMOS and MOS Integrated Circuits" *Bell Systems Technical Journal* pp 1449-1474, May-June 1978

[WaiLin89] J.A. Waicukauski, E Lindbloom, "Failure Diagnosis of Structured VLSI," *IEEE Design and Test*, Vol 6, No 4, pp 49-60, Aug 1989

[Wang88] F. Wang, "BIST Using Pseudo-Random Test Vectors and Signature Analysis", *Proc IEEE Custom Integrated Circuits Conference*, pp 16 1 1–16 1 8, May 1988

[WilBro81] T W Williams, N C Brown, "Defect Level as a Function of Fault Coverage," *IEEE Trans. Comp.*, Vol C-30, pp 987–988, Dec 1981

[WilPar83] T.W. Williams, K.P. Parker, "Design for Testability - A Survey," *Proc. IEEE Transactions on Computers*, Vol. C-31, pp. 2–14, Jan. 1983.

[WoNeSa87] B. W. Woodhall, B.D. Newman, A.G. Sammuli, "Empirical Results on CMOS Stuck-Open Failures" *Proc. International Test Conference* pp 166–170, Sep. 1987.

[YanOka87] T. Yano, H. Okamoto, "Fast Fault Diagnostic Method Using Fault Dictionary for Electron Beam Tester", *Proc. International Test Conference*, pp. 561–565, Sep 1987.