# Enhancing Transition Fault Coverage in Built-In Self-Test

by

Fayaz M. Kadri

B.Sc., Middle East Technical University, 1991

ACCEPTED
FACULTY OF GRADUATE STUDIES
DEAN
DATE May 3/93

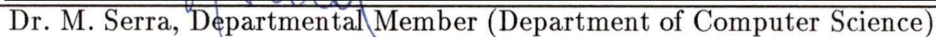
A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of

MASTER OF SCIENCE

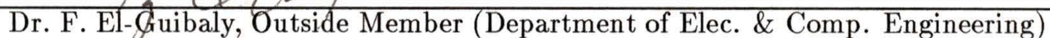
in the Department of Computer Science

We accept this thesis as conforming to the required standard

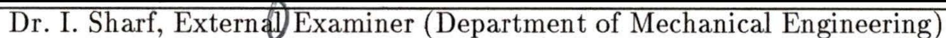
Dr. D. M. Miller, Supervisor (Department of Computer Science)



Dr. M. Serra, Departmental Member (Department of Computer Science)



Dr. F. El-Guibaly, Outside Member (Department of Elec. & Comp. Engineering)



Dr. I. Sharf, External Examiner (Department of Mechanical Engineering)

©FAYAZ M. KADRI, 1993

University of Victoria

All rights reserved. Thesis may not be reproduced in whole or in part, by photocopy or other means, without the permission of the author.

QA 76.9
F38K2

Supervisor: Dr. D. M. Miller

## Abstract

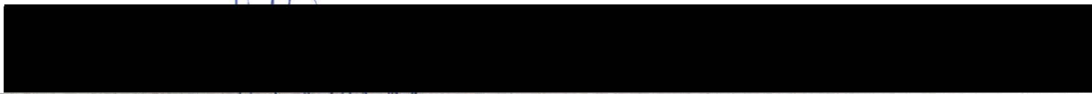
A pair of test vectors is required to detect a *transition fault*, therefore, a high fault coverage for a circuit under test (CUT) depends on the sequence of test vectors (patterns) applied during a test. *Linear feedback shift registers* (LFSRs) are commonly used as test pattern generators in built-in self-test (BIST) because of their low implementation cost.

The shift dependency problem inherent in an LFSR is a hindrance to achieving a high transition fault coverage. In this thesis, we show that it is possible to achieve a high *transition fault* coverage by randomly permuting the connections between the outputs of the test pattern generator (LFSR) and the inputs of the CUT. We then present an algorithm that permutes these connections based on the structure of the CUT.

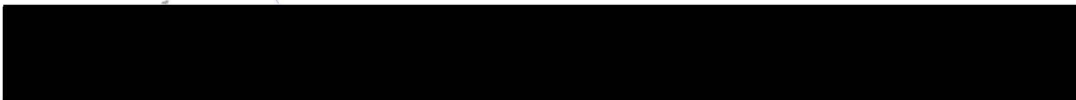
Results are presented to show that the algorithm compares favourably with the published methods. Some preliminary results for the effect of an initializing vector (seed value) for the LFSR are also presented. The thesis concludes with a number of further research direction arising from this work.

Examiners:

Dr. D. M. Miller, Supervisor (Department of Computer Science)



Dr. M. Serra, Departmental Member (Department of Computer Science)



Dr. F. El-Guibaly, Outside Member (Department of Elec. & Comp. Engineering)

Dr. I. Sharf, External Examiner (Department of Mechanical Engineering)

# Acknowledgements

I would sincerely like to thank my supervisor, Dr. D. M. Miller, for his excellent guidance throughout my M.Sc. program. I would like to thank Mr. Rodrigue Byrne for being a constant source of help and a special thanks to Mr. Shujian Zhang for allowing me to have access to his fault simulator that enabled me to generate important results in this thesis. I also like to thank Dr. M. Serra and Dr. J. Muzio for allowing me to audit their VLSI courses.

Furthermore, I would like to thank my parents and family members for their constant encouragement and efforts.

# Contents

**Abstract** ii

**Acknowledgements** iv

**Contents** v

**List of Tables** ix

**List of Figures** xii

**1 Introduction** 1

**2 Fault Models** 4

2.1 Single Stuck-at-Fault Model . . . . . . . . . . . . . . . . . . . . . . . . 5

2.2 Bridging Fault Model . . . . . . . . . . . . . . . . . . . . . . . . 8

2.3 Stuck-Open Fault Model . . . . . . . . . . . . . . . . . . . . . . . . 10

2.4 Delay Fault Models . . . . . . . . . . . . . . . . . . . . . . . . 14

2.4.1 Types of Delay Fault Model . . . . . . . . . . . . . . . . . . . . 15

2.5 Summary . . . . . . . . . . . . . . . . . . . . . . . . 18

**3 Testing and Test Pattern Generators 19**

3.1 Test Pattern Generation . . . . . 19

3.1.1 Fault Coverage . . . . . 20

3.2 Data Compression . . . . . 21

3.3 Pseudorandom Pattern Generators . . . . . 22

3.3.1 Linear Feedback Shift Register (LFSR) . . . . . 22

3.3.2 Linear Cellular Automata Register (LCAR) . . . . . 26

3.4 Summary . . . . . 28

**4 Delay Fault Testing 29**

4.1 Requirements of Delay Fault Testing . . . . . 29

4.2 Concerns in Delay Fault Testing . . . . . 30

4.2.1 Hazards . . . . . 30

4.2.2 Fault Masking and Robust Test . . . . . 33

4.3 Transition Fault Testing . . . . . 35

4.4 Transition Fault Testing with a LFSR . . . . . 37

4.4.1 Shift Dependency Problem with a LFSR . . . . . 38

4.5 Overcoming the Shift Dependency . . . . . 39

4.5.1 Cross Over Permutation . . . . . 39

4.5.2 Random Connections . . . . . 40

4.5.3 Input Separation using Dummies (ISUD) . . . . . 44

4.6 Summary . . . . . 49

**5 Algorithm To Identify Connections 50**

5.1 Requirements and Characteristics of the Algorithm . . . . . . . . . . . . . . 51

5.1.1 Information Required . . . . . . . . . . . . . . . . . . . . . . . . 52

5.1.2 Characteristics . . . . . . . . . . . . . . . . . . . . . . . . . . . . 54

5.1.3 Phases of the Algorithm . . . . . . . . . . . . . . . . . . . . . . . 56

5.1.4 Summary of Requirements and Characteristics . . . . . . . . . . . . 60

5.2 The Algorithm . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 61

5.2.1 Steps for executing the Algorithm . . . . . . . . . . . . . . . . . . 62

5.3 Examples . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 66

5.3.1 Using Phase Two only (The *ISSLA* method) . . . . . . . . . . . . . 67

5.3.2 Comparing Single Phase (*ISSLA*) with both Phases (*ISASLA*) . . . 72

5.3.3 Logic with Single Output . . . . . . . . . . . . . . . . . . . . . . . 78

5.4 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 85

**6 Simulation Results and Observations 86**

6.1 Options of the Algorithm . . . . . . . . . . . . . . . . . . . . . . . . . 88

6.1.1 Options of Interest . . . . . . . . . . . . . . . . . . . . . . . . . . 89

6.1.2 Results . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 91

6.2 Comparing Results . . . . . . . . . . . . . . . . . . . . . . . . . . . . 95

6.2.1 Comparing Results for the Algorithm with Random Connections . . 96

6.2.2 Comparing Results for the Algorithm with Other Methods . . . . . 102

6.3 Effect of Seeds . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 107

6.3.1 Experiments . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 107

6.4 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 110

**7 Conclusions And Future Directions** **111**

**Bibliography** **114**

# List of Tables

4.1 Transition fault coverage for the ISCAS '85 circuits using NP-CXN and XLFSR. . . . . 41

4.2 Best observed transitions fault coverage for the ISCAS '85 circuits using random connections. . . . . 44

4.3 Transition fault simulation results for the ISCAS '85 circuits using ISUD method. . . . . 48

5.1 Transition fault coverage for randomly generated connections (R-CXN) and the input separation method using dummies (ISUD). . . . . 51

5.2 Secondary logic levels at which two inputs meet. . . . . 54

5.3 Effect of temporarily assigning input 5 to each of the unassigned latches. . . 59

5.4 Effect of temporarily assigning input 6 to each of the unassigned latches. . . 60

5.5 Logic levels at which two inputs "logically" meet in binary to excess 3 code circuit. . . . . 67

5.6 Effect of temporarily assigning $PI_2$ to each of the unassigned latches. . . . . 70

5.7 Effect of temporarily assigning $PI_3$ to each of the unassigned latches. . . . . 70

5.8 Resulting effect of assigning circuit inputs to SRLs using the second phase. 71

5.9 Resulting effect of assigning circuit inputs to SRLs using null permutation connections. . . . . 72

5.10 Logic levels at which two inputs meet in the example circuit. . . . . . . . . 74

5.11 Effect of temporarily assigning $PI_4$ to each of the unassigned latches. . . . . 75

5.12 Resulting effect of assigning circuit inputs to SRLs using the second phase. 75

5.13 Effect of temporarily assigning $PI_5$ to each of the unassigned latches. . . . . 78

5.14 Resulting effect of assigning circuit inputs to SRLs using both phases of the algorithm. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 78

5.15 Primary logic levels at which two inputs meet. . . . . . . . . . . . . . . . . 81

5.16 Effect of temporarily assigning $PI_4$ to each of the unassigned latches. . . . . 82

5.17 Resulting effect of assigning circuit inputs to SRLs using the second phase. 82

5.18 Resulting effect of assigning circuit inputs to SRLs using the ISSIA method. 84

6.1 ISCAS '85 combinational benchmark circuits. . . . . . . . . . . . . . . . . . 87

6.2 Fault coverage for ISCAS'85 circuits using both orientations of LFSR and connections for the first 5 options of the algorithm. . . . . . . . . . . . . . . 93

6.3 Fault coverage for ISCAS'85 circuits using both orientations of LFSR and connections for the last 5 options of the algorithm. . . . . . . . . . . . . . . 94

6.4 Summary for best observed fault coverage among various options of the algorithm. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 95

6.5 Statistical results for ISCAS'85 circuits using a 100 random connections and FBf orientation of LFSR. . . . . . . . . . . . . . . . . . . . . . . . 97

6.6 Statistical results for ISCAS'85 circuits using a 100 random connections and FBl orientation of LFSR. . . . . . . . . . . . . . . . . . . . . . . . 97

6.7 Statistical results for ISCAS'85 circuits using a 100 random connections and both orientations (FBf and FBl) of the LFSR. . . . . . . . . . . . . . 98

6.8 Statistical results for ISCAS'85 circuits for results obtained using the first 5 options of the algorithm.. . . . . . . . . . . . . . . . . . . . . . . . . . 98

6.9 Comparing fault coverage of various methods. . . . . . . . . . . . . . . . . . . . 103

6.10 Summary for the best observed fault coverage for ISCAS'85 circuits using various methods. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 104

6.11 Overhead and Coverage for Delay Fault Simulation of the ISCAS '85 Circuits as given in the [SR92]. . . . . . . . . . . . . . . . . . . . . . . . . . . . 105

6.12 Statistical results for ISCAS'85 circuits using 30 random seeds. . . . . . . . 108

# List of Figures

2.1 An example of a stuck-at fault. . . . . . . . . . . . . . . . . . . . . . . . . . 6

2.2 Test for a s-a-0 fault on line $l_1$. . . . . . . . . . . . . . . . . . . . . . . . . . . 8

2.3 Bridging fault model. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9

2.4 Feedback bridging fault model. . . . . . . . . . . . . . . . . . . . . . . . . . 9

2.5 Block diagram of a CMOS cell and enhancement mode transistor symbols. . 11

2.6 CMOS inverter cell. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12

2.7 CMOS NOR cell. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12

2.8 Hardware model for delay test of a combinational network. . . . . . . . . . 15

2.9 Deficiency of gate delay fault model. . . . . . . . . . . . . . . . . . . . . . . 17

3.1 Linear feedback shift registers. . . . . . . . . . . . . . . . . . . . . . . . . . 22

3.2 Autonomous linear cellular automata register. . . . . . . . . . . . . . . . . . 26

4.1 Delay fault testing. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 30

4.2 Hazards in delay fault testing. . . . . . . . . . . . . . . . . . . . . . . . . . 31

4.3 Unintended spike (hazard) during delay fault testing. . . . . . . . . . . . . . 32

4.4 Masking during delay fault testing. . . . . . . . . . . . . . . . . . . . . . . . 33

4.5 Timing diagram for the test in Figure 4.4. . . . . . . . . . . . . . . . . . . . 34

4.6 Transition fault testing. . . . . . . . . . . . . . . . . . . . . . . . . . . . 36

4.7 Shift dependency problem with LFSR. . . . . . . . . . . . . . . . . . . . . . . 38

4.8 Cross over permutation. . . . . . . . . . . . . . . . . . . . . . . . . . . . 40

4.9 Shift dependency problems within logic cones. . . . . . . . . . . . . . . . . . 45

4.10 Input separation for logic cones. . . . . . . . . . . . . . . . . . . . . . . . 47

5.1 Levels of a circuit. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 52

5.2 Logic cones of the sample circuit. . . . . . . . . . . . . . . . . . . . . . . 55

5.3 Assigning inputs to latches during the second phase. . . . . . . . . . . . . . 58

5.4 Binary (0-9) to excess 3 code circuit. . . . . . . . . . . . . . . . . . . . . . 68

5.5 Logic cones for binary (0-9) to excess 3 code circuit. . . . . . . . . . . . . . 69

5.6 Connections between the SRLs and the circuit inputs. . . . . . . . . . . . . 71

5.7 Example circuit. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 73

5.8 Logic cones for the example circuit. . . . . . . . . . . . . . . . . . . . . . . 73

5.9 Connections between the SRLs and the circuit inputs obtained by using the second phase. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 76

5.10 SRLs to circuit inputs connections obtained using both phases. . . . . . . . 79

5.11 Two of five checker circuit. . . . . . . . . . . . . . . . . . . . . . . . . . . 80

5.12 SRLs to circuit inputs connections obtained using the second phase. . . . . 83

5.13 SRLs to circuit inputs connections obtained using the ISSIA method. . . . . 85

6.1 Comparing the results for R-CXN with the best observed results for H-CXN.101

# Chapter 1

# Introduction

Digital circuits are widely used in systems that play an important role in our daily lives. *Physical failure* of digital circuits due to environmental factors and wear out through usage is inevitable. Therefore, testing of these components in a digital system is considered to be essential.

In general, physical failures, also called *physical faults*, do not allow a mathematical treatment of testing and diagnosis. It is much easier to detect the logical effect of a physical fault rather than to discover the exact physical attributes. The abstraction which maps a large number of physical faults into a smaller number of logical faults is called a *fault model.*

Several fault models have been developed and studied. Some of these fault models are *stuck-at, bridging, stuck-open, delay* and *transition* fault models. Considerable research work has been done on the *stuck-at* fault model and it is widely used in the industry.

For the *stuck-at* fault model, the *static tests*, which are single pattern tests, are useful in detecting certain physical faults by verifying the input-output behaviour of the digital circuits. But, developments in the VLSI technology and the tremendous increase in the speed at which these digital circuits operate, raises the concern of testing the circuits to verify their correct operation at *system clock speed.*

Physical defects that lead to incorrect operation of digital circuits at system clock speed are represented by the *transition fault model* as a *slow-to-rise* or *slow-to-fall* transitions

on the lines. A transition fault requires a pair of test vectors in order to be detected. Therefore, when a circuit is tested for transition faults using *pseudorandom* vectors, the sequence of test vectors applied is important for a high transition fault coverage to be achieved.

*Built-In Self-Test* (BIST) refers to the ability of the hardware to test itself. *Linear feedback shift registers* (LFSRs) are commonly used as the source of test stimuli during *pseudorandom* testing in BIST. But, the shift dependency problem of the test vectors from a LFSR is a hindrance to achieving a high fault coverage during transition fault testing.

Two methods, XLFSR (Cross Over permutation) and ISUD (Input Separation Using Dummies), have appeared in the literature. These methods try to overcome the shift dependency problem by rearranging the connections between the outputs of a test pattern generator and the inputs to the circuit under test (CUT). Although these methods improve the *transition fault* coverage, each method has its own disadvantage.

In this thesis, we show results which indicate that a high transition fault coverage can be achieved by randomly permuting the connections between the generator outputs and the circuit inputs. Since, the probability of random connections that can give a high transition fault coverage is low, we present an algorithm that permutes the connections based on the structure of the circuit. We then present results to show that the algorithm compares quite favourably with the published methods.

We also present some preliminary results for the effect of initializing vectors (seed values) of LFSR on the test length during pseudorandom testing of transition faults.

In chapter 2, some of the fault models that have been developed to represent physical faults in a circuit are reviewed.

Chapter 3 briefly describes the basic testing concepts. This chapter also describes the linear feedback shift register (LFSR) and some of its properties. LFSRs are a low cost test pattern generator used for generating test stimuli during a test. Recently an alternative to LFSR was proposed and has been widely studied. This generator called the linear cellular automata register (LCAR) is briefly described in section 3.3.2.

*Delay fault* testing, which is of interest in this thesis, is described in chapter 4. The *gate*

*delay fault* is implicit in the transition fault model, therefore, the requirements and testing protocols for *transition faults* are given in section 4.3. The shift dependency problem that hinders the achievement of a high *transition fault* coverage is described in section 4.4.1. Some of the methods (XLFSR and ISUD) that have appeared in the literature and which try to overcome this problem are described in section 4.5.

In chapter 5, we introduce our algorithm which was developed not only to enhance transition fault coverage, but also to overcome the disadvantages of the published methods. The algorithm consists of *two phases.* Either a combination of both phases (the *ISASLA* method) or only the second phase (the *ISSLA* method) can be used. Some examples are given in section 5.3 that describe how the *ISASLA* and *ISSLA* methods work. Another method for input separation is suggested in section 5.3.3.2. This method (the *ISSIA* method) is similar to the ISUD method.

In chapter 6, the *transition fault* simulation results for connections obtained using the new algorithm are presented. These results are compared with the results obtained for our implementation of the XLFSR and the ISUD methods in section 6.2. The simulation results for the effect of random seed values on *transition fault* coverage and the *test length* are presented in section 6.3.

Chapter 7 concludes the thesis by summarizing the contributions made and listing some interesting problems that require further investigation.

# Chapter 2

# Fault Models

The function of a digital circuit can be described by the relationship between the set of primary inputs and the set of primary outputs. When the circuit is stimulated by a certain combination of input values, a certain output response is expected. But, when the output response is different from the expected ones, the digital system is said to be in *error.*

The causes of observed errors may be *design errors, fabrication errors, fabrication defects* and *physical failures.* Unlike the *design and fabrication errors,* the *fabrication defects* are not directly attributable to a human error; rather they result from an imperfect manufacturing process.

*Physical failures* occur during the lifetime of a system due to component wear-out and/or environmental factors. For example [ABF90], aluminium connectors inside an IC package thin out with time and may break because of electron migration or corrosion. Examples of environmental factors that accelerate the aging of components are temperature, humidity and vibrations.

The identification of *design errors* falls within the more precise category of *verification* of circuits. *Fabrication errors, fabrication defects* and *physical failures,* are collectively called *physical faults* and their detection is done by testing of circuits. According to their stability in time, physical faults can be classified as:

- permanent – always being present after occurrence;

- intermittent – those which appear, disappear and reappear repeatedly; and
- transient – appearing for a short period of time caused by a temporary change in some environmental factor.

The interest is particularly in the *detection* of *physical faults* rather than their identification. In general, physical faults do not allow a direct mathematical treatment of testing. The solution is to deal with *logical faults*, which are a convenient representation of the *effect* of the *physical faults* on the operation of the system. Thus, a *physical fault* can be *detected* by observing an error caused by it.

The abstraction which maps a large number of *physical faults* into a smaller number of *logical faults* is called a *fault model*. The degree to which a *fault model* truly does represent *physical faults* in a particular technology is central to the success of any test method based on that model. *Fault models* have been developed for both, *gate level* and the *transistor level* description of circuits.

The *single stuck-at fault* (SSF) model, which is a gate level fault model, is by far the most prevalent, both in the literature and in the testing industry. Although the SSF model is widely used, its validity is not universal. Therefore, other fault models have been developed which try to account for the *physical faults* not adequately covered by the SSF model. Examples of other gate level fault models are the *bridging-fault* (BF) model and the *delay fault* model. An example of a transistor level fault model is the *stuck-open* transistor in CMOS (complimentary metal oxide semiconductor) circuits [Mil87].

The following sections briefly describe some of the fault models and how the modeled faults are detected. The details of delay fault testing, which is of interest in this thesis, will be given in chapter 4.

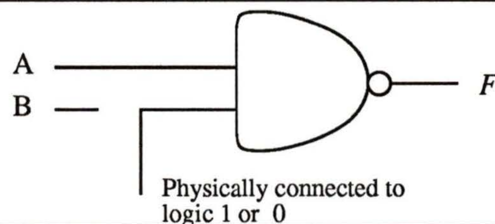
## 2.1 Single Stuck-at-Fault Model

The *single stuck-at fault* (SSF) model is a *classical* or *standard* fault model because it has been the first and the most widely studied and used. This model has gained its popularity because of its effectiveness and simplicity. The basic assumptions of the *stuck-at* fault

model are [Joh89, page 32]:

- a fault results in a module[1] responding as if one of its inputs or outputs is physically stuck-at a logic 1 or 0.
- the basic functionality of the module is not altered by the fault.
- the fault is permanent.

**Figure 2.1** An example of a stuck-at fault.




To illustrate these assumptions, consider the NAND gate in Figure 2.1, with two inputs $A$ and $B$, and output $F$. The input lines $A$ and $B$ are free to take on either value (0 or 1), but the gate responds as if line $B$ is physically stuck at either logic 1 or logic 0. For example, a stuck-at-1 fault on line $B$ results in the output of the NAND gate, being 0 whenever input $A$ is 1, regardless of the actual value on line $B$. Likewise, a stuck-at-0 fault on line $B$ causes the NAND gate to always have an output of 1, regardless of the actual value on line $B$.

The *second assumption* does not mean that the circuit continues to produce the correct results. It simply implies that, given the existence of the fault, the circuit produces the results expected of it. For the given example in Figure 2.1, if the line $B$ is stuck-at-0, the output will always be at logic 1 if the gate continues to behave as a NAND gate. If, however, a failure transforms the NAND gate into an AND gate, the assumptions of the stuck-at fault model are violated. Similarly, if a failure transforms a combinational circuit into a sequential circuit, the assumptions of the stuck-at-fault are violated. The later example, as will be seen in section 2.3, is a significant problem in some circuit technologies, like CMOS.

[1]The logic module can be a single gate or a collection of gates that implements some logic function.

In the *third assumption*, the faulty module *always* performs as if the line is stuck-at a specific logic value. This assumption simplifies the fault model by avoiding the difficulty of modeling *intermittent* faults. For example [Joh89, page 31], a fault resulting from a weak solder joint in a circuit. The solder joint provides proper contact at certain times and improper contact at others.

The usefulness of SSF results from the following attributes[ABF90]:

- It represents many different physical faults.
- It is independent of technology, as the concept of a signal line being stuck at a logic value can be applied to any structural model[2].
- Tests that detect SSFs detect many non-classical faults as well.
- Compared to other fault models, the number of SSFs in a circuit is small. Moreover, the number of faults to be explicitly analyzed can be reduced by *fault-collapsing techniques*[3].

**Test for a SSF**

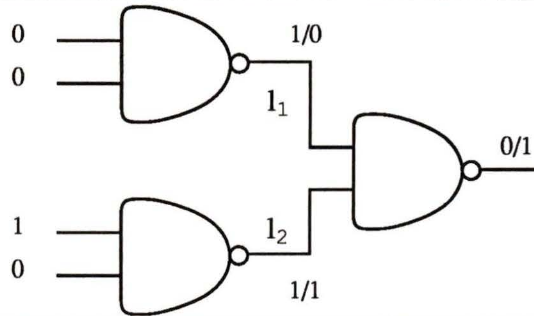
As mentioned earlier, a physical fault is detected by observing errors caused by it. In the case of a SSF model, the stuck-at-0 (stuck-at-1) fault on a line, $l_1$, in a circuit is detected by applying signals at the inputs of the circuit which set $l_1$ to 1(0). In the presence of a stuck-at-0 (stuck-at-1) fault, an inverted value is propagated to one of the outputs.

Consider the circuit in Figure 2.2. In order to detect a stuck-at-0 (s-a-0) fault on line $l_1$, the test vector (test pattern) (0,0,1,0) is selected such that $l_1$ can be set to the inverted logic value, 1. The expected value on $l_1$ is 1 but due to the presence of a s-a-0 fault on $l_1$, the logic value remains at 0. This is represented as 1/0 on the line in Figure 2.2. In order to propagate the inverted value on $l_1$, $l_2$ must be at logic 1. The output response is therefore 0/1 (*i.e.* an inverted value appears at the output and thus the s-a-0 fault on $l_1$ is detected).

[2] A *structural model* describes a circuit as a collection of interconnected smaller circuits.

[3] *Fault-collapsing techniques* group modeled faults, that require the same set of tests in order to be detected, into equivalent classes.

**Figure 2.2** Test for a s-a-0 fault on line $l_1$.




The test for SSFs are called *static* tests, since only a single test pattern is required to detect a fault. In some fault models, for example the *delay fault* model, a pair of test patterns is required to detect a modeled fault. Such a test is called an *AC test.*

The other distinction between a *static test* and an *AC test* is that the test patterns in *static tests* are applied at a much slower speed than the normal operation speed [ABF90]. Whereas, in *AC tests*, the test patterns are applied at normal operation speed.

## 2.2 Bridging Fault Model

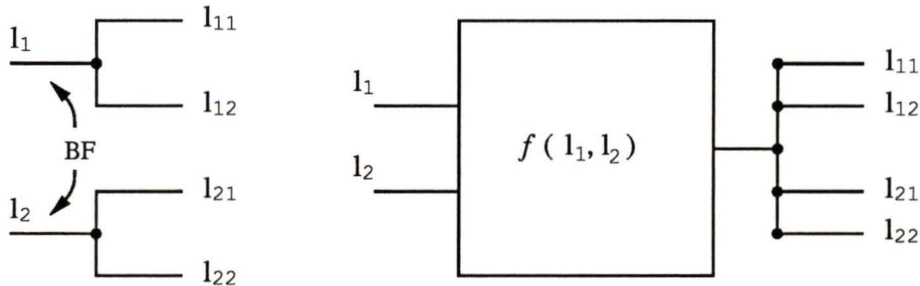
Shorts between normally unconnected signals are manufacturing defects that often occur in digital circuits. The fault model which models such defects is the *bridging-fault* (BF) model.

When the lines in a circuit are shorted, they become equipotential and therefore carry the same logic value. Therefore, a BF can:

- replace the shorted lines by a logical combination (AND or OR) of those lines.
- change a combinational circuit into a sequential one.

Consider the lines $l_1$ and $l_2$ in Figure 2.3. The BF has no effect on the values carried by the lines *if* they carry the same logic values. But, if the lines are driven to *opposite values*, then the effect of a BF on the resulting value on both lines depends on the technology [ABF90]. For example, in MOS, the resulting value, in general, is indeterminate (*i.e* the resulting value does not correspond to any logic value).

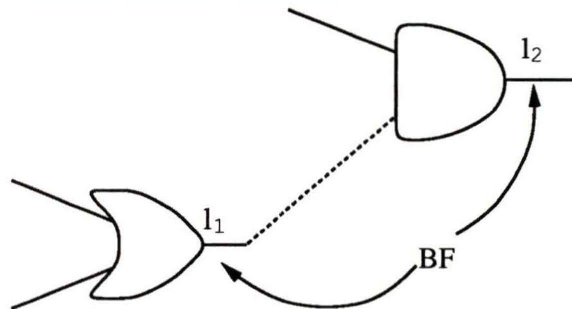
**Figure 2.3** Bridging fault model.




In many technologies (such as TTL[4] or ECL[5]), when two lines carrying opposite values are shorted, one value overrides the other. The overriding value is called the *strong* value. If $s$ ($s \in \{0, 1\}$) is the *strong value*, then the resulting function $f(l_1, l_2) = c$ in Figure 2.3, introduced by the BF is AND if c = 0 and OR if c = 1.

As a result of the BF in Figure 2.3 the fanout branches ($l_{11}, l_{12}, l_{21}$ and $l_{22}$) always have the same logic value. Therefore the BF can be considered only between gate outputs and/or primary inputs.

**Figure 2.4** Feedback bridging fault model.




Depending on the lines that are shorted, the bridging faults can be referred to as [ABF90]:

1. a *feedback bridging-fault* (FBF) - A BF that creates one or more feedback loops (Figure 2.4).

[4]Transistor/Transistor Logic

[5]Emitter Coupled Logic

2. a *non-feedback bridging-fault* (NFBF) - A BF that does not create feedback (Figure 2.3).

The multiple bridging fault (MBF) model represents shorts involving more than two lines. More details on MBF can be found in [ABF90].

The BF model is often used *in addition* to the SSF model. There exists some relations between the detection of SSFs and the detection of BFs. These relations can be exploited in fault simulation and test generation methods for BFs [ABF90], so that the processing of the two models can be combined.

A FBF can cause a *combinational* circuit to become a *sequential* one which, in general, require a test sequence in order to be detected. However, in many cases, a FBF can be detected by a single test [ABF90]. The details on the detection of both FBF and NFBF can be found in [ABF90].

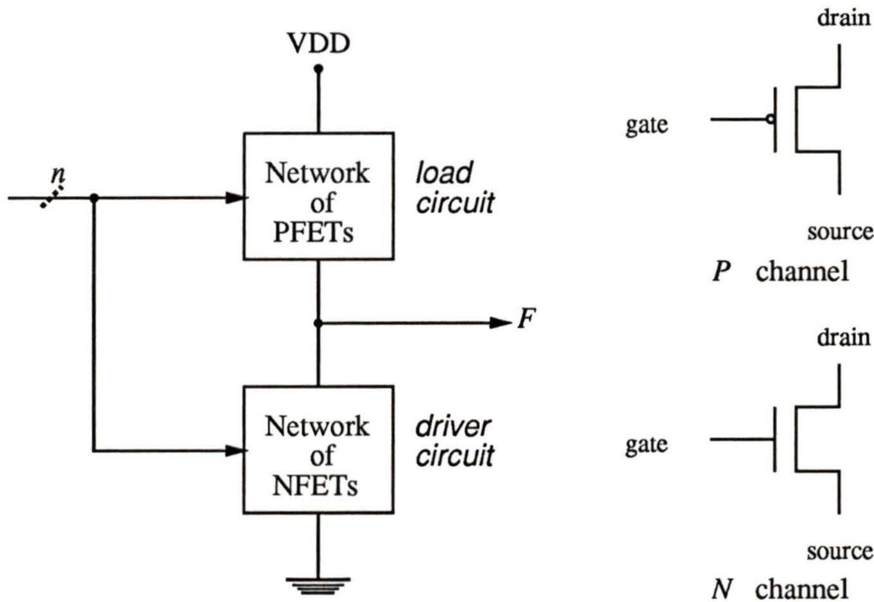
## 2.3 Stuck-Open Fault Model

A variety of fault models have been developed to account for the physical failures not adequately covered by the SSF model. CMOS (complementary metal oxide semiconductor) has become a major VLSI technology primarily due to its low static power dissipation and the reasonable circuit density that can be achieved [Mil87].

CMOS circuits are composed of $P$ channel and $N$ channel enhancement mode field effect transistors (FETs). A simple block diagram of a CMOS cell implementing a function $F$ is given in Figure 2.5 (a).

The load circuit is a network of $P$ channel transistors ($p$-transistors), and, the driver circuit is a network of $N$ channel transistors ($n$-transistors). Every input to the circuit is connected to the *gate* of the transistors in both, the *load circuit* and the *driver circuit.* The transistors can be viewed as simple switches. The symbols for both kind of transistors are given in Figure 2.5 (b).

**Figure 2.5** Block diagram of a CMOS cell and enhancement mode transistor symbols.



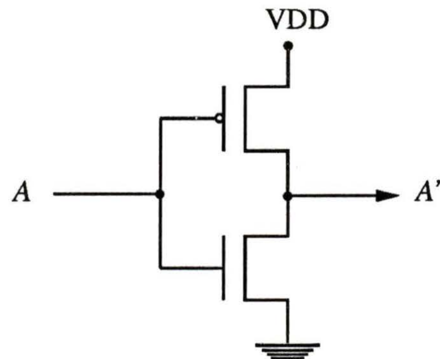

(a) Block diagram of a CMOS cell

(b) Symbols for CMOS enhancement mode transistors

Depending on the value at the gate of a transistor, a conducting path is created between the source and the drain. For a *p*-transistor, a logic 0 (GND) at the *gate* creates a conducting path between the drain and source. For an *n*-transistor, a conducting path is created when the *gate* is at logic 1 (VDD). When the *inverted* values are applied at the *gates*, the transistors do not conduct.

To understand the operation of a CMOS cell, consider the CMOS inverter cell in Figure 2.6. The load circuit consists of a single *P* channel transistor, and the driver circuit consists of a single *N* channel transistor. When the input is at 0, the *P* channel transistor is conducting and therefore the output (A') goes to 1. When the input is at 1, a path is created from the output to the ground (GND) through the conducting *N* channel transistor, thus the output goes to 0 (GND).

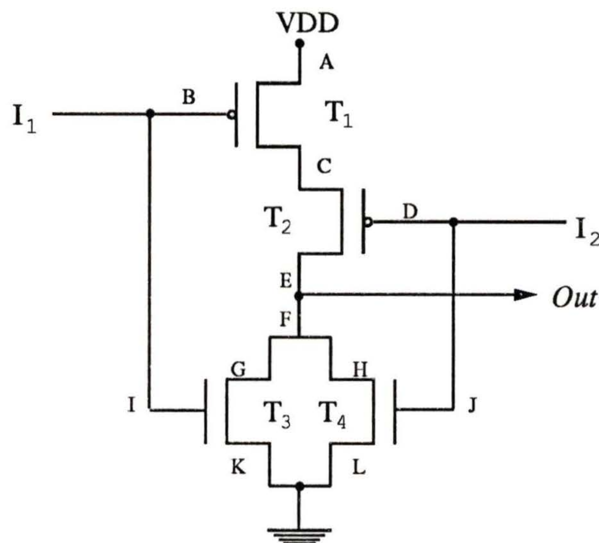
**Figure 2.6** CMOS inverter cell.




A CMOS circuit is formed by superimposing several layers of conducting material (metal and diffusion), insulating material and transistor forming material. The physical failures in a CMOS circuit which lead to permanent functional faults are broken conductors, faulty transistors and shorted conductors.

A *stuck-open* fault represents any failure in a CMOS cell which results in one or more transistors that cannot conduct. This can be the result of a broken conductor or can arise from a failure in a transistor such as a faulty channel or gate shorted to a substrate [Mil87].

**Figure 2.7** CMOS NOR cell.



Consider the CMOS NOR cell in Figure 2.7 with two inputs $I_1$ and $I_2$, and output *Out*. Suppose conductor $C$ is broken. In this case, the load circuit cannot conduct and the output cannot be pulled up to 1. If both inputs are 0, neither the driver circuit nor, the load circuit is conducting, and the cell output is floating. The load capacitance[6] will hold the previous value for some time which is a function of the amount of capacitance and the leakage currents in the circuit. The circuit failure has thus introduced memory into a combinational circuit which violates the assumptions of the SSF fault model (section 2.1).

**Test for a Stuck-Open Fault**

The *stuck-open* faults cannot be detected by *static* tests. For example, consider the transistor $T_1$ in Figure 2.7 to be *stuck-open.* In order to distinguish between a faulty and a fault free CMOS NOR cell, an input vector is required that completes a path from VDD to the output, *Out.* The input vector $v = (I_1, I_2) = (0,0)$ completes this path. In the fault-free CMOS NOR cell, the output for $v = (0,0)$ is 1, but since $T_1$ is *stuck-open,* the output is connected neither to VDD nor to GND. Thus, there is no input vector which drives the output, to one logic value in the absence of the fault, and drives the output to another logic value in the presence of the fault. Thus, a *static test* does not detect a *stuck-open* fault.

The *stuck-open* faults can be detected by a pair of test vectors $(v_1, v_2)$. The test for a *stuck-open* $p$-transistor ($n$-transistor), consists of a test vector pair $(v_1, v_2)$ such that:

1. $v_1$ completes a path from the output to GND (VDD);
2. $v_2$ completes a path from the output to VDD (GND) through the faulty $p$-transistor ($n$-transistor).

The capacitance associated with the output node (*Out*) can temporarily store some charge. Therefore, when $v_1$ is applied, the charge on the capacitor is such that the output is at logic 0 (logic 1). When $v_2$ is applied, the charge on the capacitor is such that the output is *inverted* in the absence of a faulty transistor. If the output remains unchanged for $v_2$, then a *stuck-open* transistor fault is detected.

---

[6]The output load capacitance is a result of the capacitance of the transistor gates or chip pins to which the output of the cell is connected.

Two important assumptions for the *stuck-open* transistor fault test are:

1. only a single *stuck-open* transistor fault exists; and

2. $v_2$ is such that the fault is not masked by a parallel path.

For example, the pair of test-vectors $(v_1, v_2) = (I_1I_2, I_1I_2) = (00, 01)$ can detect the stuck-open $T_4$ transistor as follows:

1. $v_1 = (00)$, completes a path form the output to VDD; therefore the output goes to 1.

2. $v_2 = (01)$ sets $I_2 = 1$ to exercise the faulty transistor and sets $I_1 = 0$ to avoid masking the fault by parallel path (through $T_3$).

In the presence of $T_4$ *stuck-open* fault, the path from the output to GND cannot be completed via $T_4$. At the same time, the path from the output to VDD is not complete, since $I_2 = 1$ and therefore $T_2$ is not conducting. But, the capacitance at the output retains the logic 1 value set by $v_1$ and hence the fault is detected.

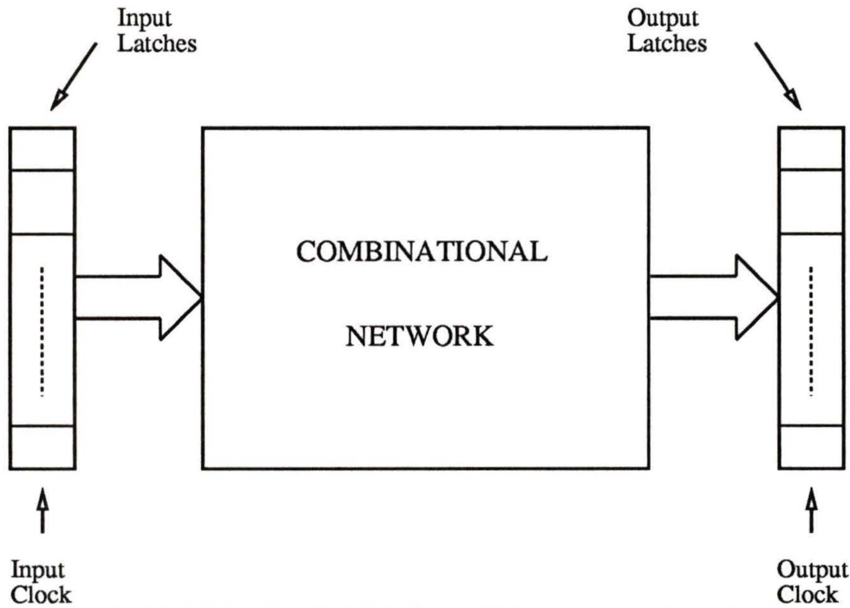
Therefore, a pair of test vectors (*i.e.* an AC test) is required to detect a stuck-open transistor fault.

## 2.4 Delay Fault Models

One of the fault models that is used to represent the effect of a physical fault in a VLSI circuit is the *delay fault* model. For the SSF fault model, the *static tests* verify the *functional behaviour* of a given logic circuit under the assumption that the signals are allowed sufficient time to propagate from the inputs of the circuit to its outputs. In the *delay fault* model, it is assumed that some of the signals in the logic may not propagate in time when operating at the system clock speed.

Consider the hardware model for a delay test of a combinational network in Figure 2.8 [Smi85]. The model consists of input latches that feed the combinational logic at the arrival of the input clock. The logic signals are expected to propagate to the outputs of the combinational logic in time $\delta$t, when, the output clock arrives to latch these signals in the output latches.

**Figure 2.8** Hardware model for delay test of a combinational network.




The interval, $\delta t$, is the time allowed for the propagation delays through the logic gates in the circuit. The logic gates have a distribution of delay values. The clocking can therefore be designed in one of two ways:

1. to meet the worst case criterion.
2. to meet the *statistical* worst case criterion.

The delay values for the gates may exceed their specifications due to *process variations, stray capacitances* and/or *physical defects* such as open metal lines [PS92]. Therefore, the actual delay for the logic values to propagate to the outputs may be greater than $\delta t$ which may lead to incorrect logic values being latched at the outputs. These failures causing logic circuits to malfunction at the desired clock rate are called *delay faults* or *AC faults* [BR83].

### 2.4.1 Types of Delay Fault Model

Two types of *delay fault* models have been mentioned in the literature. The first one is the gate delay fault model (or the Transition Fault Model [BR83]). The second one, the

*path delay* fault model [Smi85], overcomes the deficiencies inherent in the *gate delay* fault model.

#### 2.4.1.1 Gate Delay Fault Model

In the *gate delay* fault model, the AC fault is assumed to be on a line which is either a gate input or a gate output.

If a line, say a gate output, that is *slow-to-rise*, can change from logic 0 to 1 but not as fast as it should, then it is said to have a *slow-to-rise* delay fault. Similarly, a *slow-to-fall* delay fault is associated with a line if the logic 1 to 0 transition on the line takes longer than it should (the *slow-to-rise* and the *slow-to-fall* delay faults are called *transition faults*). Thus, each delay fault has two attributes associated with it :

1. the type of transition it affects;
2. the magnitude (the difference between expected delay and the actual delay.)

In [PS92] the *slack* of a path is defined as the difference between the clock period, and the propagation delay of the path (from an input to an output) under consideration.

$$\textbf{Slack} = \delta\textbf{t} - (\textbf{Propagation Delay})$$

A *negative* slack value implies that the propagation delay exceeds the clock period (*i.e.* a fault exists), whereas, a positive value of the slack implies that the propagation delay is within the clock period (*i.e.* no fault.)

With this definition, the *gate delay* fault is further classified as one whose magnitude is such that it causes:

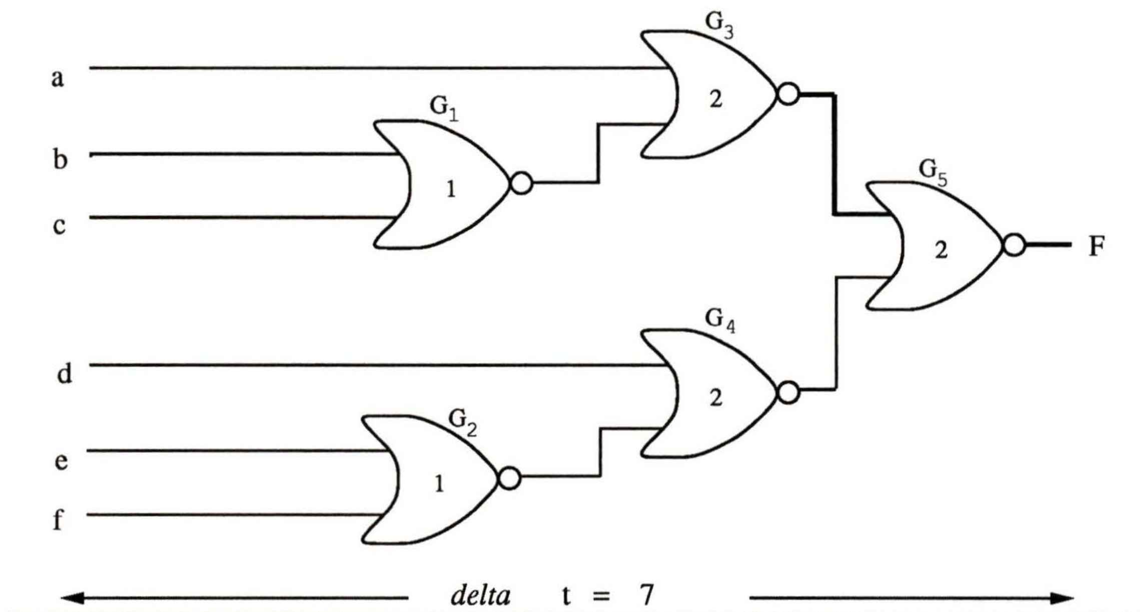
1. the slack of **at least one path** passing through the fault site to an output to be *negative.*
2. the slack of **all paths** passing through the fault site to be *negative.* This type of fault is called a *gross delay fault* [SM86].

The *transition fault* model is implicit in the gate delay fault model. This is because the former assumes that a line in a circuit is either *slow-to rise* or *slow-to fall*, which in later is a consequence of a gate having a delay fault.

#### 2.4.1.2 Path Delay Fault Model

Consider the combinational circuit of Figure 2.9, with inputs *a* through *f* and output F. The numbers within the logic gates indicate the expected delay for the respective gates.

**Figure 2.9** Deficiency of gate delay fault model.




Suppose the clock interval ($\delta$t = 7) is designed to meet the *statistical* worst case criterion. Therefore, the individual worst case delays for gates $G_1$ and $G_2$ is 3, and that for the other gates is 4. Since each gate has a distribution of delay values, assume gates $G_1$, $G_3$ and $G_5$ to have gate delay values of 2, 3 and 3 respectively. Although the delay values of the individual gates do not exceed their respective worst delays, the circuit still has a delay fault because the circuit path through gates $G_1$, $G_3$ and $G_5$ has a slack of $7 - 8 = -1$. The *gate delay* fault model is deficient in modeling such delay defects.

The *path delay fault* model considers the cumulative effect of delays along a path from an input to an output. Therefore, the *path delay fault* is able to model a delay defect which may be distributed over a region of a circuit unlike the gate delay fault which is a localized fault model.

## 2.5 Summary

This chapter has presented a review of the common fault models used in digital circuit testing. In this thesis, the testing of *transition faults* is of interest. The interest, in particular, is in enhancing the transition fault coverage by simply enhancing the transition coverage at the inputs of the CUT. The details of delay/transition fault testing will be given in chapter 4.

# Chapter 3

# Testing and Test Pattern Generators

In this chapter, some basic concepts of testing and the basic theory of test pattern generators are presented. Linear feedback shift register (LFSRs), which are commonly used as test pattern generators are described. An alternative to LFSRs is also presented.

## 3.1 Test Pattern Generation

For a given combinational circuit, it is possible to apply algorithms such as the *d*-algorithm and PODEM [BMS87] to determine a set of test vectors that detect all testable *stuck-at* faults in the CUT. It is then possible, but not practical, to store these vectors in an on-chip ROM as a source of test vectors during the test mode.

Another approach for the test pattern generation is *exhaustive testing*, which uses all input combinations as test vectors. An example of an exhaustive test pattern generator is a binary counter. For the *stuck-at fault* model, this approach guarantees to detect all testable faults. But, for a circuit with large number of inputs, it will require a prohibitively long time to go through all input combinations.

*Random testing* is a practical test pattern generation approach, where, a fraction of all possible input combinations are *randomly* generated using a low cost generator like

a *linear feedback shift register* (LFSR) [BMS87]. The LFSR will be described later in section 3.3.1. The patterns generated by a LFSR are not truly random because they are predictable and moreover, the sequence can be repeated. Thus, the patterns generated by a LFSR are called *pseudorandom* patterns. The advantage of pseudorandom patterns, over truly random patterns is that, they can be reproduced for simulation purposes. Moreover, the pseudorandom patterns can be generated repeatedly in the exact sequence to ensure repeatable results.

An alternative structure for pseudorandom test pattern generation is a *linear cellular automata* (LCAR) [SSMM90]. LCARs will be described in section 3.3.2. Although, LCARs have been found to have better randomness properties and are superior as pattern generators [ZM90], their implementation cost is higher than that of LFSRs, since LCARs require more exclusive-or (XOR) gates. In this thesis, only LFSRs are considered as test pattern generators for *transition fault* testing.

### 3.1.1 Fault Coverage

*Fault coverage* is the *fraction* of a class of logical faults in a circuit that a testing technique can detect. When two testing techniques yield different fault coverage for the same fault model, it may be fairly assumed that the difference is real, although, the fault coverage may not be accurate for the *physical faults* covered.

For a given fault model and a given testing technique, usually the fault coverage is less than 100%. When *pseudorandom* patterns, generated by a LFSR, are used as test stimuli, there is always a concern about the shift dependencies within the sequence of patterns (the shift dependency problem is described in section 4.4.1). Moreover, during a test, it is not possible to generate all patterns because it takes a very long time. Therefore, it is possible for a specific test pattern (or a test pattern pair, as in the case of $AC$ test) to be excluded from the sequence of applied patterns. If this specific test pattern (or test pattern pair) is in the test set, that detect all modeled faults in the CUT, then a 100% fault coverage cannot be achieved.

## 3.2 Data Compression

Data compression has been used in a variety of fields in order to reduce the amount of data that has to be stored and analyzed. A number of these compression techniques have been found to be useful in testing applications. Some of these techniques: signature analysis, ones count, syndrome testing and transition count, are described in [BMS87].

For testing applications, a data compressor maps a long stream of data (output responses of the CUT) into a relatively short word called a signature. An ideal compressor would yield a faulty signature for a faulty input stream. However, no such compressor exists [BMS87] and therefore, it is possible for a known compression technique to map an erroneous data stream into a correct signature. Such an action is called *error masking.* The quality of a data compressor may be measured by its error masking probability, given some standard error distribution at its input data stream [BMS87]. Although, the term data compression has been used here, the process of decompressing data is neither possible nor required in digital testing.

Some implementation of built-in self-test (BIST) use a LFSR as a compression circuit by feeding the LFSR with the output data stream (assuming only a single output). A LFSR performs a polynomial division over the Galois Field GF(2). The input stream to the LFSR represent the coefficients (either 0 or 1) of a polynomial and the LFSR itself implements a divisor polynomial. At the end of the division process, the contents of the LFSR (LFSR1, see section 3.3.1) is the remainder. The output of the LFSR, which represent the coefficients of the quotient, is normally discarded.

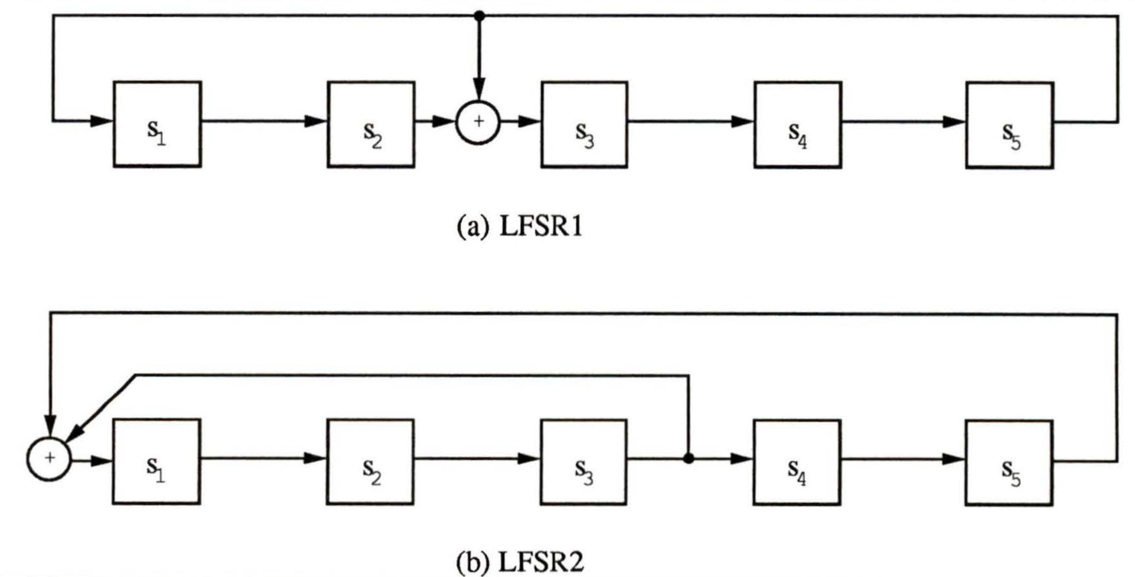
For a circuit with multiple outputs, the overhead of a single input signature analyzer on every output would be high. A preferred structure for signature analysis for multiple output circuits is a *multiple-input signature register* (MISR). The details can be found in [BMS87]. LCARs can also be used to obtain the signature for the CUT in a similar manner.

## 3.3 Pseudorandom Pattern Generators

### 3.3.1 Linear Feedback Shift Register (LFSR)

A Linear Feedback Shift Register (LFSR) is a finite state machine (FSM) consisting of storage elements (LFSR cells) chained together and controlled by a synchronous clock. Some of the connections between the storage elements contain XOR gates. Two possible orientations of a five-stage LFSR are shown in Figure 3.1. Each stage, $s_i$, is a storage element (*e.g. a flip-flop*) and the feed back line from the output of the last stage is connected to some of the LFSR cells either, directly or, via XOR gates ($\oplus$). The storage elements hold a binary value and the XOR gates perform modulo 2 addition. When the XOR gates are between the LFSR cells, as in Figure 3.1 (a), the LFSR is referred to as LFSR1 [BMS87, page 119] (Type 2 LFSR in [ABF90, page 434]). When the XOR gates are implemented externally, as in Figure 3.1 (b), then the LFSR is referred to as LFSR2 [BMS87, page 119] (Type 1 LFSR in [ABF90, page 434]).

**Figure 3.1** Linear feedback shift registers.




(a) LFSR1

(b) LFSR2

The state of the LFSRs in Figure 3.1 is a binary five-tuple. These binary values correspond to the values stored in the memory cells $(s_1, s_2, s_3, s_4, s_5)$. Like any other finite

state machine, the LFSRs have a set of finite state machine equations which describe the transition to the next state. The equations for the LFSR in Figure 3.1 (a) are:

$$\begin{aligned} s_1^+ &= s_5, \\ s_2^+ &= s_1, \\ s_3^+ &= s_2 \oplus s_5, \\ s_4^+ &= s_3, \\ s_5^+ &= s_4. \end{aligned}$$

where $s_i^+$ indicates the next state of stage $i$ of the LFSR. If the state of the LFSR is given as $S = [s_1, s_2, s_3, s_4, s_5]$ and $T$ is a $5 \times 5$ binary matrix, the next state of the LFSR, $S^+$ is given by $S^+ = ST$, where the matrix multiplication is over GF(2). Therefore, given the above set of equations for the LFSR1, the corresponding transition matrix, $T$, is:

$$T = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Let the sequence $\{a_m\} = \{a_0, a_1, a_2, ...\}$, represent the history of the output stage of a LFSR, where $a_i$ is a 0 or a 1 depending on the state of the output stage at time $t_i$. The definition for the **characteristic polynomial**, $f(x)$, of the sequence, $\{a_m\}$, and of the LFSR which produces the $\{a_m\}$ is given in [BMS87, page 66]. The characteristic polynomial of the LFSRs in Figure 3.1 is $f(x) = |xI - T| = x^5 + x^2 + 1$, where $I$ is an identity matrix. Note that the output of stage-1 represents $x$ or $x^1$, the output of stage-2 is $x^2$, and so forth.

The **reciprocal polynomial**, $f^*(x)$, of a given characteristic polynomial, $f(x)$, is defined as:

$$f^*(x) = x^n f(1/x).$$

Therefore, the reciprocal of the above mentioned $f(x)$ is:

$$f^*(x) = x^5 \left(1/x^5 + 1/x^2 + 1\right) = x^5 + x^3 + 1$$

To implement a LFSR1, corresponding to a given characteristic polynomial, $f(x)$ the $n$ stages of a LFSR are numbered from 1 to $n$ beginning at the left-most stage. The outputs of stages, other than the last stage, corresponding to the exponents of $x$ in the $f(x)$ are connected to XOR gates. The second (common) input to these XOR gates is the output of the last stage (stage-n). The output of the last stage is also connected to the input of the first stage. The output of these XOR gates (with inputs from $s_i$ and $s_n$) are connected to the next ($s_{i+1}$) stage. For example, the LFSR1 in Figure 3.1 (a), with characteristic polynomial $f(x) = x^5 + x^2 + 1$ has a single XOR gate at the output of stage-2, corresponding to the term $x^2$. The output of the last stage (stage-5) is connected to the XOR gate, as well as to the input of the first stage. The output of this XOR gate is connected to the next stage, stage-3.

To implement a LFSR2 corresponding to a given $f(x)$, the $n$ stages are numbered as above. The outputs of stages corresponding to the exponents of $x$ in the *reciprocal polynomial*, $f^*(x)$, are selected and connected to an external XOR gate. The output of the XOR gate is connected to the input of stage-1.

An implementation of LFSRs, as mentioned above, will realize the sequence $\{a_m\}$ associated with the given polynomial at any output. A characteristic of LFSRs is that if they are initialized with a non-zero state, $S \neq [0, 0, ..., 0]$ , they cycle through a number of states.

An *autonomous*[1] LFSR is cyclic in the sense that when clocked repeatedly, it goes through a fixed sequence of states. For an n-stage shift register, there are at most $2^n$ possible states. In the case of a LFSR, the successor of the all zero state is itself, since the output of the feedback circuit is 0 when all of its inputs are 0. Hence, a LFSR starting from a non-zero state can reach at most $2^n - 1$ different states. Therefore, the sequence generated by any stage is periodic with a period no greater than $2^n - 1$. The proof for

[1]A LFSR that has no external inputs, except for clocks.

theorem 3.1 is given in [BMS87, pages 69–70].

**Theorem 3.1** *Given a LFSR, initialized such that the first $n-1$ stages contain 0 and the $n$ th stage contains a 1, then the LFSR sequence $\{a_n\}$ is periodic with a period which is the smallest integer $k$ for which $f(x)$ divides $1-x^k$.*

Another interesting property concerning the periodicity of shift register sequences is that, when $f(x)$ is irreducible, the period is independent of the initial conditions (except for all zeroes case).

**Definition 3.1** *If the sequence generated by an $n$-stage LFSR has a period $2^n-1$, it is called the* **maximal length** *sequence or m-sequence. The characteristic polynomial of a maximal length sequence is called a* **primitive polynomial.** *From theorem 3.1, the smallest $k$ for which an $n$ th degree primitive polynomial divides an expression of the form $1-x^k$ is $k=2^n-1$.*

The characteristic polynomial of the LFSRs in Figure 3.1, $f(x)=x^5+x^2+1$, is *primitive.* Therefore, the sequence generated has period $2^5-1$, which is the maximal length sequence (m-sequence) for a 5-stage LFSR. Some of the properties for a given m-sequence, $\{a_n\}$, with a primitive characteristic polynomial $f(x)$ of degree "$n$" are [BMS87, pages 77–80]:

**Property 3.1** *The period of $\{a_n\}$ is $p=2^n-1$,* i.e., *$a_{p+i}=a_i$ for all $i\geq 0$.*

**Property 3.2** *Starting from a non-zero state, the LFSR that generates $\{a_n\}$ goes through all $2^n-1$ states before repeating.*

**Property 3.3** *If a window of width $n$ is slid along an m-sequence, then each of the $2^n-1$ non-zero binary n-tuples is seen exactly once in a period.*

Sequences generated by LFSR are not strictly random because it is possible to reset the sequence generator so that the same sequence is produced at another time. Since the sequences are repeatable, they are called *pseudorandom* sequences. The properties of these sequences can be found in [BMS87, pages 77–80] and some of which are listed below.

**Property 3.4** *The number of ones in an m-sequence differs from the number of zeroes by 1.*

**Property 3.5** *An m-sequence produces an equal number of runs*[2] *of ones and zeroes.*

**Property 3.6** *In every period of m-sequence, one-half the runs have length 1, one-fourth have length 2, one-eighth have length 3, and so forth, as long as the fractions yield integral number of runs.*

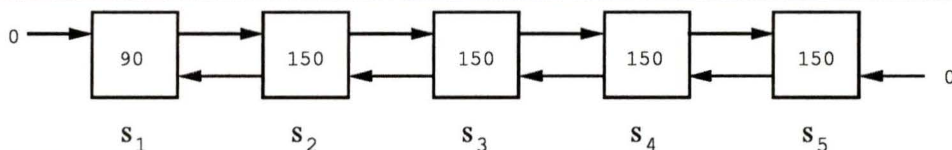
These randomness properties make it feasible to use LFSRs as test sequence generators in BIST circuitry.

### 3.3.2 Linear Cellular Automata Register (LCAR)

A linear one-dimensional cellular automata register (LCAR) is a one dimensional array of cells where each cell communicates with its two neighbours (except for the first and the last cells of the array which have a single neighbour only.) The state of each cell is represented by a binary value (0 or 1), that is stored by the cell. The next state of a cell is determined by applying specific linear rules on the current state of the neighbouring cells and the current state of the cell itself.

An example of a 5-cell LCAR is shown in Figure 3.2. The cell whose next state is determined by the current state of its neighbouring cells is referred to as a Rule 90 cell. The next state of Rule 150 cell is determined by the current states of not only its neighbours but also of the cell itself. Only LCARs with a combination of Rule 90 and Rule 150 cells yield a maximal length cycle [SSMM90].

**Figure 3.2** Autonomous linear cellular automata register.




[2] A maximal contiguous grouping of symbols is called a "run".

At every clock cycle, each cell, $s_i$, receives an input from its nearest neighbours, $s_{i-1}$ and $s_{i+1}$. In an autonomous LCAR, the boundary cells receive a 0 external input. The computation rules 90 and 150 are defined as follows:

$$\begin{aligned} Rule\ 90 &: s_i^+ = s_{i-1} \oplus s_{i+1} \\ Rule\ 150 &: s_i^+ = s_{i-1} \oplus s_i \oplus s_{i+1} \end{aligned}$$

Every LCAR has a set of next state equations and a corresponding state transition matrix, $T$. The characteristic polynomial of the $T$ is the characteristic polynomial of the LCAR. The next state equations for the LCAR in Figure 3.2 are:

$$\begin{aligned} s_1^+ &= s_2 \\ s_2^+ &= s_1 \oplus s_2 \oplus s_3 \\ s_3^+ &= s_2 \oplus s_3 \oplus s_4 \\ s_4^+ &= s_3 \oplus s_4 \oplus s_5 \\ s_5^+ &= s_4 \oplus s_5 \end{aligned}$$

If the state of the LCAR is given as $S = [s_1, s_2, s_3, s_4, s_5]$ and $T$ is a $5 \times 5$ binary matrix, the next state of the LCAR, $S^+$ is given by $S^+ = ST$, where the matrix multiplication is over GF(2). The transition matrix, $T$, corresponding to the above next state equations is:

$$T = \begin{pmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

The characteristic polynomial, $f(x) = x^5 + x^2 + 1$, of the above transition matrix is called the characteristic polynomial of the LCAR. As mentioned earlier (section 3.3.1), this $f(x)$ is primitive and hence the LCAR of Figure 3.2 produces a maximal length cycle.

There are two advantages of LCARs as opposed to LFSRs. First, all connections between cells are near neighbour connections, thus avoiding possible routing problems and long feedback loops; and the second, it has been shown that LCARs can be better pseudorandom pattern generators and can give better results when used in the detection

of delay faults. The disadvantage of LCARs is that they require more XOR gates than LFSR.

## 3.4 Summary

In this chapter, the basic theory of test pattern generators was described. LFSRs, which are commonly used as test pattern generators, and an alternative to LFSRs, the LCARs, were presented. In this thesis, the shift dependency problem exhibited by the test patterns from an LFSR1 are considered and a low cost solution to this problem is presented.

# Chapter 4

# Delay Fault Testing

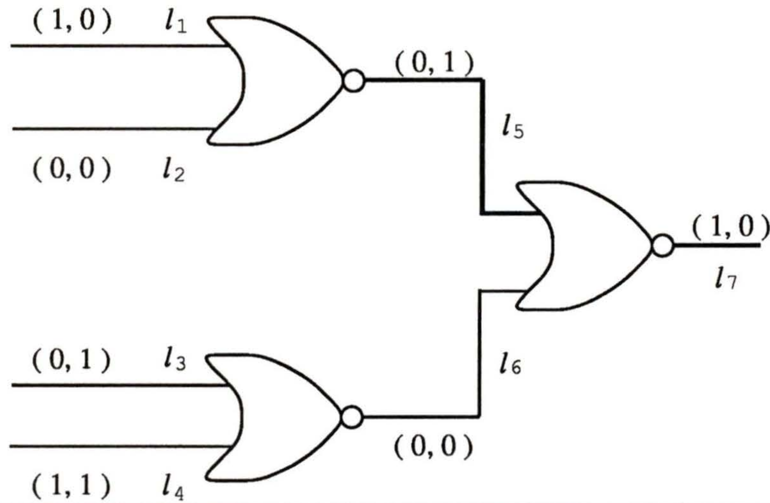
The different types of delay fault models were described in section 2.4.1. This chapter describes the requirements and methods for testing delay faults. Some concerns of delay fault testing are also mentioned (section 4.2). The *gate delay* fault model is implicit in the *transition fault* model. Therefore, the requirements and protocol for *transition fault* testing is given in section 4.3. A problem related with a low cost method used for *transition fault testing* is pointed out in 4.4.1. Section 4.5 describes a few methods that try to overcome this problem.

## 4.1 Requirements of Delay Fault Testing

As mentioned in section 2.4, a property of delay faults is that the propagation delays within the circuits exceed their limit. As a result, the signals arrive at the outputs of the circuit under test (CUT) after the sampling of the outputs has already taken place.

Consider line $l_5$ in Figure 4.1. Assuming it has a *slow-to-rise* transition fault [BR83] that results in the propagation delays of all paths from line $l_5$ to exceed their limits.

**Figure 4.1** Delay fault testing.




In order to detect this slow-to-rise transition fault, two test patterns are required:

1. the first pattern initializes the line $l_5$ to the zero logic value;

2. the second pattern causes a 0 to 1 transition on line $l_5$ **and** propagates the transition to the output of the combinational logic (CUT).

## 4.2 Concerns in Delay Fault Testing

Some concerns in delay fault testing are:

1. Presence of hazards – which invalidate a test and cause a faulty circuit to be erroneously declared good.

2. Fault Masking – The delay in one part of a circuit masks the effect of faults in other parts of the circuit.

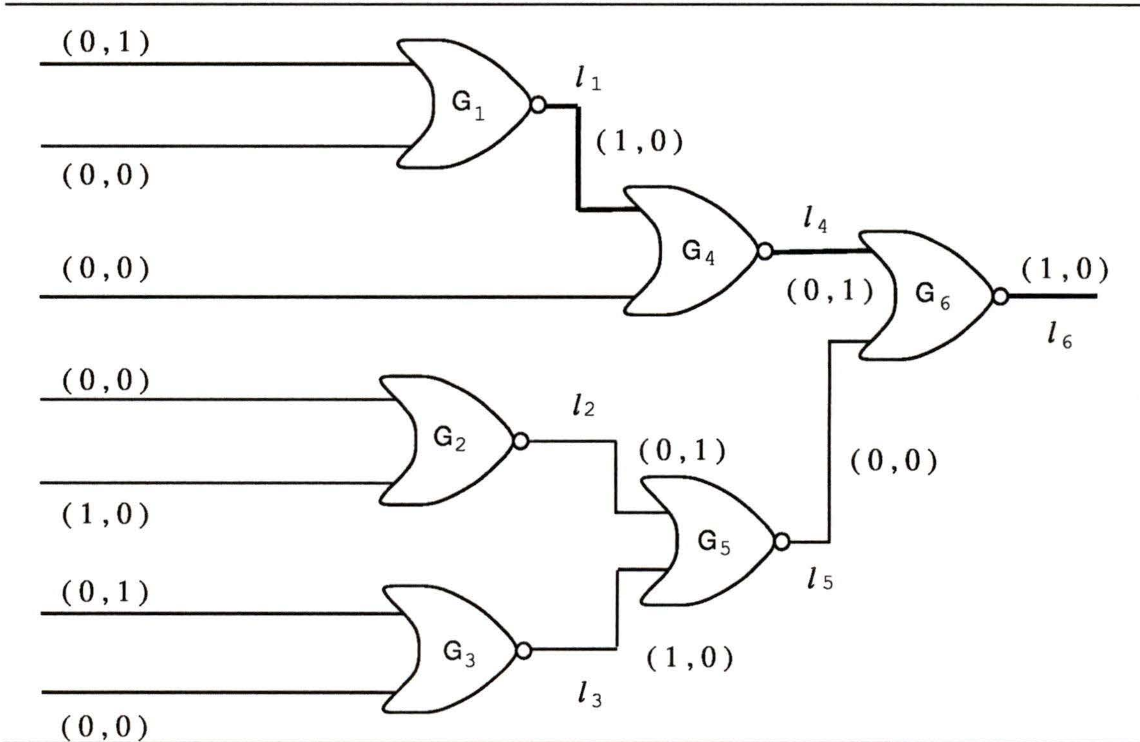
### 4.2.1 Hazards

Hazards are unintended transitions (spikes) that occur during a delay test. The presence of hazards can potentially invalidate a test by letting the circuit pass a test despite the presence of a delay fault.

Consider a slow-to-fall transition fault on line $l_1$ in Figure 4.2. The pair of patterns (0000100, 1000010) can detect the presence of the slow-to-fall transition fault under consideration. A pair of binary values (a,b) on each line indicates the logic value carried by the line when each pattern is applied. The given pair of pattern detects the slow-to-fall transition fault on line $l_1$ as follows:

1. The first pattern initializes line $l_1$ to 1;
2. The second pattern creates a 1 to 0 transition on line $l_1$ **and** propagates the transition to the output via the path $l_1 l_4 l_6$.

**Figure 4.2** Hazards in delay fault testing.



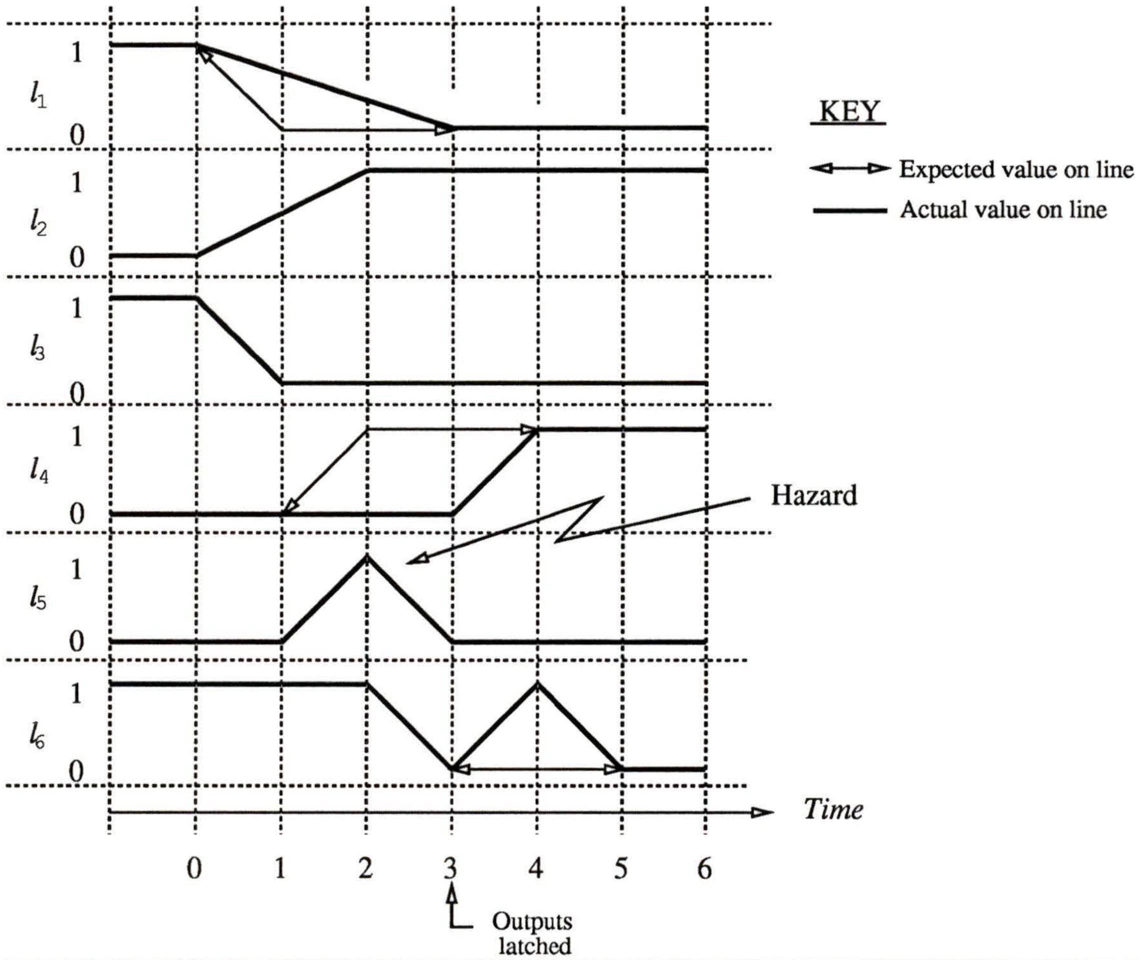

Although the second pattern propagates the 1 to 0 transition to the output, this transition, under certain conditions, may include an unintended transition that can invalidate the test.

Suppose the propagation delay of gate $G_2$ is greater than that of gate $G_3$. The inputs

to gate $G_5$ will change from $(l_2, l_3) = (0,1)$ to $(l_2, l_3) = (0,0)$ and stabilize at $(l_2, l_3) = (1,0)$ (See the timing diagram in Figure 4.3.)

These transitions result in a momentary rising spike (hazard) at line $l_5$ and can potentially cause a falling spike at the output ($l_6$) as seen in the timing diagram of Figure 4.3. The timing of the falling spike at the output depends on the gate delay fault of the circuit.

**Figure 4.3** Unintended spike (hazard) during delay fault testing.



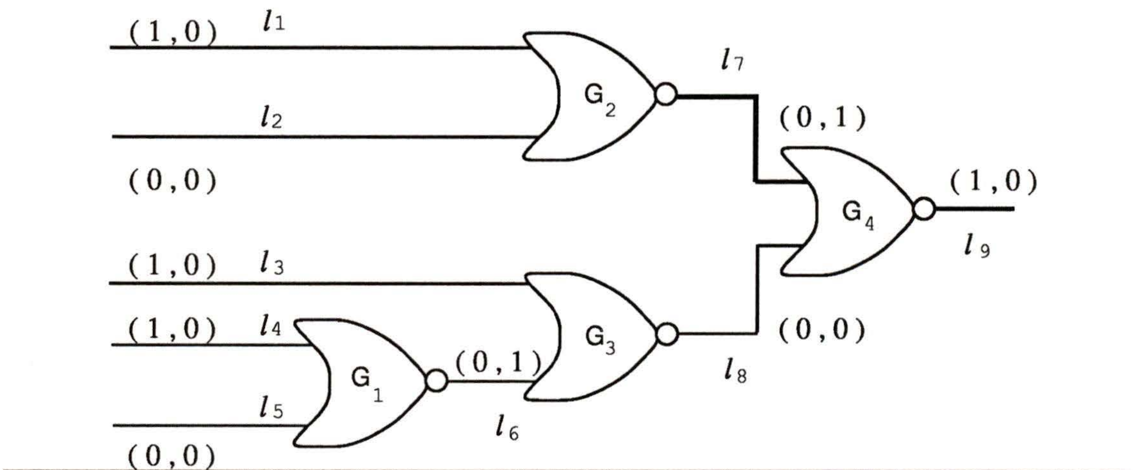
The delay due to the *slow-to-fall* transition on line $l_1$, will cause the final transition on $l_6$ to take place after the output has been latched. If by chance the output is latched at the falling spike (at t=3 in Figure 4.3), the delay fault will go undetected due to the hazard on line $l_5$.

### 4.2.2 Fault Masking and Robust Test

Another concern in delay fault testing is the effect of a fault in one part of the circuit getting masked due to delays in other parts of the circuit.

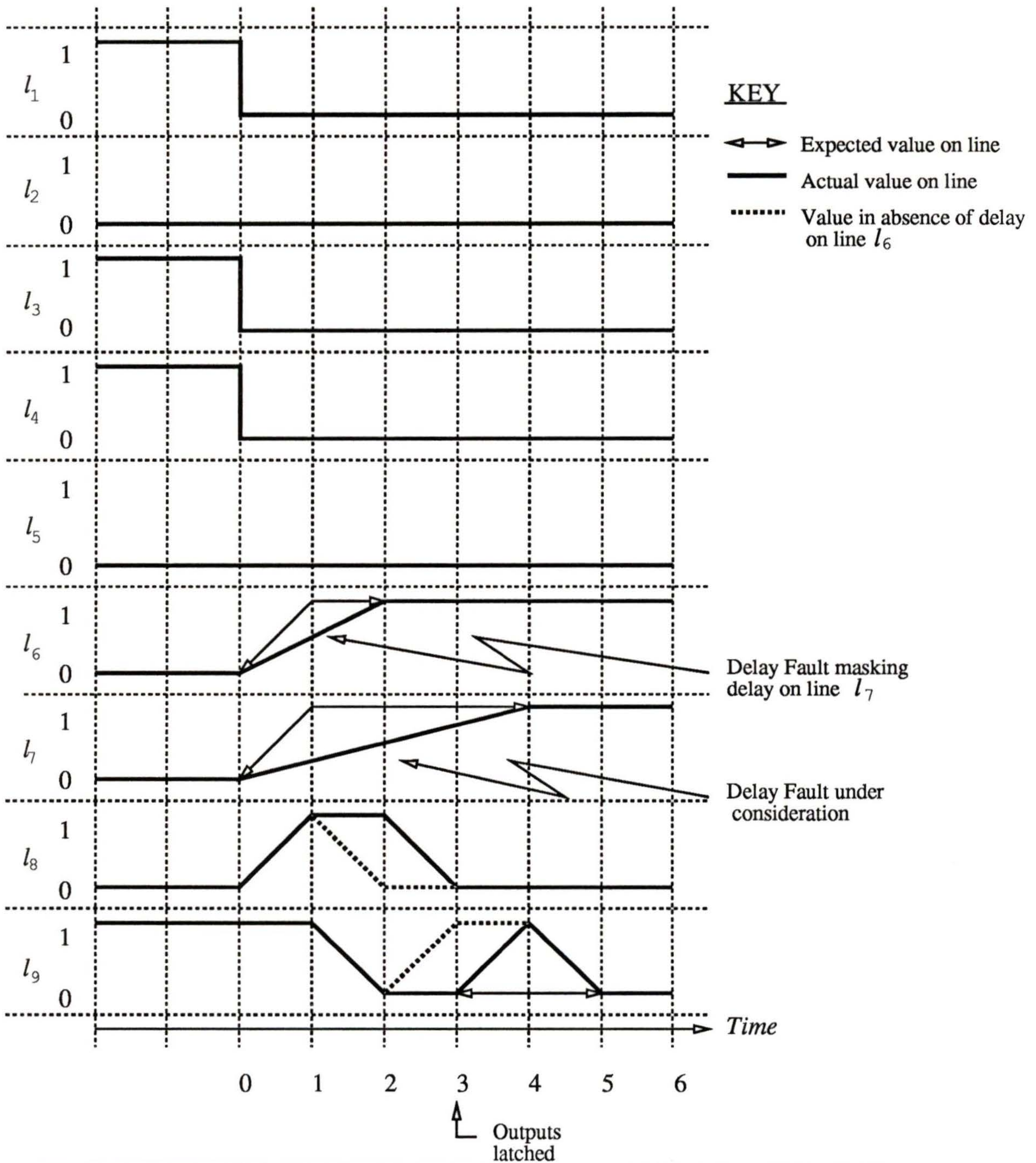
Consider the example in Figure 4.4. The *slow-to-rise* transition fault on line $l_7$ can be detected by the pair of patterns (10110, 00000). The pairs of binary values (a,b) on each line indicate the logic value on the line due to first and second patterns at the inputs respectively.

**Figure 4.4** Masking during delay fault testing.




Suppose line $l_6$ has a *slow-to-rise* transition fault as well. This will cause a temporary 0 to 1 (and finally to 0) transition on line $l_8$ (Figure 4.5.) So, during the test for the *slow-to-rise* transition fault on line $l_7$, the temporary 0 to 1 transition of line $l_8$ erroneously drives the output to 0 while the line $l_7$ is not yet 1. In the absence of a *delay fault* on line $l_6$, the output $l_9$ would be at logic 1 when the output is latched (at time $t = 3$) and the delay fault on line $l_7$ could then be detected. Thus, the *slow-to-rise* transition fault

**Figure 4.5** Timing diagram for the test in Figure 4.4.



on line $l_7$ gets masked (goes undetected) due to a (slow-to-rise transition) delay in other part ($l_6$) of the circuit.

The example in Figs. 4.4 and 4.5 also includes a hazard on line $l_8$ (between $t = 0$ and 2 in Figure 4.5). But this example of a hazard does not invalidate the test for the slow-to-fall transition test on line $l_7$.

Tests which ensure that the delay test is valid irrespective of delays in other parts of the CUT are called **robust tests** [Smi85]. However, robust tests do not exclude the existence of hazards (section 4.2.1) in the position of the circuit being tested. **Hazard free** tests are a subset of robust tests, and they exclude the existence of hazards in the signal values used to propagate the fault effect [SM86].

## 4.3 Transition Fault Testing

Robust test and hazard free test were briefly mentioned in sections 4.2.2 and 4.2.1 respectively. Both tests impose severe constraints on the signals associated with the test. Generation of robust tests is more complex than generation of non-robust tests. Also, the coverage attainable by robust tests is generally lower, and never better, than that attainable by non-robust tests [RLP87].

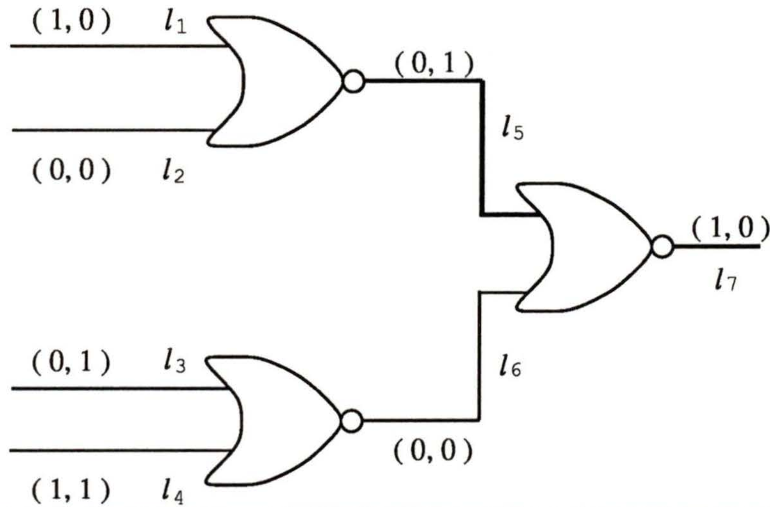
The transition fault model [BR83] allows one to generate delay tests without prohibitive increase in the cost of test generation and fault simulation [PS92]. The tests generated for the transition fault model are called *transition tests* which can either be robust or non-robust.

The requirements of a transition test are [PS92]:

1. A pair of patterns that creates the appropriate transition on the faulty line;

2. The second pattern of the pair tests the corresponding *stuck-at fault.*

**Figure 4.6** Transition fault testing.




For example, in Figure 4.6, the pair of patterns (1001, 0011) detects the slow-to-rise transition fault on line $l_5$ by:

1. creating a 0 to 1 transition on line $l_5$; and

2. the second pattern (0011) is a test for line $l_5$ *stuck-at-zero.*

By reversing the order in which the patterns are applied, the test pair detects a slow-to-fall transition fault on line $l_5$ by:

1. creating a 1 to 0 transition on line $l_5$; and

2. the second pattern (in this case 1001) is a test for line $l_5$ *stuck-at-one.*

An important point to be noted in the requirements of the transition test is that the test only has one extra condition over the test for a *stuck-at-fault.* Hence, it is expected that the transition fault coverage will be very close to the *stuck-at-fault* coverage for the same CUT [PS92].

It is also pointed out in [PS92] that the results obtained for *transition fault* coverage are for *gross delay faults,* and more detailed analysis is necessary to evaluate the coverage for smaller delay faults [IRW90]. However, in timing optimized designs, it can be shown

that the *transition fault* coverage can be used to accurately predict the actual delay fault coverage, even when arbitrary delay fault sizes are assumed [PUWM91].

The remaining sections and chapters are all concerned with *transition tests* and *transition fault* coverage.

## 4.4 Transition Fault Testing with a LFSR

An effective *transition test* requires test patterns to be generated at machine speed. As seen in the earlier sections, the *transition test* requires pairs of patterns and therefore the test pattern generator should be able to generate arbitrary test pattern-pair. Some generators were mentioned in section 3.3. Savir and Berry [SR92] give references to generators that include extensions to the normal shift register latch used in level sensitive scan design (LSSD), but they all suffer from high overhead.

LFSRs that implement a primitive polynomial ([BMS87], pages 75–76) are good candidates for test pattern generators as far as implementation cost is concerned. When LFSRs are used as the source of test patterns, the test protocol is such that a new pattern is applied at the inputs of the CUT after the application of every LFSR shift clock. Since the maximum number of patterns that a LFSR of size $n$ can generate is $2^n - 1$, then with the above mentioned protocol the maximum number of test pattern pairs that can be applied at the inputs of the CUT is $2^n - 1$.

In [SR92], a measure is proposed for assessing how well a generator can apply AC patterns (patterns for transition test) to the CUT. This measure, called the *AC Strength* (AS), for a pattern generator is defined as the fraction of the exhaustive two-pattern count that it can apply to the CUT given sufficient time.

The AS of a generator takes values between 0 and 1. The higher the AS value, the better the performance of the generator. An ideal generator (of size $n$) is one which can generate $2^n(2^n - 1)$ test pattern-pair and therefore has AS of 1.

### 4.4.1 Shift Dependency Problem with a LFSR

LFSRs of size n that implement a primitive polynomial have an AC Strength of AS = $(2^n - 1)/2^n(2^n - 1) = 2^{-n}$. This low AS value indicates that LFSRs may not perform well as AC pattern generators.

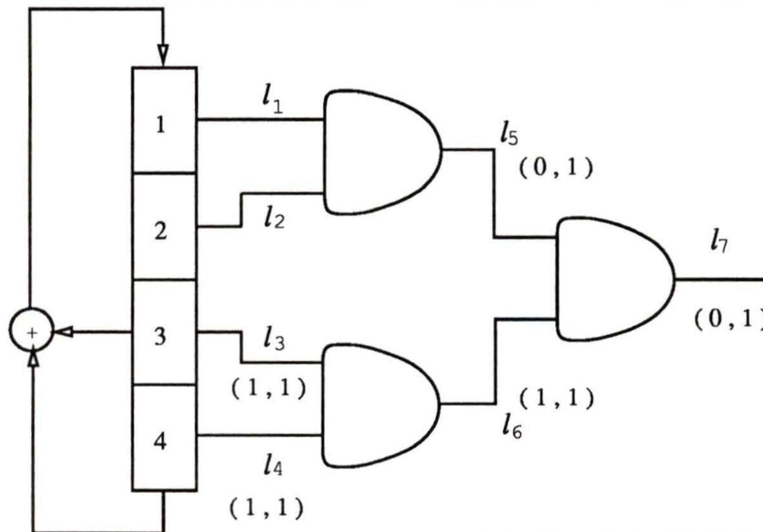
A delay test protocol, called the *skewed-load transition test* or SLOTT is described in [Sav92]. It is similar to the test protocol used for LFSR that are implemented with external XOR gates (section 3.3.1.) The second pattern ($P_2$) of the test pair is obtained by shifting the first pattern ($P_1$) one bit down the scan chain and appending to it an extra bit through the scan-in port. Thus, except for the appended bit, $P_2$ is a one-bit shift over its predecessor, $P_1$.

Patil and Savir [PS92] discuss the loss of coverage due to shift dependency in SLOTT and also give a lower bound for *transition fault* coverage achieved using the SLOTT protocol.

**Shift Dependency Problem**

Consider the *slow-to-rise* transition fault on line $l_5$ in Figure 4.7. The pair of patterns that can detect this fault are (0X11, 1111) and (X011, 1111), where X represents a 0 or a 1.

**Figure 4.7** Shift dependency problem with LFSR.



Although, there are 4 different test pattern-pair that can detect the fault under consideration, not a single pair can be generated by the LFSR (that implements a primitive polynomial) shown in Figure 4.7.

Recently, some methods that try to overcome the shift dependency problem by re-arranging the connections between the test pattern generator and the inputs to the CUT have appeared in the literature [ZBM92], [SR92]. Section 4.5 briefly describes these methods and shows the fault coverage achieved.

## 4.5 Overcoming the Shift Dependency

*Pseudorandom* testing using low cost test pattern generators is a convenient way of testing circuits. But, the shift dependency problems, as pointed earlier, may not give a high *transition fault* coverage. In [SR92], some references are given to generators that include extensions to the normal shift register latch (SRL) used in *level sensitive scan design* (LSSD). But these designs suffer from high hardware overhead.

Two methods have been suggested in the literature that try to overcome the shift dependency problem by re-arranging the connections between the generator outputs and the inputs to the CUT. These methods are described in sections 4.5.1 and 4.5.3. Although each method gives a good transition fault coverage, each has some disadvantages. In this thesis, an algorithm is given that separates inputs and also overcomes the disadvantages.

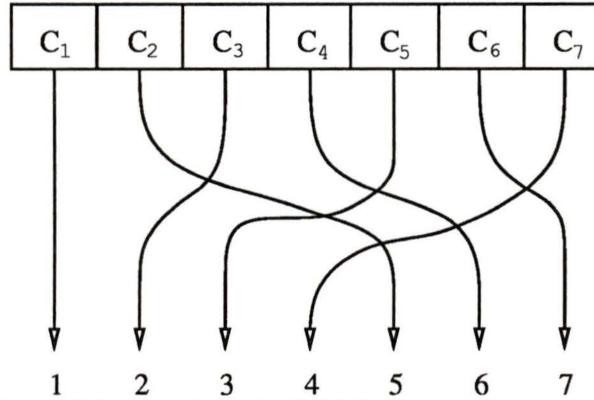
### 4.5.1 Cross Over Permutation

The Cross Over permutation, suggested in [ZBM92], assigns the odd numbered outputs[1] of the generator to the first $\lceil n/2 \rceil$ inputs of the circuit under test (CUT). The even numbered outputs of the generator are assigned to the remaining inputs of the CUT (Figure 4.8.)

Consider a LFSR with external XOR gates implementing a primitive polynomial. The advantage of permuting the outputs of the generator in the cross over fashion is that every sub window of the XLFSR of size less than $\lceil n/2 \rceil$ can potentially see all pairs of binary

[1] The leftmost output of the generator is labeled 1 and hence is an odd numbered output.

**Figure 4.8** Cross over permutation.




patterns, provided the generator is run for a sufficiently long time [ZBM92].

Hence, this scheme solves the shift dependency problems for any arbitrary window of size less than $\lceil n/2 \rceil$ of a LFSR of size n with external XOR gates. The transition fault simulation results using 100,000 test patterns for the 10 ISCAS '85 combinational benchmark circuits [BF85], using both, NP-CXN (*null permutation connections* between the generator outputs and the circuit inputs) and XLFSR scheme, are given in Table 4.1[2]. For each scheme, two columns of results are given (FBf and FBl). These columns are for the two possible orientations of a LFSR where in one the *feedback* goes into the first cell of the LFSR (FBf) and in the other, the *feedback* goes into the last cell (FBl). The results indicate significant improvement over the traditional method (NP-CXN) of leaving the generator outputs unshuffled (*i.e.* output number k of the generator connected to primary input k of the CUT.)

### 4.5.2 Random Connections

The special permutation of the generator outputs mentioned in section 4.5.1 is an attempt to enhance the transition coverage for arbitrary windows of size k within a larger window of size n ($n > k$.) This special permutation makes a LFSR a good candidate for sequential fault (*delay* and *stuck-open*) testing [ZBM92].

[2]These results and, unless otherwise stated, all other fault coverage results given in this thesis were obtained by implementing the methods and running simulations.

**Table 4.1** Transition fault coverage for the ISCAS '85 circuits using NP-CXN and XLFSR.

| | | NP-CXN | | XLFSR | |
|---|---|---|---|---|---|
| Cct. | Faults‡ | FBf Coverage (%) | FBl Coverage (%) | FBf Coverage (%) | FBl Coverage (%) |
| c432 | 864 | 97.22 | 94.68 | 98.84 | 98.84 |
| c499 | 998 | 99.10 | 99.10 | 99.20 | 99.20 |
| c880 | 1760 | 96.31 | 97.16 | 99.50 | 99.60 |
| c1355 | 2710 | 97.08 | 97.23 | 98.97 | 99.00 |
| c1908 | 3816 | 97.69 | 97.96 | 99.42 | 99.34 |
| c2670 | 5492 | 88.75 | 89.31 | 93.19 | 93.65 |
| c3540 | 7080 | 90.42 | 90.54 | 93.49 | 92.90 |
| c5315 | 10630 | 97.11 | 97.15 | 99.21 | 99.20 |
| c6288 | 12576 | 99.20 | 99.06 | 98.98 | 99.00 |
| c7522 | 15106 | 97.29 | 97.45 | 97.84 | 97.79 |

‡ Fault collapsing techniques were not applied.

Because the simple reordering of the generator outputs has a significant effect on transition fault coverage, it was of interest to see the effect of *random* connections. Hence, some experiments were conducted to see the effects of *random* connections on *transition fault* testing. The experiments included the following:

1. A hundred, randomly generated, distinct connections of the generator outputs were used. These permutations *excluded* the *null permutation connections* and the cross over permutation (section 4.5.1.)
2. A fixed seed value was used in all experiments. The seed was a vector of alternating binary values beginning with 1 (*i.e.* 101010....)
3. The test pattern generator was an appropriate sized minimal cost LFSR with external XOR gates implementing a primitive polynomial.

4. For each simulation, a 100,000 test patterns were used.

5. Since patterns generated by a LFSR with external XOR gates exhibit a shift property, the type of shifting (either right-shift or left-shift) is decided by the two possible orientations of the LFSR (FBf and FBl). Therefore, for each permutation in (1), two simulations were run, each using one of the two orientations of the LFSR.

**Importance of LFSR Orientations**

The orientation of the generator was considered to be of importance because of the following reasons:

1. The maximum number of test pattern pairs for a CUT with $n$ inputs are $2^n(2^n - 1)$. A LFSR of the same size can only apply a maximum of $2^n - 1$ test pattern pairs. Out of these $2^n - 1$ only $100,000 - 1$ pairs are applied during simulations. Therefore the fraction of all possible test pattern pairs that are applied is only $(99,999/(2^n(2^n - 1))$. For example, for the c6288 ISCAS circuit with 32 inputs, this fraction is only about $5.421 \times 10^{-15}$.

2. The order in which the test patterns in a test pair are applied, is significant in *transition fault* testing.

3. In both orientations, although every cell of the LFSR goes through the same sequence of binary values, a different sequence of patterns are applied at the inputs of the CUT.

In practice, only one orientation of the LFSR will be implemented. Since there is no difference in their implementation cost, it is wise to know and to implement the one that gives the better coverage.

The best result for each circuit was selected using the following rules:

1. For each orientation of the generator:

   (a) The permutation with the best observed transition fault coverage was selected.

   (b) In cases where, more than one permutation had the best observed transition fault coverage, the one with the shortest test length was considered to be the best one.

2. The orientation (FBf or FBl) of the LFSR is decided by the better result of the two in Step 1.

**Results and Conclusions**

The results of the experiments for the 10 ISCAS '85 circuits are given in Table 4.2. It is seen that some of the results obtained were better than those for XLFSR (Table 4.1.) The conclusion is that , although the XLFSR overcomes the shift dependency problem to a certain extent, the *transition fault* coverage can be further enhanced by some careful re-arrangement of the connections between the outputs of the test pattern generator and the inputs of the CUT.

Since the *cross over* permutation (XLFSR) is identified based on the structure and property of the test pattern generator (LFSR) and *not* based on the structure of any CUT, the permutation may not work well for all circuits. As an example, for one of the ISCAS '85 circuit, c6288, which is a 16 bit multiplier (32 inputs and 32 outputs), the XLFSR does worse than NP-CXN (Table 4.1.) After analyzing the structure of c6288 to see the effect of the cross over permutation, it was found that each *logic cone* (output cones) in the circuit, including the one with the minimum number of inputs (2 inputs), is fed from adjacent generator outputs.

Savir and Berry [SR92] have suggested a method which analyzes the structure of the CUT and identifies the connections between the inputs of the CUT and the shift register latches (SRLs) in the scan chain. Their method (described in section 4.5.3) is based on identifying output cones (logic cones) and avoiding two adjacent generator outputs from feeding circuit inputs that belong to the same logic cone.

The conclusion is that the transition fault coverage can be further enhanced when structural information of the CUT is used instead of relying solely on the structure of the generator as in the case of XLFSR, which enhances transition coverage of arbitrary windows of specific lengths.

**Table 4.2** Best observed transitions fault coverage for the ISCAS '85 circuits using random connections.

| Cct. | Faults$\xi$ | FBf | | FBl | |
|---|---|---|---|---|---|
| | | Und.† | Cvg.‡(%) | Und. | Cvg. (%) |
| c432 | 864 | 10 | 98.84* | 10 | 98.84 |
| c499 | 998 | 8 | 99.20 | 8 | 99.20 |
| c880 | 1760 | 14 | 99.20 | 2 | 99.89* |
| c1355 | 2710 | 25 | 99.08* | 31 | 98.86 |
| c1908 | 3816 | 21 | 99.45 | 11 | 99.71* |
| c2670 | 5492 | 888 | 83.83 | 637 | 88.40* |
| c3540 | 7080 | 276 | 96.10* | 310 | 95.62 |
| c5315 | 10630 | 66 | 99.38* | 99 | 99.07 |
| c6288 | 12576 | 94 | 99.25* | 95 | 99.24 |
| c7522 | 15106 | 548 | 96.37* | 582 | 96.15 |

‡Fault Coverage. * Best observed result.

†Number of undetected faults.

$\xi$ Fault collapsing techniques were not applied.

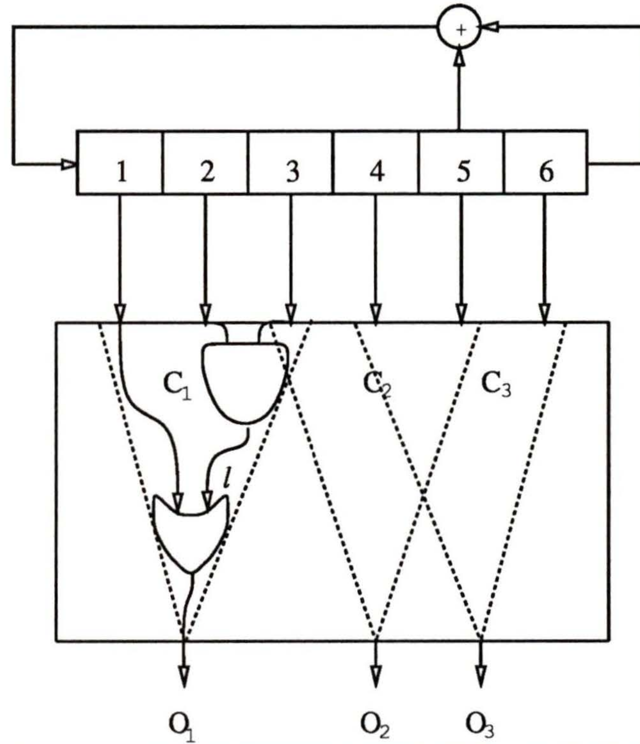
### 4.5.3 Input Separation using Dummies (ISUD)

Savir and Berry point out in [SR92] that if a cone of a circuit is driven from contiguous shift register latches (SRLs), it can only exercise a small fraction of all possible pairs of patterns that the CUT might require for an effective AC test (transition test is also called an AC test [SR92].)

Consider the circuit shown in Figure 4.9 with six inputs. The circuit has three outputs ($O_1, O_2$, and $O_3$) and therefore three logic cones. The first cone ($C_1$) is driven by the first three cells of the test pattern generator (TPG). The second cone ($C_2$) is driven by the 3rd, 4th and 5th cells. The last cone ($C_3$) is driven by the last three cells of the TPG. Ideally, if the generator runs for a sufficiently long time then each cone should be able to

see a total of $2^3(2^3 - 1) = 56$ distinct test pairs.

**Figure 4.9** Shift dependency problems within logic cones.




Consider the first cone ($C_1$) driven by the first three cells of the TPG. Suppose, a test pattern 010XXX (where X is 1 or 0) is applied as the first pattern of a test pair. With this pattern at the inputs of the circuit, the pattern at the inputs of $C_1$ is 010. The next pattern that will make a pair required for a transition test within $C_1$ can either be a 001 or a 101. In other words, there are only 2 test-pattern pairs out of the possible 7 for $C_1$ with the initializing pattern as 010, *i.e.* (010, 001) and (010, 101).

Suppose, the line $l$ in $C_1$ has a *slow-to-rise* transition fault. The 3 test-pattern pairs that can detect this fault are (010, 011), (001, 011) and (000, 011), *none* of which can appear at the inputs of $C_1$. Hence, the *slow-to-rise* transition fault on line $l$ cannot be tested. Note that, if the orientation of the generator is reversed (from FBf to FBl), then the test-pattern pair (001, 011) can potentially appear at the inputs of $C_1$ and detect the fault under consideration. *This can also be considered as an example to support the idea*

*of trying both orientations of the generator when running simulations* (section 4.5.2, page 42).

The method suggested in [SR92] overcomes the shift dependency as indicated in the above example. The problem is solved by separating the inputs that belong to the same cone such that no two inputs belonging to the same cone are driven from contiguous SRLs. A brief description of their algorithm to separate the inputs is given below followed by a short example.

Step 1: The inputs to the CUT are ordered so that the ones feeding more outputs appear before those feeding fewer outputs.

Step 2: If there are inputs in the list, from Step 1, that feed all primary outputs (POs) of the CUT, then these inputs are assigned to alternative SRLs with dummy latches in between.

Step 3: The list of unassigned PIs is scanned from top to bottom. The next SRL is assigned to a PI that does not feed any of the outputs already fed by the previously assigned input. If such an input does not exist then Step 4 is executed, otherwise Step 5 is executed.

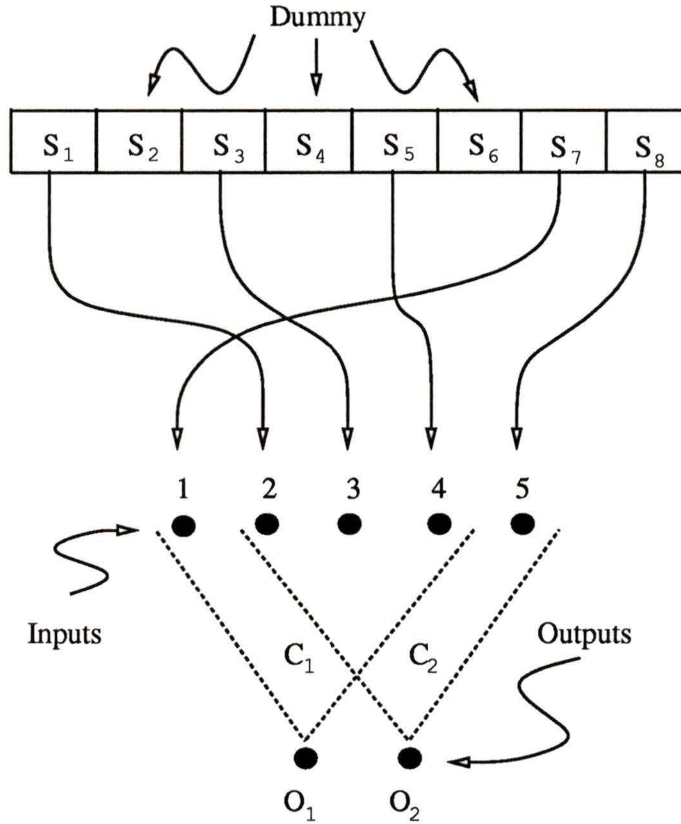
Step 4: The SRL under consideration is skipped (left as a dummy) and the next SRL is assigned to the input that appears at the top of the unassigned PIs list.

Step 5: The input is marked as assigned. If all PIs have not been assigned then Step 3 is repeated.

For example, consider the cones of influence in Figure 4.10 for some circuit with 5 inputs and two outputs. The circuit has 2 logic cones and each cone has four inputs. The result of executing Step 1 of the algorithm on this example circuit is as follows:

| | | | | | | |
|---|---|---|---|---|---|---|
| Input Number | : | 2 | 3 | 4 | 1 | 5 |
| Number of Outputs Affected | : | 2 | 2 | 2 | 1 | 1 |

**Figure 4.10** Input separation for logic cones.




Since the inputs 2, 3 and 4 feed all the outputs in the circuit, Step 2 of the algorithm assigns these inputs to the SRLs 1, 3 and 5 respectively. SRLs 2 and 4 are used as dummies to separate the inputs from being driven by contiguous SRLs. The remaining part of the algorithm introduces an additional dummy, SRL 6, and assigns PIs 1 and 5 to SRLs 7 and 8 respectively.

Now that all the inputs belonging to the same logic cone have been separated, each cone can potentially see all pairs of patterns. The results obtained by implementing this algorithm and applying on the 10 ISCAS '85 circuits appear in Table 4.3. Note that, for the reasons mentioned in section 4.5.2, both orientations of the generator (LFSR) were used. The results in Table 4.3 slightly differ from those given in [SR92] because no fault collapsing techniques were applied during simulations. Moreover, the implementation details were not available in [SR92]. Nevertheless, all the inputs belonging to the same

logic cones were separated using LFSR sizes corresponding to the sizes of SRL chain given in [SR92].

**Table 4.3** Transition fault simulation results for the ISCAS '85 circuits using ISUD method.

| | | FBf | | FBl | |
|---|---|---|---|---|---|
| Cct. | Faults$\xi$ | Und. | Cvg. (%) | Und. | Cvg. (%) |
| c432 | 864 | 10 | 98.84 | 10 | 98.84 |
| c499 | 998 | 8 | 99.20 | 8 | 99.20 |
| c880 | 1760 | 0 | 100 | 0 | 100 |
| c1355 | 2710 | 28 | 98.97 | 28 | 98.97 |
| c1908 | 3816 | 12 | 99.69 | 11 | 99.71 |
| c2670 | 5492 | 866 | 84.23 | 762 | 86.13 |
| c3540 | 7080 | 257 | 96.37 | 257 | 96.37 |
| c5315 | 10630 | 63 | 99.41 | 63 | 99.41 |
| c6288 | 12576 | 85 | 99.32 | 85 | 99.32 |
| c7522 | 15106 | 335 | 97.78 | 378 | 97.50 |

$\xi$ Fault collapsing techniques were not applied.

The obvious disadvantage of the above algorithm is that it requires additional hardware (dummies), although it is suggested in [SR92] that some of the dummy latches be replaced by the output SRLs when available.

## 4.6 Summary

In this chapter, the details and concerns of *delay fault* testing were described. The requirements and testing of transition fault testing were also given. The shift dependency problem with LFSRs, which is a hindrance to achieving higher transition fault coverage, was pointed out. Two methods (the XLFSR and the ISUD methods) which appeared in the literature, that attempt to overcome this problem were described. The transition fault coverage results for random connections between the generator outputs and the inputs of ISCAS '85 circuits were presented. The transition fault coverage results for the ISUD method were also given.

Motivated by the two methods described in this section (ISUD) and in section 4.5.1 (XLFSR) and also by the interesting results of the experiments mentioned in section 4.5.2, a new algorithm was developed to separate the inputs that belong to the same logic cone. The algorithm which is described in chapter 5, unlike the ISUD method, restricts the number of latches to be equal to the number of inputs in the circuit, so as to keep the hardware cost low. With this algorithm, it is possible (in some exceptional cases) to separate all the inputs using fewer latches than the number of inputs in the circuit. Very recently, it was pointed out in [PS92] that reasonably high *transition fault* coverage can be achieved by using the algorithm of [SR92] (mentioned earlier) but without using dummies. The high fault coverage results without using dummies were already seen in [ZBM92] (section 4.5.1) and in the experiments of section 4.5.2. The suggested method, in [PS92], iteratively applies the algorithm of [SR92] to *secondary outputs* at different levels of the CUT. The details of how the dummies are avoided were not available in [PS92].

# Chapter 5

# Algorithm To Identify Connections

Two methods were described in sections 4.5.1 and 4.5.3 that overcome the shift dependency problem (section 4.4.1) associated with a LFSR and SLOTT. Both methods improve the transition coverage at the *inputs* of the circuit under test (CUT). This improvement in turn enhances the transition *fault coverage* of the CUT. Although, these methods enhance the transition coverage, each has its own disadvantage.

It was pointed out, in section 4.5.2, that the cross over permutation may not work well for all circuits. The input separation method using dummies (ISUD) (section 4.5.3) increases the hardware cost when the logic cones in the CUT have a high ratio (greater than 0.5) of logic-cone-inputs to the total number of primary inputs (PIs), or if there is overlapping of the logic cones.

This chapter describes an algorithm that identifies the connections between the $n$ inputs of the CUT and shift register latches (SRLs) in the scan chain. In this chapter when a reference is made to SRLs, it also applies to the cells of the LFSR (with external XOR gate).

## 5.1 Requirements and Characteristics of the Algorithm

Before giving the algorithm in section 5.2, the requirements and characteristics of the algorithm are mentioned here. This section also gives some reasons to support the ideas used in the algorithm.

The surprising results of the experiments mentioned in section 4.5.2, where randomly generated connections (R-CXN) between the outputs of the LFSR and the inputs of the CUT were used, indicate that a reasonably high transition fault coverage can be achieved without using dummies. For comparison purposes, the best observed transition fault coverage from the experiments (R-CXN) and the transition fault coverage obtained using the ISUD method are given in Table 5.1.

**Table 5.1** Transition fault coverage for randomly generated connections (R-CXN) and the input separation method using dummies (ISUD).

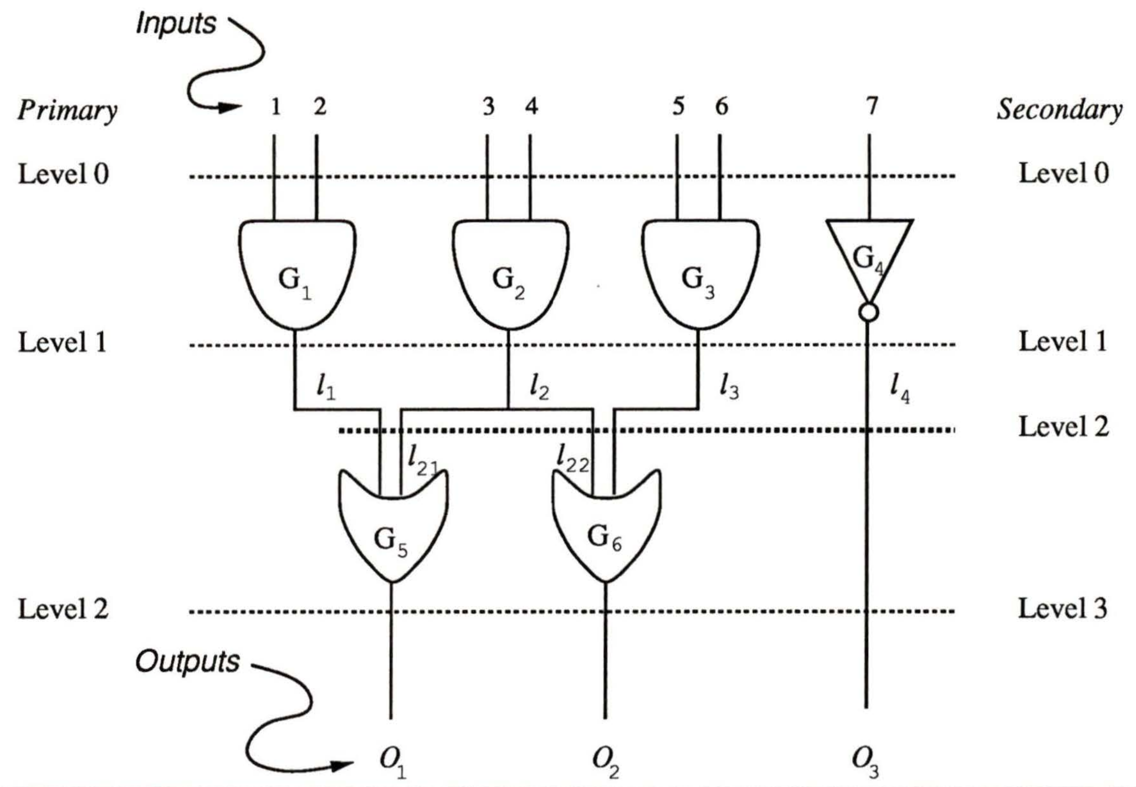
| Circuit name | Number of Faults$\xi$ | R-CXN | | ISUD | |
|---|---|---|---|---|---|
| | | Und.† | Cvg.‡(%) | Und. | Cvg. (%) |
| c432 | 864 | 10 | 98.84 | 10 | 98.84 |
| c499 | 998 | 8 | 99.20 | 8 | 99.20 |
| c880 | 1760 | 2 | 99.89 | 0 | 100* |
| c1355 | 2710 | 25 | 99.08* | 28 | 98.97 |
| c1908 | 3816 | 11 | 99.71 | 11 | 99.71 |
| c2670 | 5492 | 637 | 88.40* | 762 | 86.13 |
| c3540 | 7080 | 276 | 96.10 | 257 | 96.37* |
| c5315 | 10630 | 66 | 99.38 | 63 | 99.41* |
| c6288 | 12576 | 94 | 99.25 | 85 | 99.32* |
| c7522 | 15106 | 548 | 96.37 | 335 | 97.78* |

‡Fault Coverage. * Best observed result.

†Number of undetected faults.

$\xi$ Fault collapsing techniques were not applied.

**Figure 5.1** Levels of a circuit.




### 5.1.1 Information Required

Two sets of information are used by the algorithm:

1. The set of primary outputs (POs) affected by each primary input (PI). This set will be referred to as *POset*.

2. The level at which any two PIs "logically" meet. This is explained in detail below.

#### POset

Consider the circuit shown in Figure 5.1. It has 7 PIs and 3 POs. The POset for each PI is as follows:

| $PI$ | : | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| $POset$ | : | $\{O_1\}$ | $\{O_1\}$ | $\{O_1, O_2\}$ | $\{O_1, O_2\}$ | $\{O_2\}$ | $\{O_2\}$ | $\{O_3\}$ |

### Levels in a Circuit

The PI lines are considered to be at level 0. Gates $G_1, G_2, G_3$ and $G_4$ are considered to be one level higher (Level 1) than the level of their inputs (Level 0). The output of a gate is at the same level as the gate. Hence, lines $l_1, l_2, l_3$ and $l_4$ are at level 1.

The gate $G_2$ has a fanout of 2 ($> 1$) and therefore, the fan out lines, $l_{21}$ and $l_{22}$, can be considered to be either at the same level as $l_2$ or one level higher. Therefore, a circuit can be assigned levels in two different ways:

1. considering the fanout lines to be at the same level as their source; or

2. considering the fanout lines to be at one level higher than their source.

To differentiate between these two methods of leveling, the first one will be referred to as *Primary Leveling*, and the second one as *Secondary Leveling*.

The level of a gate is one level higher than the highest level of its input lines. For example, the gate $G_5$ has inputs $l_1$ and $l_{21}$. In secondary leveling, $l_1$ is at level 1 and $l_{21}$ is at level 2. Therefore, $G_5$ is at one level higher (Level 3) than the highest level (Level 2) of its input line, $l_{21}$.

Each line is also assigned a range of levels. Consider the line $l_4$, which is at level 1 in both leveling methods. Suppose, an arbitrary horizontal line is drawn across the circuit, after the gates at level 1, the line will always cross $l_4$. Thus, $l_4$ appears at all levels after the gates at level 1. The *maximum level* of a line is one less than the level of the gate that it feeds. If a line goes straight to the output, as in the case of $l_4$, its maximum level equals the maximum level of the circuit. For example, in Figure 5.1 the maximum *primary level* is 2 and the maximum *secondary level* is 3.

Two primary inputs are said to "logically" meet at a level '$l$', when paths starting from the two PIs pass through the same gate at level '$l$'. The "logical" meeting point for the PIs of the circuit in Figure 5.1 is represented in Table 5.2.

As an example, the entry in row 3 and column 2, which is 3, implies that ($PI_3$) "logically" meets ($PI_2$) at the *third* secondary level. A "0" entry implies that the two PIs do not meet. A "-" marks the undefined entries.

**Table 5.2** Secondary logic levels at which two inputs meet.

| INPUTS | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 1 | - | 1 | 3 | 3 | 0 | 0 | 0 |
| 2 | 1 | - | 3 | 3 | 0 | 0 | 0 |
| 3 | 3 | 3 | - | 1 | 3 | 3 | 0 |
| 4 | 3 | 3 | 1 | - | 3 | 3 | 0 |
| 5 | 0 | 0 | 3 | 3 | - | 1 | 0 |
| 6 | 0 | 0 | 3 | 3 | 1 | - | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | - |

- : undefined entry. 0 : lines do not meet.

### 5.1.2 Characteristics

The information, related with the CUT, which is required by the algorithm, was explained in section 5.1.1. This section mentions some characteristics of the algorithm.

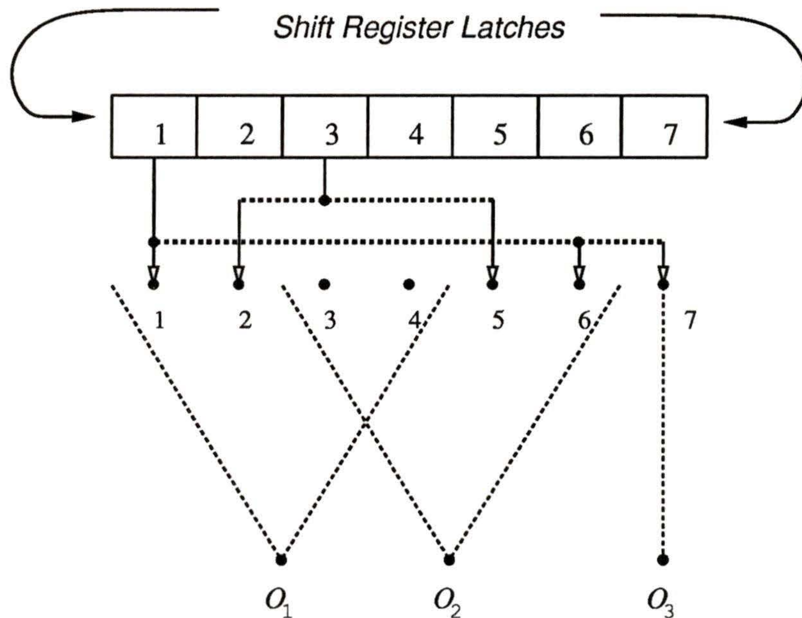
**Fixed length of the SRL chain**

Unlike the ISUD method ([SR92], section 4.5.3), where there is no upper limit to the length of SRLs in the chain, an upper limit to the length of the SRL chain is set at the beginning of the algorithm. Usually, this value is set to the number of PIs in the circuit.

**Single Latch Feeding Multiple Inputs**

The disadvantage of the algorithm in [SR92], where the insertion of dummies lengthens the SRL chain can, in some cases, be overcome by allowing a latch to feed multiple PIs. The algorithm to be mentioned in section 5.2 allows a latch to feed multiple PIs under the strict condition that the *PIs have mutually exclusive POset* (section 5.1.1). As an example, consider the logic cones for the circuit of Figure 5.1 shown in Figure 5.2. Two sets of PIs with mutually exclusive POsets are {1, 6, 7} and { 2, 5}. Hence, the inputs in each set can be fed from the same source (SRL) as shown in Figure 5.2. It should be noted that, the cost and complexity of routing increases as a result of the above mentioned approach. The other concern of this approach is the fanout limit of the SRLs.

**Figure 5.2** Logic cones of the sample circuit.




It is desirable that each SRL feed as many PIs as possible which satisfies the earlier mentioned condition. This is done to take advantage of the circuit structure and separate inputs that belong to the same cone. In certain cases, this approach requires fewer SRLs than the number of inputs in the CUT.

**Adjacent latches feeding the same logic cone**

In cases where:

1. more than half the number of inputs in the CUT (*i.e.* $k > \lceil n/2 \rceil$) feed the same output;

2. many inputs feed all outputs,

the assigning of adjacent latches to the inputs that belong to the same logic cone (collision [1]) becomes inevitable. Therefore, the latch to PI assignment is done in such a way that:

1. the intersection of the POsets of the concerned PIs is minimal;

[1]The term collision [SR92] is used to describe a situation where two adjacent latches are assigned to PIs that belong to the same cone.

2. the level at which the PIs "logically" meet is closest to the level of the POs.

The reason for the second condition, mentioned above, is that if two PIs, that belong to the same cone, are connected to adjacent SRLs, then the effect of shift dependency problem is significant when the paths starting from these inputs enter a gate at the first level. The shift property becomes less significant if these paths separately pass through a number of gates and "meet" at the gate at a later level.

### 5.1.3 Phases of the Algorithm

The algorithm has two phases which will be referred to as the *first phase* or the *alternate latch assignment* phase, and the *second phase* or the *selective latch assignment phase.*

#### 5.1.3.1 First Phase (Alternate Latch Assignment Phase)

The *alternate latch assignment* phase, as the name implies, assigns the PIs to alternate latches. This *optional* phase serves two purposes:

1. within the restricted length of the SRL chain, this phase separates as many inputs as possible that belong to the same logic cones;

2. it assigns PIs with mutually exclusive POsets to the same latch. For example in Figure 5.2, the PIs, $PI_1$, $PI_6$ and $PI_7$, have mutually exclusive POsets and are therefore assigned to the same SRL, $SRL_1$.

This phase begins by assigning the first SRL to a PI, say $I_1$, in the list of the unassigned PIs. After doing so, a search is conducted for PIs that, together with the already assigned PI, $I_1$, have mutually exclusive POsets. If one or more PIs satisfying this condition are found, then they are assigned to the SRL under consideration.

After assigning some PIs to a SRL, say $SRL_i$, the adjacent SRL ($SRL_{i+1}$) is skipped. The process explained in the previous paragraph, for the first SRL, is then repeated for the unassigned SRL ($SRL_{i+2}$). This process is repeated until one of the following occurs:

1. the length of the SRL reaches its preset limit; or

2. all PIs have been assigned. In this case the number of SRLs used may be less than the preset limit.

The main point in the first phase is to assign more PIs to fewer SRLs and yet avoid collisions. Figure 5.2 shows an example of an intermediate stage during the first phase. A total of five PIs have been assigned to the first three SRLs, leaving four SRLs to easily separate the remaining 2 inputs ($PI_3$ and $PI_4$).

At the end of the *first phase*, if all PIs have not been assigned, then the *second phase* assigns the remaining PIs to the SRLs that are skipped in the *first phase*. In such cases, a collision is inevitable but the *second phase* attempts to minimize not only the collisions but also their effect in the logic cones.

#### 5.1.3.2 Second Phase (Selective Latch Assignment Phase)

The *selective latch assignment phase*, selects a "suitable" unassigned latch, in the SRL chain, for each of the unassigned PI. A "suitable" latch is one such that, when a PI is assigned to this SRL:

1. the assignment is *collision-free*; if this is not possible then,

2. a minimum number of logic cones are affected (*i.e.* the inputs that are connected to adjacent latches are common inputs to *minimum* number of cones.)

A major difference between the first phase and the second phase is that, in the second phase, only a single PI is assigned to a SRL.

Consider a situation during the process of assigning PIs to the latches as shown in Figure 5.3. This is a typical situation when the *optional* first phase of the algorithm is omitted. In Figure 5.3, the second phase has already assigned $PI_1$ to $SRL_5$, $PI_2$ to $SRL_7$, $PI_3$ to $SRL_1$ and $PI_4$ to $SRL_3$. At this point, all *assigned* PIs, which belong to the same cone, have been separated (*i.e.* are not fed by adjacent latches).

Suppose, we would like to assign $PI_5$ to a "suitable" unassigned SRL such that the above mentioned two conditions are satisfied. The unassigned SRLs are 2, 4 and 6.

**Figure 5.3** Assigning inputs to latches during the second phase.

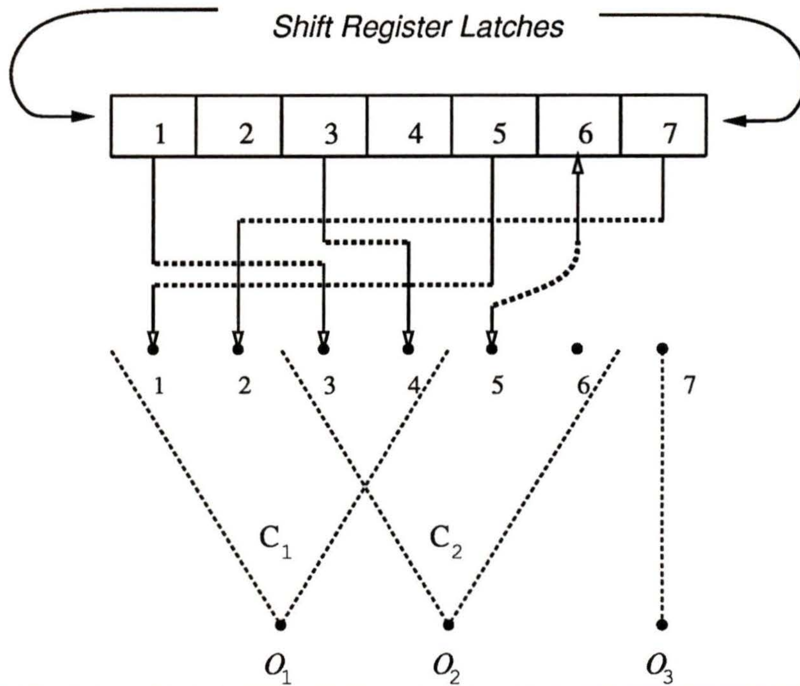



Table 5.3 shows the effect of *temporarily* assigning $PI_5$ to each of these unassigned SRLs. The five columns in the table contain the following information:

1. SRL - is the *unassigned latch* ($SRL_k$) to which the PI, $i$, is temporarily assigned and the effect of this temporary assignment is shown in the remaining columns of the table.

2. Adjacent Latches - When the PI, $i$, is temporarily connected to $SRL_k$, then it is important to see which PIs are connected to the 2 pairs of *adjacent latches* ($SRL_{k-1}$, $SRL_k$) and ($SRL_k$, $SRL_{k+1}$).

3. PIs-Concerned - These are the PIs fed by the adjacent latches ($SRL_{k-1}$, $SRL_k$) and ($SRL_k$, $SRL_{k+1}$).

4. Affected-Cones - When 2 or more PIs (PIs-Concerned), that are fed by adjacent SRLs, belong to the same logic cone(s), the *cone(s)* will suffer from shift dependency problem (section 4.4.1).

5. Total - The total number of cones that are affected (Affected-Cones) when the PI, $i$, is temporarily assigned to $SRL_k$. This total also includes the number of times the same cone is affected. A "suitable" SRL, $SRL_k$, is one such that when a PI, $i$, is assigned to this SRL , no logic cones are affected.

**Table 5.3** Effect of temporarily assigning input 5 to each of the unassigned latches.

| SRL | Adjacent-Latches | PIs-Concerned | Affected-Cones | Total |
|---|---|---|---|---|
| 2 | 1,2 | 3,5 | $C_2$ | |
| | 2,3 | 4,5 | $C_2$ | 2 |
| 4 | 3,4 | 4,5 | $C_2$ | |
| | 4,5 | 1,5 | *None* | 1 |
| 6 | 5,6 | 1,5 | *None* | |
| | 6,7 | 2,7 | *None* | 0 |

**Effect of temporarily assigning $PI_5$ to $SRL_2$**

If $PI_5$ is assigned to $SRL_2$, then the effect of the two pairs of adjacent latches $(SRL_1, SRL_2)$ and $(SRL_2, SRL_3)$ that feed the inputs of the circuit need to be looked at.

1. $(SRL_1, SRL_2)$: The first pair, $(SRL_1, SRL_2)$, feeds the inputs $PI_3$ and $PI_5$, respectively. Since, both PIs belong to the same logic cone, $C_2$, this is recorded. (See the the first row in Table 5.3 under the column *Affected-Cones.*)

2. $(SRL_2, SRL_3)$: The second pair $(SRL_2, SRL_3)$ feeds the inputs $PI_5$ and $PI_4$, respectively. These two PIs also belong to the same logic cone, $C_2$. Therefore, we see that if $PI_5$ is assigned to $SRL_2$, this results in two pairs of adjacent SRLs feeding the same logic cone $C_2$. This is indicated in the last column (Total) of Table 5.3.

Similarly, the effect of assigning $PI_5$ to $SRL_4$ (second row in Table 5.3) and also to $SRL_6$ (third row in able 5.3) are analyzed and recorded. In an ideal case, the total (last column in the table) number of cones affected is zero. This implies that no pair of adjacent latches feed inputs that belong to the same cone. Thus, $PI_5$ is assigned to $SRL_6$.

To give a better understanding of the second phase, the next step for the example in Figure 5.3 follows.

At this point, the unassigned latches are $SRL_2$ and $SRL_4$, and the unassigned inputs are $PI_6$ and $PI_7$. In order to select a suitable unassigned SRL for $PI_6$, the effect of assigning $PI_6$ to each of the unassigned SRL is done in the same way as explained above. The results are shown in Table 5.4.

**Table 5.4** Effect of temporarily assigning input 6 to each of the unassigned latches.

| SRL | Adjacent-Latches | PIs-Concerned | Affected-Cones | Total |
|---|---|---|---|---|
| 2 | 1,2 | 3,6 | $C_2$ | 2 |
| | 2,3 | 4,6 | $C_2$ | |
| 4 | 3,4 | 4,6 | $C_2$ | 1 |
| | 4,5 | 1,6 | *None* | |

The data in Table 5.4 indicates that there is no choice of SRL that will avoid a collision (*i.e.* no entry of SRL in Table 5.4 gives a 0 in the *Total* column). Hence, the choice of SRL that will affect the minimum number of cones is selected. For $PI_6$, the "suitable" choice of SRL is, therefore, $SRL_4$.

Suppose, there is more than one choice of SRL that results in a minimum number of cones being affected. The tie is broken by rejecting the SRL choices that feed PIs which "logically" meet at levels closer to level 0.

### 5.1.4 Summary of Requirements and Characteristics

In sections 5.1.1, 5.1.2 and 5.1.3, the requirements and characteristics of the algorithm, that will be mentioned in section 5.2, were given in detail. This section summarizes the last three sections and gives appropriate references for details.

The requirements and characteristics of the algorithm are:

1. Requires two sets of information related with the CUT. These two sets are (section 5.1.1):

   (a) The set of POs affected by each PI (also called the POset).
   (b) The level at which any two PIs "logically" meet.

2. Before executing the algorithm, an upper limit is assigned to the length of the SRL chain. This limit is usually equal to the number of PIs in the CUT.

3. In situations where the assignment of adjacent SRLs to PIs that belong to the same logic cone (also called a *collision)* is inevitable, the PI is assigned to a SRL using the following rules (section 5.1.3.2):
   (a) The PIs that are fed by the adjacent SRLs affect minimum number of POs (*i.e.*, the intersection of the POsets of the concerned PIs is minimal).
   (b) The PIs concerned, "logically' meet (section 5.1.1) at a level that is closer to the level of the POs.

4. The algorithm has two phases:
   (a) An optional *first phase* (section 5.1.3.1) or *alternate latch assignment* phase serves two purposes:
      i. within the restricted length of the SRL chain, this phase separates as many inputs, that belong to the same cone, as possible. The inputs are separated by assigning them to *alternate* latches.
      ii. where possible, the PIs that have mutually exclusive POsets are assigned to the same latch.
   (b) The *second phase* (section 5.1.3.2) or the *selective latch assignment* phase selects a "suitable" unassigned SRL in the chain for each unassigned PI.

## 5.2 The Algorithm

This section gives the algorithm that dictates the connections between the PIs of the CUT and a fixed number of SRLs (usually equal to the number of PIs). The connections are made with the intent of avoiding collision. If collisions are inevitable, then an attempt is

made to make the connections in such a way that the collisions affect a minimum number of logic cones. Also, the *effect* of collisions is kept at a *minimum*. These connections enhance the transition coverage at the inputs of the logic cones. This in turn enhances the transition fault coverage of the CUT.

### 5.2.1 Steps for executing the Algorithm

The various requirements and characteristics of the algorithm were mentioned in section 5.1 and summarized in section 5.1.4. As mentioned in section 5.1, the algorithm uses the following information about the CUT:

1. The set of POs affected by each PI (POset).
2. The level at which any two PI "logically" meet.

The two phases of the algorithm, of which the first one is optional, were introduced in the earlier sections. Both phases are preceded by an *optional* step of sorting the PIs. The reason for ordering the inputs and also the different ways of ordering are mentioned in section 5.2.1.1.

The *first phase* is subdivided into 5 steps and these are mentioned in section 5.2.1.2. If the *first phase* is skipped, then the *second phase* must be executed. In some exceptional cases, the *second phase* is not required as all PIs get assigned in the *first phase* of the algorithm. The *second phase* is subdivided into 4 steps and these are mentioned in section 5.2.1.3.

The second phase (*selective latch assignment phase*) can either be executed independently or preceded by the optional first phase (*alternate latch assignment phase*). When both phases are executed, the method will be referred to as **ISASLA** (Input Separation by Alternate and Selective Latch Assignment) method. When the second phase is executed independently, the method will be referred to as **ISSLA** (Input Separation by Selective Latch Assignment) method.

#### 5.2.1.1 Sorting the Primary Inputs

As mentioned earlier, each phase of the algorithm is preceded by an optional step of sorting the PIs. Each PI feeds a certain number of POs, which is represented as a set, the *POset*. The PIs can be sorted either in increasing or decreasing order of the size of their POsets.

If the sorting of the PIs is done in increasing order, the PI with the smallest POset (*i.e.* the PI that affects the least number of POs) appears at the top of the list and the PI with the largest POset (*i.e.* the PI that affects the largest number of POs) appears at the bottom of the list.

For a given circuit with $n$ inputs and a given SRL chain of equal size, there are $n!$ ways to assign the inputs of the circuit to the latches. Moreover, if a SRL is allowed to feed more than one PI, the number of ways to assign inputs to latches gets larger than $n!$. For this reason it is difficult to keep track of how a decision, regarding the assignment of a PI to a SRL, made at an earlier stage of the algorithm may affect a few stages later. Thus, the algorithm is a greedy one in the sense that the decisions, regarding the assigning of a PI to a SRL, are made based on the prevailing conditions without any considerations for its effect a few stages later. Therefore, the ordering of primary inputs (PIs) may play a significant role in the way the PIs are assigned to the SRLs.

Three different ways of ordering the PIs are:

1. Sorting PIs in increasing order of the size of their POset.

2. Sorting PIs in decreasing order of the size of their POset.

3. Leaving the PIs as they are (*i.e.* not sorted).

Depending on the structure of the CUT, the ordering of PIs affects the way the algorithm generates the connections between the PIs and the SRLs. These connections affect the transition coverage at the inputs of the logic cones of the CUT. The transition coverage in turn affects the *transition fault coverage* of the CUT. Therefore, the *ordering of PIs*, indirectly, may affect the transition fault coverage of the CUT. This will be shown later in chapter 6.

#### 5.2.1.2 Steps for the First Phase

The *first phase* of the algorithm is an optional one where the PIs with intersecting POsets are assigned to alternate latches. Also, where possible, the PIs that have mutually exclusive POsets are assigned to a single latch.

Phase one is subdivided into the following steps:

Step 1.1 The PI at the top of the list of unassigned PIs (PIlist) is assigned to the first latch in the SRL chain.

Step 1.2 After assigning a PI, $i$, to a SRL ($SRL_l$), a search is conducted in the PIlist for PIs whose POset is disjoint with that of the already assigned PI, $i$. If no such PI is found then Step 1.5 is executed. If a single PI is found then step 1.4 is executed. If more than one PI is found, that gives a new list of PIs, $L$, then Step 1.3 is executed.

Step 1.3 The POset of each PI in $L$ (from Step 1.2) is disjoint with that of PI $i$. But, the PIs collectively in $L$ may not have mutually exclusive POsets. Hence, $L$ needs to be trimmed and this can be done in two ways:

(a) scanning $L$ from top to bottom and creating a new list of PIs that collectively have a mutually exclusive POset (a greedy algorithm); or

(b) searching $L$ for the largest number of PIs that collectively have a mutually exclusive POset.

The PIs in the trimmed list are all assigned to the same latch as PI $i$ and Step 1.5 is then executed.

Step 1.4 The single PI obtained from the search in Step 1.2 is assigned to the same latch as PI $i$.

Step 1.5 If all PIs have not been assigned and all alternate latches in the given scan chain have also not been assigned, then the following is done.

The SRL ($SRL_{l+1}$), adjacent to the last assigned SRL ($SRL_l$) is skipped and the alternate unassigned SRL ($SRL_{l+2}$) is selected. The unassigned PI at the top of the

PIlist is assigned to the newly selected $SRL_{l+2}$ and Step 1.2 is repeated.

Given a graph, $G = (V, E)$, with vertices (V) representing PIs. When two PIs have *disjoint* POsets, their respective vertices (V) are connected by an edge (E). The search in Step 1.3 is for a *clique*[2] with the maximum number of vertices. This search is NP-complete [GJ79, page 194]. When $L$, in Step 1.3 is long, the search can be costly. During the experiments with this algorithm on ISCAS '85 circuits, the lengthy search was avoided whenever $L$ from Step 1.2 was longer than a preset limit.

The *first phase* ends, when either:

1. all PIs have been assigned; or

2. *all alternate* latches within the restricted length of the SRL chain have been assigned.

If all PIs are assigned in the *first phase*, then in some exceptional cases, it is possible that the number of SRLs used would be less than the number of inputs in the CUT.

If all the PIs are not assigned in the *first phase*, then the *second phase* assigns the remaining PIs to the unassigned SRLs between the alternately assigned SRLs.

#### 5.2.1.3 Steps for the Second Phase

The *second phase* of the algorithm is executed in one of the following situations:

1. when the *first phase* has been executed but *all* the PIs are not assigned to the alternate latches within the restricted length of the SRL chain (the *ISASLA* method).

2. when the *first phase* is omitted (the *ISSLA* method).

[2] A *clique* is a sub-graph which is complete. A complete sub-graph is one in which every pair of vertices is connected by an edge.

The steps of the *second phase* are as follows:

Step 2.1 The PI, $i$, at the top of the unassigned PIs list (PIlist) is picked.

Step 2.2 A "suitable" unassigned SRL is selected for the PI $i$ by:

Step 2.2.1 checking the effect of temporarily assigning $i$ to each of the unassigned SRL; and

Step 2.2.2 selecting a "suitable" SRL for $i$ by using the information obtained in Step 2.2.1.

A "suitable" SRL is one such that when the PI $i$ is assigned to this SRL:

1. the assignment is *collision-free*; if a collision is inevitable, then,
2. minimum number of cones are affected.

Step 2.3 If all the PIs have not been assigned, then Step 2.1 is repeated.

The details of the *second phase* are given, together with an example, in section 5.1.3.2, page 57.

## 5.3 Examples

The following sections give some examples where the *second phase* and also both phases of the algorithm, that were described in section 5.2, are applied to obtain connections between the SRLs and the PIs of a CUT.

Section 5.3.1 gives an example where only the *second phase* of the algorithm is applied (the *ISSLA* method) . Using the *ISASLA* method, instead of the *ISSLA* method, can make a difference. This is shown with an example in section 5.3.2.

When the *ISSLA* method is applied to circuits:

1. that have a single output; or,
2. that have logic cones with with inputs exceeding half the number of inputs in the circuit; or
3. where each input of the circuit affects (or feeds) all outputs,

then, the decision regarding the assigning of an input (PI) to a latch (SRL) in the chain is based on the levels at which two PIs "logically" meet. Section 5.3.3 gives an example where the *ISSLA* method is applied to a circuit with single output. An alternative method is suggested in section 5.3.3 which will be referred to as *ISSIA* method (Input Separation by Selective Input Assignment), and an example is given by applying the suggested *ISSIA* method to the circuit with single output.

### 5.3.1 Using Phase Two only (The *ISSLA* method)

This section shows how the *second phase* of the algorithm works without executing the first phase. The circuit in Figure 5.4 has four inputs (PIs) and four outputs (POs). The four logic cones, one for each output, are shown in Figure 5.5.

The two sets of information that are required by the algorithm are:

1. the set of outputs (POset) fed by each input (PI). These are as follows:

$$\begin{array}{llcccc} PI & : & 1 & 2 & 3 & 4 \\ POset & : & \{O_1, O_2, O_3, O_4\} & \{O_2, O_3, O_4\} & \{O_3, O_4\} & \{O_4\} \end{array}$$

2. the logic levels (*Primary levels*) at which any two PIs in Figure 5.4 'logically" meet. This information is given in Table 5.5.

**Table 5.5** Logic levels at which two inputs "logically" meet in binary to excess 3 code circuit.

| INPUTS | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | - | 1 | 3 | 4 |
| 2 | 1 | - | 3 | 4 |
| 3 | 3 | 3 | - | 4 |
| 4 | 4 | 4 | 4 | - |

- : undefined entry.

Since there are 4 inputs in the CUT, the steps 2.1 and 2.2 of the *second phase* (section 5.2.1.3, page 65) will be repeated for each input. Since all the SRLs are unassigned at the

**Figure 5.4** Binary (0-9) to excess 3 code circuit.

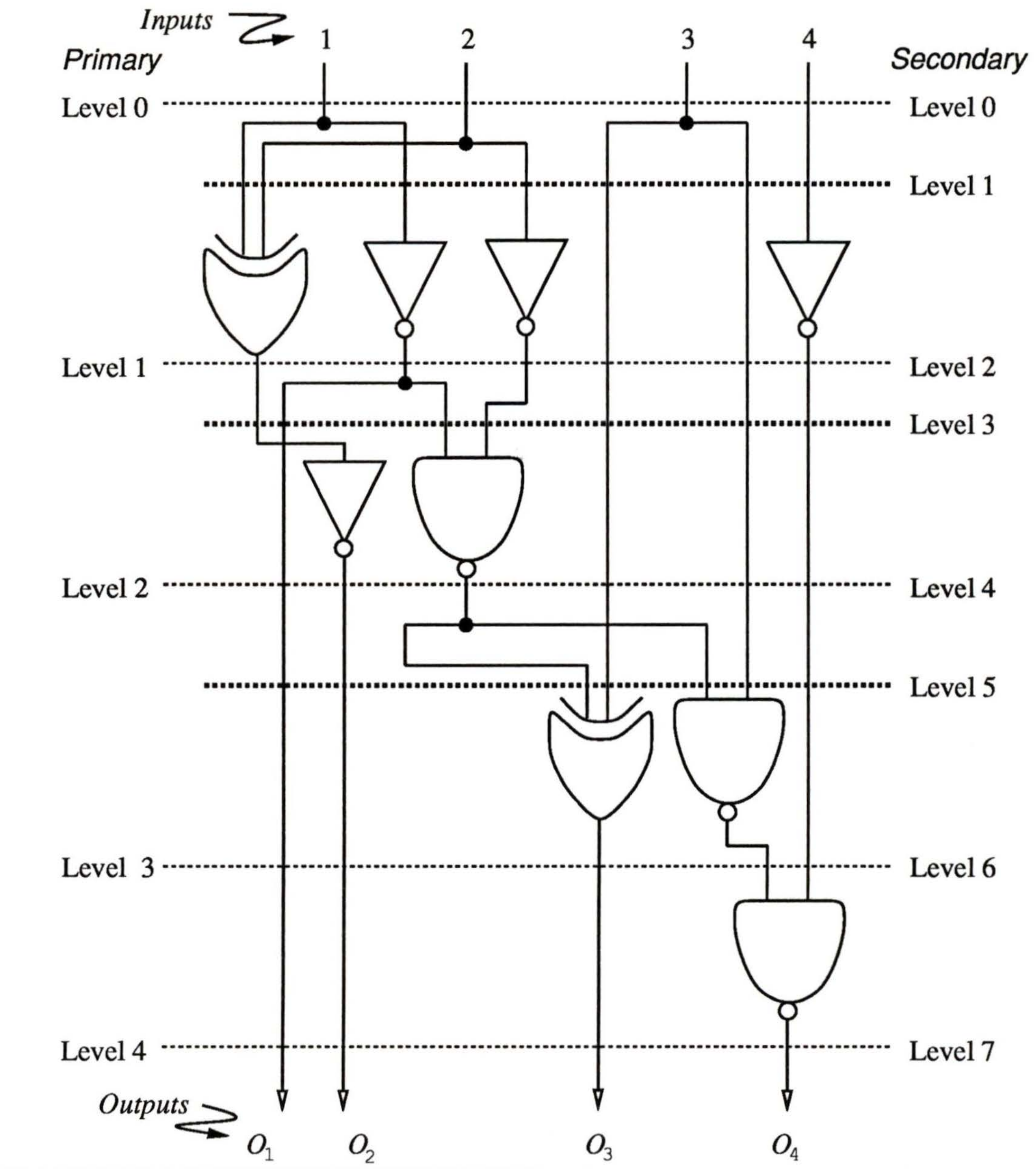
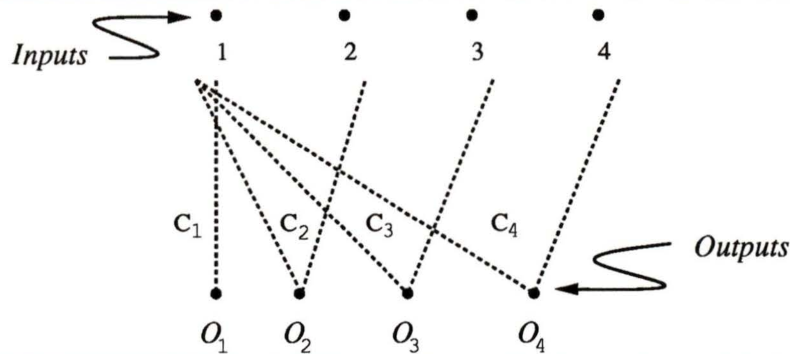


**Figure 5.5** Logic cones for binary (0-9) to excess 3 code circuit.




beginning, steps 2.1 and 2.2 assign $PI_1$ to the first SRL, $SRL_1$ (see Figure 5.6 (a), page 71).

In the *second loop* of the *second phase*, the following takes place:

Step 2.1 $PI_2$, which is at the top of the list of unassigned inputs (PIlist), is picked.

Step 2.2 At this stage, the unassigned SRLs are $SRL_2$, $SRL_3$, and $SRL_4$.

Step 2.2.1 The effect of temporarily assigning $PI_2$ to each of these SRLs is determined and is shown in Table 5.6 (For details about the table, see page 58 in section 5.1.3.2.

Step 2.2.2 The *Total* column indicates that there are two "suitable" choices of SRLs, $SRL_3$ and $SRL_4$. Therefore, $PI_2$ can be assigned to any of these two SRLs. To break a tie, the first one ($SRL_3$) is chosen (see Figure 5.6 (a), page 71).

After assigning $PI_2$ to $SRL_3$, steps 2.1 and 2.2 are repeated for $PI_3$ as follows:

Step 2.1 $PI_3$, which is at the top of the PIlist, is picked.

Step 2.2 At this stage, the unassigned SRLs are $SRL_2$, and $SRL_4$.

Step 2.2.1 The effect of temporarily assigning $PI_3$ to each of these two SRLs is determined and is shown in Table 5.7.

Step 2.2.2 The *Total* column indicates that the "suitable" SRL is $SRL_4$. Therefore, $PI_3$ is assigned to $SRL_4$ (see Figure 5.6 (a), page 71).

**Table 5.6** Effect of temporarily assigning $PI_2$ to each of the unassigned latches.

| SRL | Adjacent-Latches | PIs-Concerned | Affected-Cones | Total |
|---|---|---|---|---|
| 2 | 1,2 | 1,2 | $C_2, C_3, C_4$ | |
| | 2,3 | 2 | None | 3 |
| 3 | 2,3 | 2 | None | |
| | 3,4 | 2 | None | 0 |
| 4 | 3,4 | 2 | None | 0 |

**Table 5.7** Effect of temporarily assigning $PI_3$ to each of the unassigned latches.

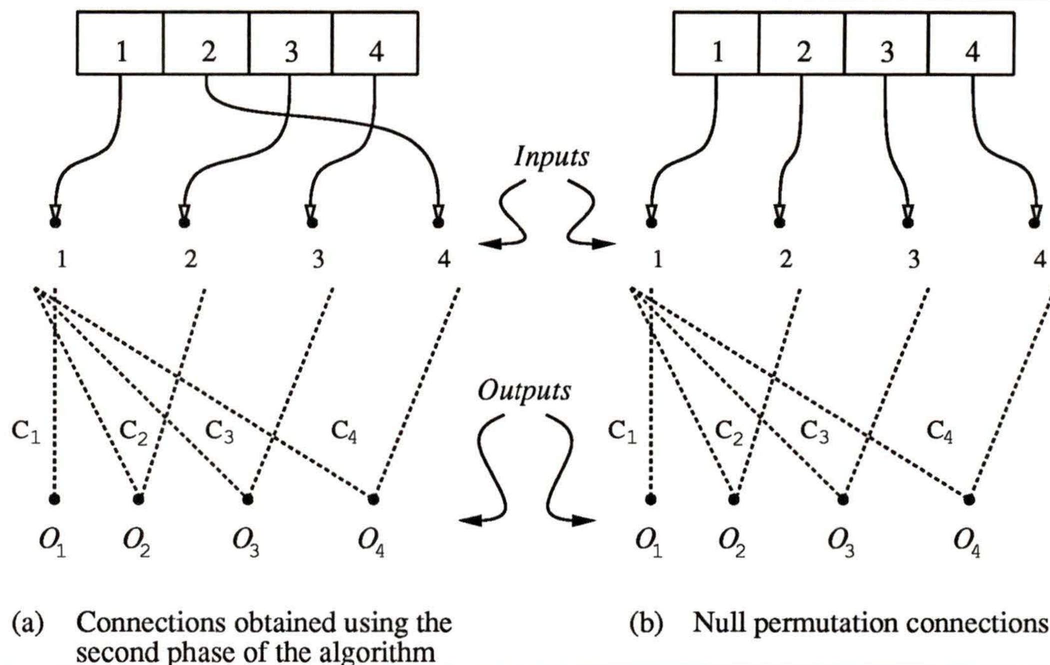
| SRL | Adjacent-Latches | PIs-Concerned | Affected-Cones | Total |
|---|---|---|---|---|
| 2 | 1,2 | 1,3 | $C_3, C_4$ | |
| | 2,3 | 2,3 | $C_3, C_4$ | 4 |
| 4 | 3,4 | 2,4 | $C_4$ | 1 |

After assigning $PI_3$ to $SRL_4$, the last unassigned input, $PI_4$, is assigned to the remaining SRL, $SRL_2$. Figure 5.6 (a) shows the connections between the SRLs and the PIs of the circuit at the end of the second phase.

Table 5.8, shows the effect resulting from the connections between the SRLs and the inputs, obtained by executing the second phase of the algorithm. The table includes a column, *Level*, for the earliest level at which two PIs, that are connected to adjacent latches, "logically" meet. For example, in the first row, the adjacent latches $SRL_1$ and $SRL_2$ are connected to inputs $PI_1$ and $PI_4$, respectively. From Table 5.5 (page 67), it can be seen that $PI_1$ and $PI_4$ "logically" meet at level 4. The *higher* the level, the *better*, because it is expected that the shift dependency problem may not be significant at *higher* levels. This can be considered to be similar to *low* controllability[3] of ilines deep in the circuit.

[3]Controllability is the ease with which a line within a circuit can be set to a binary logic value by applying the appropriate signals (binary values) at the inputs of the circuit.

**Figure 5.6** Connections between the SRLs and the circuit inputs.




(a) Connections obtained using the second phase of the algorithm

(b) Null permutation connections

It is worth comparing the connections obtained using the second phase, as described above, and the *null* permutation connections (*i.e.* $SRL_k$ connected to $PI_k$, for $k = 1$ to 4. See Figure 5.6 (b)). Therefore, Table 5.9, which is similar to Table 5.8, shows the effect resulting from using the *null* permutation connections.

The disadvantage of *null permutation*, for the given example, can be seen by comparing the *Total* and *Level* columns of the Tables 5.8 and 5.9. The total number of cones affected (sum of items in *Total* column) is more in Table 5.9. Moreover, in Table 5.9, $PI_1$ and

**Table 5.8** Resulting effect of assigning circuit inputs to SRLs using the second phase.

| Adjacent-Latches | PIs-Concerned | Level | Affected-Cones | Total |
|---|---|---|---|---|
| 1,2 | 1,4 | 4 | $C_4$ | 1 |
| 2,3 | 2,4 | 4 | $C_4$ | 1 |
| 3,4 | 2,3 | 3 | $C_3, C_4$ | 2 |

**Table 5.9** Resulting effect of assigning circuit inputs to SRLs using null permutation connections.

| Adjacent-Latches | PIs-Concerned | Level | Affected-Cones | Total |
|---|---|---|---|---|
| 1,2 | 1,2 | 1 | $C_2, C_3, C_4$ | 3 |
| 2,3 | 2,3 | 3 | $C_3, C_4$ | 2 |
| 3,4 | 3,4 | 4 | $C_4$ | 1 |

$PI_2$, which are connected to adjacent latches, "logically" meet at Level 1 that is close to the level of the inputs (Level 0). Hence the effect of shift dependency, on the transition coverage for *null* permutation will be more significant. This in turn may result in a lower transition fault coverage.

### 5.3.2 Comparing Single Phase (*ISSLA*) with both Phases (*ISASLA*)

Depending on the structure of the CUT, the use of single phase (the second phase) and the use of both phases of the algorithm may or may not give the same connections between the SRLs and the inputs of the circuit. The example in this section shows the difference that can occur when these two approaches are used.

The circuit in Figure 5.7 has five inputs and four outputs. The four logic cones for this example circuit are shown in Figure 5.8. One point to note in this example is that, the inputs to the logic cones are not all adjacent inputs of the CUT. The POset for each input (PI) is as follows:

$$\begin{array}{lcccccc} PI & : & 1 & 2 & 3 & 4 & 5 \\ POset & : & \{O_1, O_2, O_3\} & \{O_1, O_2, O_3\} & \{O_1, O_4\} & \{O_2, O_3\} & \{O_1, O_4\} \end{array}$$

The levels at which two PIs "logically" meet are given in Table 5.10.

#### 5.3.2.1 Effect of a Single Phase (the *ISSLA* method)

When the PIs are sorted in decreasing order of their POset size, the following order is obtained, $P_1, P_2, P_3, P_4, P_5$.

**Figure 5.7** Example circuit.

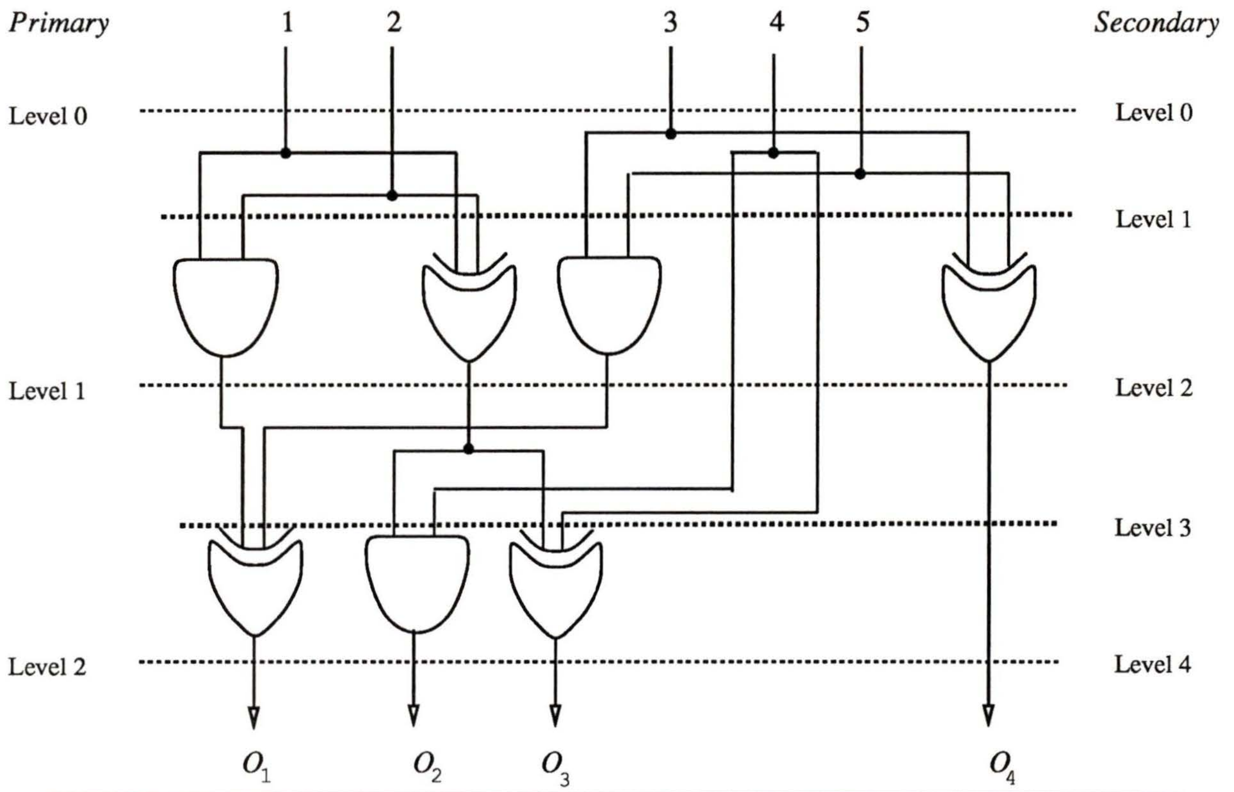



**Figure 5.8** Logic cones for the example circuit.

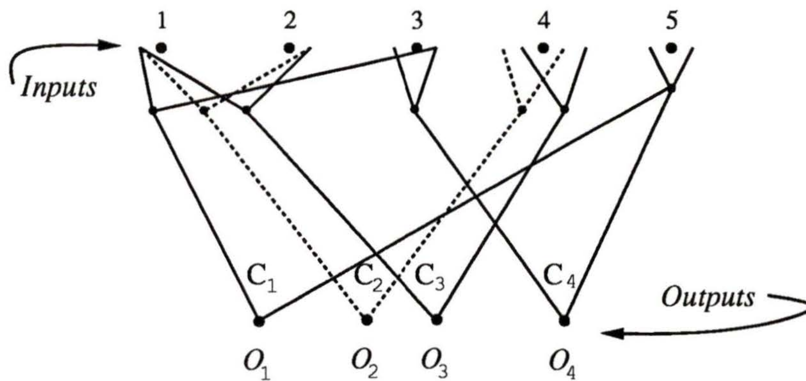


**Table 5.10** Logic levels at which two inputs meet in the example circuit.

| INPUTS | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | - | 1 | 2 | 2 | 2 |
| 2 | 1 | - | 2 | 2 | 2 |
| 3 | 2 | 2 | - | 0 | 1 |
| 4 | 2 | 2 | 0 | - | 0 |
| 5 | 2 | 2 | 1 | 0 | - |

- : undefined entry. 0 : lines not meet.

The second phase of the algorithm begins by assigning $PI_1$ to the first SRL, $SRL_1$ (Figure 5.9). In the following two loops (consisting of Steps 2.1 and 2.2) of the *second phase*, $PI_2$ and $PI_3$ are assigned to $SRL_3$ and $SRL_5$ respectively.

In the fourth loop of the *second phase*, the following takes place:

Step 2.1 $PI_4$, which is at the top of the list of unassigned inputs (PIlist), is picked.

Step 2.2 At this stage, the unassigned SRLs are $SRL_2$ and $SRL_4$.

Step 2.2.1 The effect of temporarily assigning $PI_4$ to each of these SRLs is determined and is shown in Table 5.11.

Step 2.2.2 The *Total* column indicates that the "suitable" SRL is $SRL_4$. Therefore, $PI_4$ is assigned to $SRL_4$ (see Figure 5.9, page 76).

In the fifth loop of the second phase, the last unassigned input, $PI_5$ is assigned to the remaining latch, $SRL_2$. The connections obtained using the second phase of the algorithm are shown in Figure 5.9. The effect resulting from such connections between the SRLs and the PIs is shown in Table 5.12

#### 5.3.2.2 Effect of both Phases (the *ISASLA* method)

The PIs are first sorted in decreasing order of their POset to obtain the following list (PIlist): $PI_1$, $PI_2$, $PI_3$, $PI_4$, $PI_5$. The *first phase* of the algorithm (section 5.2.1.2)

**Table 5.11** Effect of temporarily assigning $PI_4$ to each of the unassigned latches.

| SRL | Adjacent-Latches | PIs-Concerned | Affected-Cones | Total |
|---|---|---|---|---|
| 2 | 1,2 | 1,4 | $C_1, C_3$ | |
| | 2,3 | 2,4 | $C_1, C_3$ | 4 |
| 4 | 3,4 | 2,4 | $C_1, C_3$ | |
| | 4,5 | 3,4 | *None* | 2 |

**Table 5.12** Resulting effect of assigning circuit inputs to SRLs using the second phase.

| Adjacent-Latches | PIs-Concerned | Level | Affected-Cones | Total |
|---|---|---|---|---|
| 1,2 | 1,5 | 2 | $C_2$ | 1 |
| 2,3 | 2,5 | 2 | $C_2$ | 1 |
| 3,4 | 2,4 | 2 | $C_1, C_3$ | 2 |
| 4,5 | 3,4 | 0 | None | 0 |

Level 0 implies that the PIs do not meet.

assigns the unassigned input, $PI_1$, at the top of the sorted list, to the first latch, $SRL_1$ (Figure 5.10, page 79).

In the *first loop*, of the *first phase* (section 5.2.1.2), the following takes place:

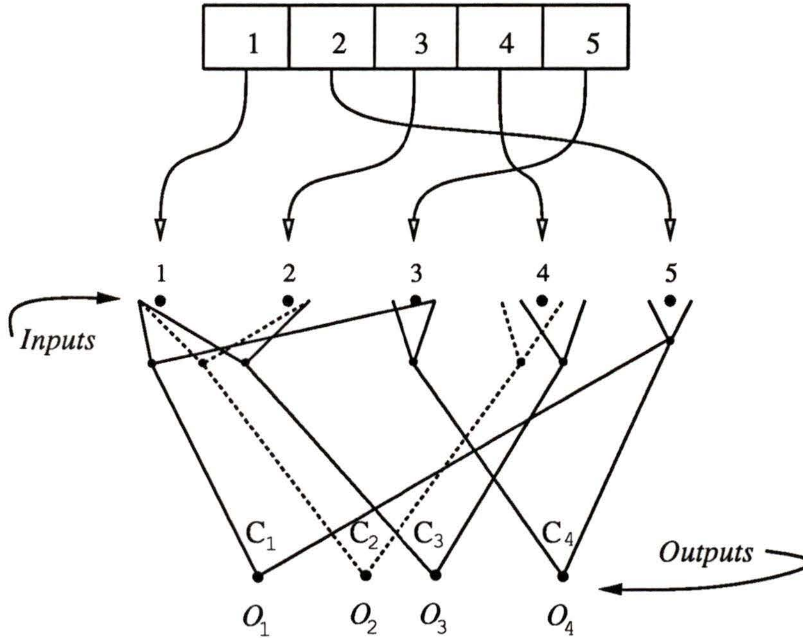
Step 1.2 A search is conducted in the PIlist for PIs whose POset is disjoint with that of $PI_1$ (*i.e.* the already assigned PI).

The search fails to find any PIs, that satisfy the mentioned condition, and therefore Step 1.5 is executed.

Step 1.5 The adjacent latch ($SRL_2$) is skipped and the unassigned input at the top of the PIlist (which currently is $PI_2$) is then assigned to $SRL_3$. The $SRL_2$ is skipped to avoid adjacent latches, $SRL_1$ and $SRL_2$, from feeding $PI_1$ and $PI_2$ whose POsets are not disjoint. Step 1.2 is then repeated and is explained below.

In the *second loop*, the following takes place:

**Figure 5.9** Connections between the SRLs and the circuit inputs obtained by using the second phase.




Step 1.2 A search is conducted in the PIlist for PIs whose POset is disjoint with that of the last assigned PI, $PI_2$.

The search, as in the previous loop, fails again and therefore Step 1.5 is executed.

Step 1.5 The adjacent latch ($SRL_4$) is skipped and the unassigned input at the top of the PIlist, which now is $PI_3$, is assigned to $SRL_5$ and Step 1.2 is repeated.

In the *third loop*, the following takes place:

Step 1.2 A search is conducted in the PIlist for PIs whose POset is disjoint with that of the last assigned PI, $PI_3$.

This time the search is successful and $PI_4$ is found whose POset is disjoint with that of the last assigned PI, $PI_3$. Since, only a single PI is found, therefore Step 1.4 is executed.

Step 1.4 The single PI, $PI_4$, is assigned to the same latch as $PI_3$, *i.e.* $SRL_5$ (see Figure 5.10.)

Step 1.5 Although all PIs have not been assigned but all alternate SRLs ( $SRL_1$, $SRL_3$ and $SRL_5$) have been assigned and therefore the *first phase* ends.

Note that all *assigned* inputs that belong to the same logic cones have been *separated* (*i.e.* they are not connected to adjacent latches.)

Since all PIs have not been assigned, the second phase must be executed to assign the remaining PIs to the unassigned SRLs. At this point the SRLs and the PIs that have been assigned to these SRLs are:

| $SRL$ | : | $SRL_1$ | $SRL_2$ | $SRL_3$ | $SRL_4$ | $SRL_5$ |
|---|---|---|---|---|---|---|
| $INPUT$ | : | $PI_1$ | *unassigned* | $PI_2$ | *unassigned* | $PI_3, PI_4$ |

**Second Phase**

Since all the PIs could not be assigned in the *first* phase, the *second* phase assigns the remaining PI, $PI_5$, as follows:

Step 2.1 $PI_5$, which is at the top of the list of unassigned inputs (PIlist), is picked.

Step 2.2 At this stage, the unassigned SRLs are $SRL_2$ and $SRL_4$.

Step 2.2.1 The effect of temporarily assigning $PI_5$ to each of these SRLs is determined and is shown in Table 5.13.

Step 2.2.2 The *Total* column indicates that the "suitable" choice of SRL is $SRL_2$, since the minimum number of cones are affected (*Total* $= 2$ when $PI_5$ is assigned to $SRL_2$).

The effect of the connections obtained by executing both phases of the algorithm are shown in Table 5.14. By comparing Tables 5.12 (page 75) and 5.14, it can be seen that the connections between the SRLs and the PIs obtained by using both phases (table 5.14) of the algorithm affect fewer logic cones than the connections obtained using the second phase only (table 5.12), although, this may not always be true. The fewer the number of logic cones that are fed by adjacent latches, the better will be the transition coverage at the inputs of the logic cones. This improved transition coverage will enhance the transition

**Table 5.13** Effect of temporarily assigning $PI_5$ to each of the unassigned latches.

| SRL | Adjacent-Latches | PIs-Concerned | Affected-Cones | Total |
|---|---|---|---|---|
| 2 | 1,2 | 1,5 | $C_2$ | |
| | 2,3 | 2,5 | $C_2$ | 2 |
| 4 | 3,4 | 2,5 | $C_2$ | |
| | 4,5 | 3,4,5 | $C_2, C_4$ | 3 |

**Table 5.14** Resulting effect of assigning circuit inputs to SRLs using both phases of the algorithm.

| Adjacent-Latches | PIs-Concerned | Level | Affected-Cones | Total |
|---|---|---|---|---|
| 1,2 | 1,5 | 2 | $C_2$ | 1 |
| 2,3 | 2,5 | 2 | $C_2$ | 1 |
| 3,4 | 2 | - | *None* | 0 |
| 4,5 | 3,4 | 0 | *None* | 0 |

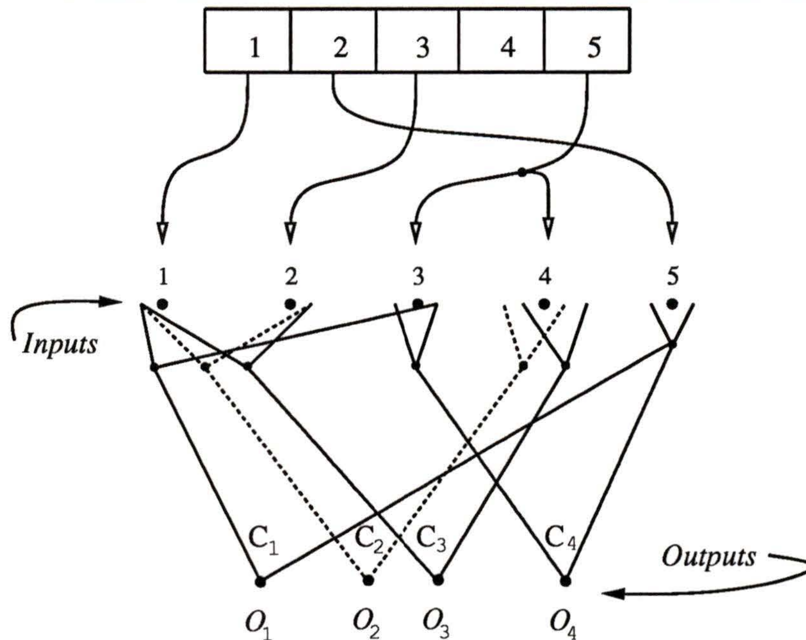
Level 0 implies that the PIs do not meet.

fault coverage. Hence, the above example indicates the advantage of allowing multiple inputs to be assigned to a single latch.

### 5.3.3 Logic with Single Output

The main idea in the second phase of the algorithm is to select a suitable SRL for a given input from the list of unassigned inputs. In this section the second phase of the algorithm is applied to the circuit in Figure 5.11, and the effect of the resulting connections between the SRLs and the inputs is shown. Later, a different approach will be suggested. The suggested approach, which is a simple modification to the ISUD method (section 4.5.3 and [SR92]), selects a suitable PI and assigns it to a given SRL. This method (*ISSIA*, Input Separation by Selective Input Assignment) is applied to the circuit in Figure 5.11 and its effects are also shown.

**Figure 5.10** SRLs to circuit inputs connections obtained using both phases.




The circuit in Figure 5.11 is an example of many inputs feeding a single output. In such cases, if the second phase of the algorithm is applied without applying the first phase, then the logic-level information of the circuit is used to decide which inputs should be separated (*i.e.* not assigned to adjacent SRLs). The atempt is to *minimize the effect* of shift dependency problem.

The circuit in Figure 5.11 has five inputs and a single output, $O_1$. Each input has the same $POset$ = $\{O_1\}$. The level at which any two PIs "logically" meet is given in Table 5.15.

#### 5.3.3.1 Using Phase Two only

Since every input, in Figure 5.11, has the same $POset = \{O_1\}$, the PIs are ordered as $PI_1$, $PI_2$, $PI_3$, $PI_4$, $PI_5$. The first three loops of the second phase assign $PI_1$ to $SRL_1$, $PI_2$ to $SRL_3$, and $PI_3$ to $SRL_5$. In fact, the first phase would also give the same result and leave the remaining inputs to be assigned by the second phase. This is summarized below:

**Figure 5.11** Two of five checker circuit.

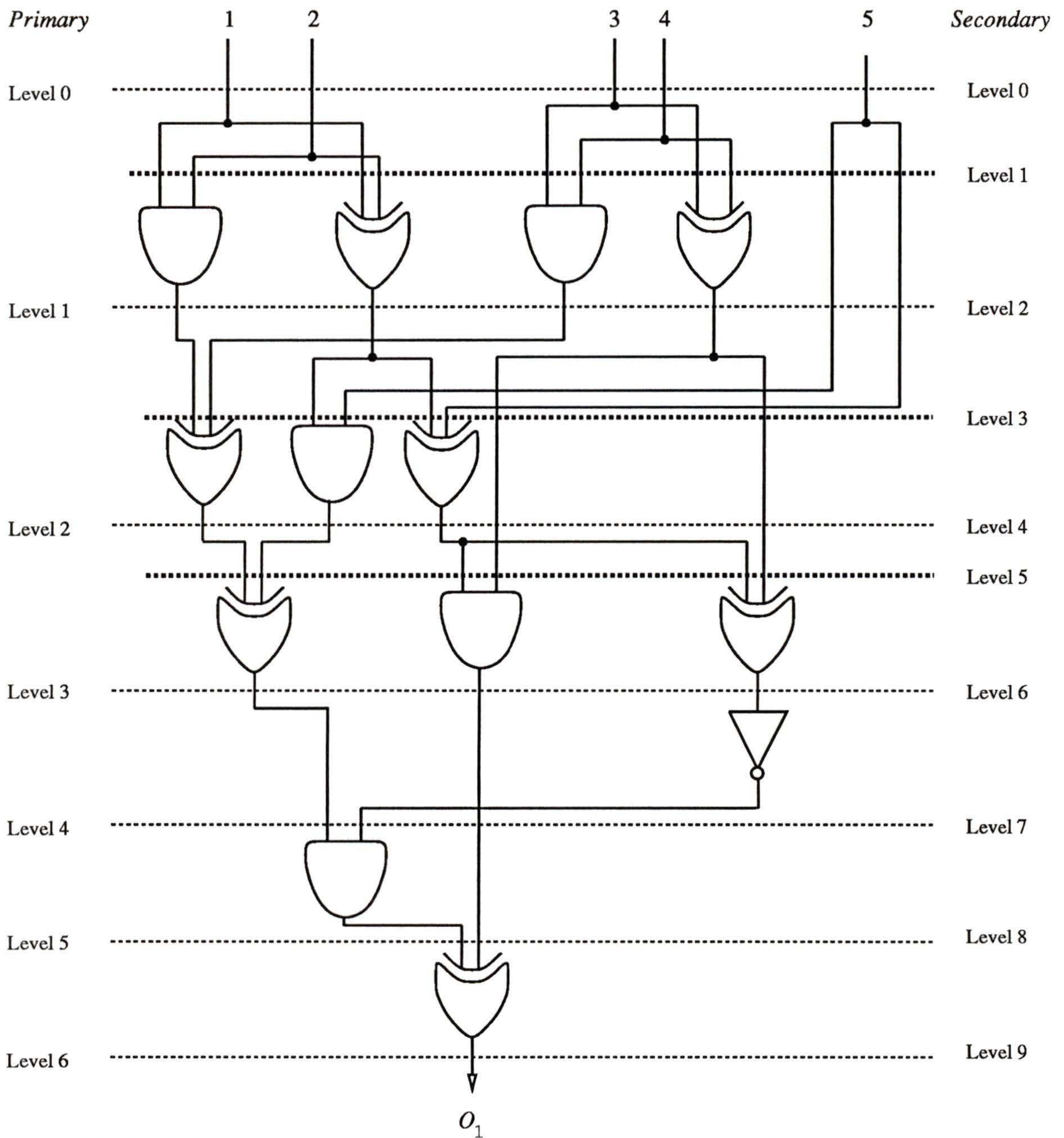


**Table 5.15** Primary logic levels at which two inputs meet.

| INPUTS | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | - | 1 | 2 | 2 | 2 |
| 2 | 1 | - | 2 | 2 | 2 |
| 3 | 2 | 2 | - | 1 | 3 |
| 4 | 2 | 2 | 1 | - | 3 |
| 5 | 2 | 2 | 3 | 3 | - |

- : undefined entry.

$$
\begin{array}{llccccc}
INPUT & : & PI_1 & PI_2 & PI_3 & PI_4 & PI_5 \\
SRL & : & SRL_1 & SRL_3 & SRL_5 & unassigned & unassigned
\end{array}
$$

The following then takes place in the *fourth loop* of the second phase:

Step 2.1 $PI_4$, which is at the top of the list of unassigned inputs (PIlist), is picked.

Step 2.2 At this stage, the unassigned SRLs are $SRL_2$ and $SRL_4$.

Step 2.2.1 The effect of temporarily assigning $PI_4$ to each of these SRLs is determined and is shown in Table 5.16.

Step 2.2.2 The *Total* column indicates that for both choices of SRLs, the same number ($3 = Total + 1$) of adjacent latches feed the inputs to the logic cone (this is expected since the circuit has a single output).

The *tie* is broken by further looking at the levels at which the inputs, that are fed by adjacent latches, "logically" meet. The additional column, *Level*, in the table indicates that this level is: *level* 1 if $PI_4$ is connected to $SRL_4$; *level* 2 if $PI_4$ is connected to $SRL_2$. $SRL_2$ being the better choice is, therefore, selected.

The last input, $PI_5$, is assigned to the remaining latch, $SRL_4$. The connections between the SRLs and the PIs, after applying the *second phase*, of the algorithm are shown in Figure 5.12. The effect of these connections is given in Table 5.17.

**Table 5.16** Effect of temporarily assigning $PI_4$ to each of the unassigned latches.

| SRL | Adjacent-Latches | PIs-Concerned | Level | Affected-Cones | Total |
|---|---|---|---|---|---|
| 2 | 1,2 | 1,4 | 2 | $C_1$ | |
| | 2,3 | 2,4 | 2 | $C_1$ | 2 |
| 4 | 3,4 | 2,4 | 2 | $C_1$ | |
| | 4,5 | 3,4 | 1 | $C_1$ | 2 |

**Table 5.17** Resulting effect of assigning circuit inputs to SRLs using the second phase.

| Adjacent-Latches | PIs-Concerned | Level | Affected-Cones | Total |
|---|---|---|---|---|
| 1,2 | 1,4 | 2 | $C_1$ | 1 |
| 2,3 | 2,4 | 2 | $C_1$ | 1 |
| 3,4 | 2,5 | 2 | $C_1$ | 1 |
| 4,5 | 3,5 | 3 | $C_1$ | 1 |

#### 5.3.3.2 Alternative Method (the *ISSIA* method)

As mentioned earlier, the second phase selects a "suitable" SRL for a given unassigned PI. An alternative method is to select a "suitable" PI for a given SRL (*ISSIA* – Input Separation by Selective Input Assignment). The suggested method proceeds as follows:

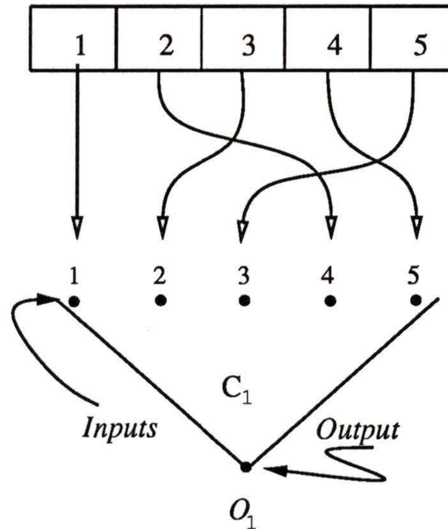
Step 1.0 Sort the PIs in decreasing order of the size of their POset.

Step 2.0 Assign the first latch ($SRL_1$) to the input at the top of the unassigned input list.

Step 3.0 For the adjacent latch, select and assign a "suitable" PI. The "suitable" PI, $i$, satisfies the following conditions:

Step 3.1 the PI, $i$, and last assigned PI have disjoint POset (*i.e.*, do not feed the same logic cones); if such $i$ is not found then,

**Figure 5.12** SRLs to circuit inputs connections obtained using the second phase.




Step 3.2 the PI, $i$, and the last assigned PI should share the minimum number of outputs (*i.e.* the intersection of their POsets is the smallest.) If two or more PIs satisfy this condition, then the best choice is the one that "logically" meets the last assigned PI, $i$, at a level closest to the level of the POs.

Step 4.0 If all PIs have not been assigned, then repeat Step 3.0 above.

Note that the above steps 1.0 and 2.0 are different from Step 1 and Step 2 mentioned earlier for the *first phase* and the *second phase* respectively.

As an example, this algorithm is applied to the circuit in Figure 5.11 The PIs are sorted as in the previous example to obtain the following list: $PI_1$, $PI_2$, $PI_3$, $PI_4$, $PI_5$.

In the *first loop*, the following takes place:

Step 2.0 $SRL_1$ is assigned to $PI_1$ (Figure 5.13).

Step 3.1 Since, the circuit has a single cone, the search in Step 3.1 always fails and therefore Step 3.2 is executed in all the remaining loops.

Step 3.2 A search is conducted for a PI that meets $PI_1$ at a level closest to the level of the POs. This is done by scanning the first row of Table 5.15 (page 81). Since there

are three choices ($PI_3$, $PI_4$ and $PI_5$), the first one, $PI_3$, is selected and assigned to $SRL_2$.

In the *second loop*:

Step 3.2 A search is conducted for a PI that meets the last assigned input, $PI_3$, at a level closest to the level of the POs. This is done by scanning the third row of Table 5.15 (page 81). $PI_5$ being the best choice is selected and assigned to $SRL_3$.

Similarly, in the *third loop* $PI_4$ is assigned to $SRL_4$ and finally, the remaining $PI_2$ is assigned to $SRL_5$.

The connections between the SRLs and the PIs are shown in Figure 5.13 and the effect of these connections are indicated in Table 5.18.

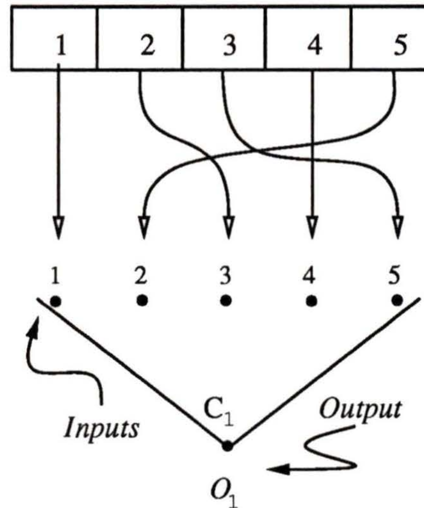
**Table 5.18** Resulting effect of assigning circuit inputs to SRLs using the ISSIA method.

| Adjacent-Latches | PIs-Concerned | Level | Affected-Cones | Total |
|---|---|---|---|---|
| 1,2 | 1,3 | 2 | $C_1$ | 1 |
| 2,3 | 3,5 | 3 | $C_1$ | 1 |
| 3,4 | 4,5 | 3 | $C_1$ | 1 |
| 4,5 | 2,4 | 2 | $C_1$ | 1 |

The *Total* column of Tables 5.17 and 5.18 are identical because the circuit has a single cone only. The *Level* columns indicate that the connections obtained using the alternative method (Table 5.18) may be better than the ones obtained using the second phase (Table 5.17). This is because more PIs in Table 5.18 "logically" meet at levels that are further away from the level of the inputs (*Level* 0).

It should be noted that the simple example used here was chosen to illustrate how the algorithm works and also what difference it can make. The difference shown may in fact make no difference on the fault coverage for the small example circuit.

**Figure 5.13** SRLs to circuit inputs connections obtained using the ISSIA method.




## 5.4 Summary

In this chapter, the details of our algorithm to identify the connections between the SRLs, or the outputs of an LFSR, and the inputs to the circuit were described. Some examples were given to illustrate how the *second phase* of the algorithm works independently (the *ISSLA* method) and also how the *second phase* works in conjunction with the *first phase* (the *ISASLA* method). We also proposed an alternative method (the *ISSIA* method) and gave an example to illustrate how it works.

In the next chapter, we present the simulation results for the *transition fault* coverage of the ISCAS '85 circuits. The results are for the connections obtained by using the *ISSLA* and the *ISASLA* methods. These results are compared with the results of the published methods. The alternative method that was proposed (the *ISSIA* method) was not implemented and is suggested as future work.

# Chapter 6

# Simulation Results and Observations

In this chapter, the simulation results for the *transition fault* coverage of the ISCAS '85 benchmark circuits [BF85] are given. These results are for various methods, including our method, that try to enhance the *transition fault* coverage.

Section 6.1 describes various options of our algorithm and in section 6.1.2, the fault coverage results for these options are presented. These results are then compared with the results for the random connections which were given in section 4.5.2. In section 6.2.2, the results for our method are compared with those for other methods which include, the XLFSR and the ISUD methods.

Section 6.3 briefly describes the experiments to see the effect of *seed value* on the *transition fault* coverage and the *test length.* Some results of these experiments are also presented.

**Fault Simulator and ISCAS '85 Circuits**

The fault simulator, that was used to conduct all the experiments was developed by Mr. Shujian Zhang at the VLSI Research Lab, Computer Science Dept., University of Victoria.

The fault simulator includes many features such as :

1. Simulation of several fault models, for example, stuck-at fault model, delay fault

model and stuck-open fault model.

2. Choice of several binary test pattern generators that include both types of LFSRs (LFSR1 and LFSR2) and Linear Cellular Automata Register (LCAR).

Additional features which make the simulator efficient include :

1. the usage of Parallel Pattern Single Fault Propagation (PPSFP) technique which allows several binary test vectors to be applied simultaneously and hence speed up the simulation process.

2. the usage of fault collapsing technique. This technique reduces the original set of faults to be simulated, by grouping together equivalent faults, and hence, reduces the simulation time.

The combinational circuits used during the simulations were ISCAS '85 benchmark circuits. A brief description of these circuits is given in Table 6.1.

**Table 6.1** ISCAS '85 combinational benchmark circuits.

| Circuit | Circuit Function | Inputs | Outputs | Gates | Transition Faults‡ |
|---|---|---|---|---|---|
| c432 | Priority Decoder | 36 | 7 | 160 | 864 |
| c499 | Error Correcting | 41 | 32 | 202 | 998 |
| c880 | ALU and Control | 60 | 26 | 383 | 1760 |
| c1355 | Error Correcting | 41 | 32 | 546 | 2710 |
| c1908 | Error Correcting | 33 | 25 | 880 | 3816 |
| c2670 | ALU and Control | 233 | 140 | 1193 | 5492 |
| c3540 | ALU and Control | 50 | 22 | 1669 | 7080 |
| c5315 | ALU and Selector | 178 | 123 | 2307 | 10630 |
| c6288 | 16-bit Multiplier | 32 | 32 | 2406 | 12576 |
| c7552 | ALU and Control | 207 | 108 | 3512 | 15106 |

‡Fault collapsing techniques were not applied.

## 6.1 Options of the Algorithm

The algorithm given in section 5.2 has two *phases*, an optional *first phase* and the *second phase*. Each phase, which assigns the primary inputs (PIs) to the shift register latches (SRLs), is preceded by an optional step to sort the PIs of the circuit under test (CUT). Many combination of these options can be tried to obtain connections between the SRLs and the PIs.

The sorting of the PIs may change the order in which the PIs are assigned to the SRLs. This affects the transition coverage at the inputs of the circuit, which in turn, affects the fault coverage of the CUT.

Various combinations of these options were used to obtain connections between the outputs of a LFSR and the PIs of the 10 ISCAS'85 circuits. These connections were then used to generate test patterns and run simulations to obtain the transition fault coverage for the ISCAS'85 circuits. It was observed that the connections for some combination of options performed well for many circuits.

The combination of options are represented as a 3 character **code** $ps_1s_2$ where:

- $p$ represents the phase option and takes values:

  1 : implying only a *single* phase (the *second phase*) of the algorithm was executed.

  2 : implying *both* phases of the algorithm were executed.

- $s_1$ *and* $s_2$ represent the sorting options before executing phase 1 and phase 2 of the algorithm, respectively. The values that can be assigned to both $s_1$ and $s_2$ are:

  0 : sorting the PIs in *decreasing* order of the size of their POset (*i.e* the number of outputs the PIs affect).

  1 : sorting the PIs in *increasing* order of the size of their POset.

  u : unsorted.

  – : implying not applicable.

For example the combination of options, $ps_1s_2$ = "1-0" means:

1. $p$ = '1', only a *single* phase (the *second phase*) of the algorithm was executed.

2. $s_1$ = '-', sorting of the inputs before executing phase 1 is not applicable here since only the second phase ($p$ = '1') is executed.

3. $s_2$ = '0', sorting the PIs in *decreasing* order of the size of their POsets before executing the second phase.

Before running the simulations, the connections obtained by using the options "1-0" and "210" were expected to perform well. The reasons for these expectations are given in section 6.1.1.

### 6.1.1 Options of Interest

Two combination of options, for executing the algorithm of section 5.2, that were of interest are "1-0" and "210". The connections between the SRLs and the PIs that would result by using these options were expected to give a high transition fault coverage.

**Option "1–0"**

The *first* of these two combination of options, $ps_1s_2$ = "1-0", uses only a single phase ($p$ = '1'; the *second phase*) of the algorithm. Before executing the second phase, the PIs are sorted ($s_2$ = '0') in *decreasing* order of the sizes of their POsets. The second phase selects a suitable latch for each PI such that no two adjacent SRLs feed the same logic cone. The ordering of the PIs, as mentioned above, results in the PIs with large POsets appearing at the top of the ordered list. In general, the PIs with large POsets feed many *common* outputs. Hence, such sorting gives these PIs a higher priority to be assigned at an early stage. Since, most of the SRLs are unassigned at an early stage of the second phase, the PIs with large POsets are easily separated by simply assigning them to the alternate SRLs.

At later stages of the second phase, many SRLs, within the SRL chain have been assigned. Thus, at this stage, when a PI is assigned to a SRL, it is *less likely* that no collision will take place. But, by this time it is expected that the unassigned PIs are

those with smaller POsets. Therefore, if a collision takes place, it will affect only a few logic cones and hence a small part of the circuit. The second phase attempts to further minimize this effect. The approach is to assign PIs to adjacent latches such that these PIs do not feed common logic at earlier levels. The PIs are selected by checking the logic level at which the concerned PIs "logically" meet. The closer this level is to the outputs, the better. This way the shift dependency problem affects only a smaller part (*i.e.* the levels closer to the outputs) of the logic. It is also possible that the shift dependency problem may not be significant at logic levels which are far away from the input level. This is similar to the low controllability of nodes which are deep in the circuit.

**Option "210"**

The other option, $ps_1s_2$ = "210", uses both phases ($p$ = '2') of the algorithm. Before executing the first phase, the PIs are sorted ($s_1$ = '1') in *increasing* order of the sizes of their POsets. By doing so, the PIs which feed fewer outputs appear at the top of the sorted list and will, therefore, be assigned first.

The first phase allows multiple PIs to be assigned to a single SRL. The advantage of sorting the PIs, as mentioned in the previous paragraph, is that the chance of finding, two or more, PIs at the top of the list having mutually exclusive POsets is high. Hence, during the first phase, two or more of these PIs can easily be found and assigned to the same latch. With this approach, more PIs are assigned to fewer SRLs, thus leaving room (more SRLs) for PIs with large POsets to be easily separated in the later stages of the first phase.

An important point regarding the functioning of the first phase is recalled here. After multiple PIs are assigned to a single SRL, the adjacent SRL is skipped (*i.e.*, left unassigned). Although, some SRLs are skipped, it is possible to obtain collision-free connections with fewer SRLs than the number of inputs to the circuit. This was possible for the c5315 ISCAS'85 circuit.

With option $ps_1s_2$ = "210", the second phase is executed only if all the PIs have not been assigned at the end of the first phase. The first phase ends when either, all inputs have been assigned, or all alternate latches have been assigned.

Before executing the second phase, the remaining PIs are sorted ($s_2$ = '0') in *decreasing* order of the sizes of their POsets. This is done to give a higher priority to the PIs with large POsets to be assigned before PIs with smaller POsets. The reason for doing this is that, at the end of the first phase, the unassigned SRLs are the ones between two assigned SRLs (*i.e.* the SRLs that are skipped in the first phase). If the PIs are not sorted, as mentioned above, then the "good" spots within the SRL chain will get assigned to the PIs with smaller POsets. This leaves the "bad" spots for the PIs with large POsets and will result in multiple collisions.

### 6.1.2 Results

In section 6.1.1 the combination of options that were of interest for executing the algorithm (section 5.2) were mentioned. This section gives the transition fault coverage results for the ISCAS'85 circuits. These are the results obtained by using the connections between the outputs of a LFSR and the PIs of the CUT, as generated by the algorithm using various options. The ten different combination of options that were tried are:

| option no. | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| code | $1-0$ | $1-u$ | 200 | 201 | 210 | $1-1$ | 211 | $2u0$ | $2u1$ | $2uu$ |

These *codes* ($ps_1s_2$) are explained below:

- 1-0 : Second phase only ($p$ = 1); PIs sorted in *decreasing* order ($s_2$=0) of the sizes of their POsets.
- 1-u : Second phase only ($p$ = 1); PIs not sorted ($s_2$=u).
- 200 : Both phases ($p$ = 2); PIs sorted in *decreasing* order of the sizes of their POsets ($s_1$ = $s_2$=0).
- 201 : Both phases ($p$ = 2); PIs sorted in *decreasing* order ($s_1$ = 0) of the sizes of their POsets before the *first phase*. PIs sorted in *increasing* order ($s_2$=1) before the *second* phase.

- 210 : Both phases ($p$ = 2); PIs sorted in *increasing* order ($s_1$ = 1) of the sizes of their POsets before the *first phase.* PIs sorted in *decreasing* order ($s_2$=0) before the *second* phase.

- 1-1 : Second phase only ($p$ = 1); PIs sorted in *increasing* order ($s_2$=1) of the sizes of their POsets.

- 211 : Both phases ($p$ = 2); PIs sorted in *increasing* order of the sizes of their POsets ($s_1 = s_2$=1).

- 2u0 : Both phases ($p$ = 2); PIs *not sorted* before the *first phase* ($s_1 = u$); PIs sorted in *decreasing* order ($s_2$=0) before the *second* phase.

- 2u1 : Both phases ($p$ = 2); PIs *not sorted* before the *first phase* ($s_1 = u$); PIs sorted in *increasing* order ($s_2$=1) before the *second* phase.

- 2uu : Both phases ($p$ = 2); PIs *not sorted* ($s_1 = s_2 = u$).

Out of these 10 options, the first five gave the best observed fault coverage for many circuits. These five options include the options of interest mentioned in section 6.1.1.

Tables 6.2 (page 93) and 6.3 (page 94) give the transition fault coverage for the first five options and the last five options, of the algorithm, respectively. These tables include the results for both orientations of the LFSR. In one orientation of the LFSR, the feed back goes into the leftmost cell (or the first cell) of the LFSR and in the other orientation, the feed back goes into the rightmost cell (or the last cell) of the LFSR. These orientations will be, respectively, referred to as FBf (feedback to *the first* cell) and FBl (feedback to *the last* cell) orientations. Table 6.4 summarizes tables 6.2 and 6.3 by indicating the best observed transition fault coverage for the ISCAS'85 circuits among the 10 options.

**Table 6.2** Fault coverage for ISCAS'85 circuits using both orientations of LFSR and connections for the first 5 options of the algorithm.

| Circuit name | LFSR orientation | Fault coverage for various options (%) | | | | |
|---|---|---|---|---|---|---|
| | | 1-0 | 1-u | 200 | 201 | 210 |
| c432 | FBf | 98.84 | 98.61 | 98.84 | 98.84 | 98.84 |
| | FBl | 98.84 | 98.26 | 98.84 | 98.84 | 98.84 |
| c499 | FBf | 99.20 | 99.20 | 99.20 | 99.20 | 99.20 |
| | FBl | 99.20 | 99.20 | 99.20 | 99.20 | 99.20 |
| c880 | FBf | 99.94 | 99.55 | 99.89 | 99.83 | 99.83 |
| | FBl | 99.77 | 99.66 | 99.77 | 99.77 | 99.77 |
| c1355 | FBf | 98.97 | 98.97 | 98.97 | 98.97 | 98.97 |
| | FBl | 99.04 | 99.04 | 99.04 | 99.04 | 99.04 |
| c1908 | FBf | 99.71 | 99.40 | 99.71 | 99.69 | 99.61 |
| | FBl | 99.71 | 99.71 | 99.71 | 99.71 | 99.61 |
| c2670 | FBf | 92.21 | 94.23 | 89.29 | 89.29 | 89.13 |
| | FBl | 91.73 | 93.24 | 91.42 | 91.42 | 89.18 |
| c3540 | FBf | 95.56 | 96.26 | 95.56 | 95.69 | 95.58 |
| | FBl | 96.12 | 95.20 | 96.12 | 95.83 | 95.89 |
| c5315 | FBf | 99.41 | 99.37 | 99.41 | 99.41 | 99.41 |
| | FBl | 99.41 | 99.24 | 99.41 | 99.41 | 99.41 |
| c6288 | FBf | 99.20 | 98.99 | 99.20 | 99.20 | 99.19 |
| | FBl | 99.19 | 99.01 | 99.19 | 99.19 | 99.20 |
| c7552 | FBf | 97.73 | 97.42 | 97.55 | 97.71 | 98.01 |
| | FBl | 97.48 | 97.24 | 97.33 | 97.59 | 98.17 |

Fault collapsing techniques were not applied.

**Table 6.3** Fault coverage for ISCAS'85 circuits using both orientations of LFSR and connections for the last 5 options of the algorithm.

| Circuit name | LFSR orientation | Fault coverage for various options (%) | | | | |
|---|---|---|---|---|---|---|
| | | 1-1 | 211 | 2u0 | 2u1 | 2uu |
| c432 | FBf | 98.84 | 98.84 | 98.26 | 98.38 | 98.61 |
| | FBl | 98.84 | 98.84 | 98.38 | 98.50 | 98.26 |
| c499 | FBf | 99.20 | 99.20 | 99.20 | 99.20 | 99.20 |
| | FBl | 99.20 | 99.20 | 99.20 | 99.20 | 99.20 |
| c880 | FBf | 99.20 | 99.89 | 99.83 | 99.89 | 99.09 |
| | FBl | 99.03 | 99.77 | 99.77 | 99.89 | 99.43 |
| c1355 | FBf | 98.97 | 98.97 | 98.97 | 98.97 | 98.97 |
| | FBl | 99.04 | 99.04 | 99.04 | 99.04 | 99.04 |
| c1908 | FBf | 99.61 | 99.61 | 99.71 | 99.69 | 99.40 |
| | FBl | 99.61 | 99.61 | 99.71 | 99.71 | 99.71 |
| c2670 | FBf | 89.88 | 89.77 | 92.24 | 91.51 | 91.51 |
| | FBl | 85.54 | 90.02 | 93.08 | 92.44 | 92.44 |
| c3540 | FBf | 95.82 | 95.82 | 96.14 | 96.09 | 96.26 |
| | FBl | 95.61 | 95.61 | 96.12 | 95.64 | 95.20 |
| c5315 | FBf | 99.25 | 99.41 | 99.41 | 99.41 | 99.41 |
| | FBl | 99.12 | 99.41 | 99.41 | 99.41 | 99.41 |
| c6288 | FBf | 99.19 | 99.19 | 98.99 | 98.98 | 98.99 |
| | FBl | 99.20 | 99.20 | 99.01 | 99.00 | 99.01 |
| c7552 | FBf | 98.15 | 97.95 | 97.83 | 97.89 | 97.51 |
| | FBl | 98.01 | 97.86 | 97.63 | 97.76 | 97.24 |

Fault collapsing techniques were not applied.

**Table 6.4** Summary for best observed fault coverage among various options of the algorithm.

| Circuit | Options of the algorithm | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| name | 1-0 | 1-u | 200 | 201 | 210 | 1-1 | 211 | 2u0 | 2u1 | 2uu |
| c432 | * | | * | * | * | * | * | | | |
| c499 | * | * | * | * | * | * | * | * | * | * |
| c880 | * | | | | | | | | | |
| c1355 | * | * | * | * | * | * | * | * | * | * |
| c1908 | * | * | * | * | | | | * | * | * |
| c2670 | | * | | | | | | | | |
| c3540 | | * | | | | | | | | * |
| c5315 | * | | * | * | * | | * | * | * | * |
| c6288 | * | | * | * | * | * | * | | | |
| c7552 | | | | | * | | | | | |
| Total | 7 | 5 | 6 | 6 | 6 | 4 | 5 | 4 | 4 | 5 |

* Best observed fault coverage.

## 6.2 Comparing Results

Two methods (XLFSR and ISUD), which appeared in the literature, that try to enhance the transition fault coverage by re-arranging the connections between the SRLs and the inputs to the CUT were described in sections 4.5.1 and 4.5.3. The experiments to see the effect of randomly generated connections between the outputs of a LFSR and the PIs of a circuit were described in section 4.5.2. In the following sections, the simulation results for transition fault coverage obtained by using the above mentioned methods are compared with the results for the algorithm given in section 6.1.2. In section 6.2.1, the results for the algorithm are compared with the results for random connections (R-CXN). In section 6.2.2, the results for the algorithm are compared with the results for the other methods

(XLFSR and ISUD).

### 6.2.1 Comparing Results for the Algorithm with Random Connections

Tables 6.5 and 6.6 give the statistical results for the experiments with random connections which were mentioned in section 4.5.2. The tables include the minimum, maximum, mean and the standard deviation values for:

1. the transition fault coverage; and
2. the number of undetected faults;

The experiments are briefly described here. A hundred randomly generated, distinct, connections between the outputs of a LFSR and the inputs of the ISCAS'85 circuits were used. A 100,000 test patterns were applied at the primary inputs (PIs) of the circuits. The seed for each simulation run was a vector of alternating binary value beginning with '1' (*i.e.* 101010...). For each of the 100 connections, the simulation was run for both orientations (FBf and FBl) of the LFSR.

Table 6.5 gives the results for FBf orientation of the LFSR where each pattern is generated by a right shift of its predecessor. Table 6.6 gives the results for the FBl orientation of the LFSR where each pattern is generated by a left shift of its predecessor. By comparing the fault coverage values in these two tables, it is seen that the *mean* fault coverage values do not differ much. But, the fault coverage can significantly vary by trying both orientations for a fixed set of connections (see Table 4.2, page 44). Table 6.7 combines the results for both orientations to give statistical results for 200 samples instead of a 100 each as in Table 6.5 and 6.6.

**Table 6.5** Statistical results for ISCAS'85 circuits using a 100 random connections and FBf orientation of LFSR.

| Circuit | Coverage (%) | | | | Undetected faults | | | |
|---|---|---|---|---|---|---|---|---|
| name | Min. | Max. | Mean | Std-Dev | Min. | Max. | Mean | Std-Dev |
| c432 | 97.11 | 98.84 | 98.19 | 0.34 | 10 | 25 | 15.66 | 2.93 |
| c499 | 99.00 | 99.20 | 99.16 | 0.06 | 8 | 10 | 8.43 | 0.59 |
| c880 | 97.84 | 99.83 | 99.23 | 0.40 | 3 | 38 | 13.53 | 7.09 |
| c1355 | 98.19 | 99.08 | 98.69 | 0.17 | 25 | 49 | 35.51 | 4.48 |
| c1908 | 98.24 | 99.69 | 99.21 | 0.32 | 12 | 67 | 30.02 | 12.22 |
| c2670 | 83.21 | 88.00 | 84.65 | 1.20 | 659 | 922 | 843.17 | 65.95 |
| c3540 | 91.84 | 96.10 | 94.40 | 1.25 | 276 | 578 | 396.72 | 88.30 |
| c5315 | 98.45 | 99.38 | 99.18 | 0.20 | 66 | 165 | 86.64 | 21.46 |
| c6288 | 99.09 | 99.25 | 99.18 | 0.03 | 94 | 114 | 102.76 | 3.31 |
| c7552 | 94.91 | 96.37 | 95.71 | 0.26 | 548 | 769 | 648.64 | 39.30 |

**Table 6.6** Statistical results for ISCAS'85 circuits using a 100 random connections and FBl orientation of LFSR.

| Circuit | Coverage (%) | | | | Undetected faults | | | |
|---|---|---|---|---|---|---|---|---|
| name | Min. | Max. | Mean | Std-Dev | Min. | Max. | Mean | Std-Dev |
| c432 | 96.53 | 98.84 | 98.16 | 0.40 | 10 | 30 | 15.90 | 3.44 |
| c499 | 99.00 | 99.20 | 99.16 | 0.06 | 8 | 10 | 8.43 | 0.59 |
| c880 | 98.12 | 99.89 | 99.21 | 0.38 | 2 | 33 | 13.84 | 6.73 |
| c1355 | 98.15 | 99.04 | 98.67 | 0.17 | 26 | 50 | 36.12 | 4.58 |
| c1908 | 98.35 | 99.71 | 99.21 | 0.31 | 11 | 63 | 30.09 | 12.01 |
| c2670 | 83.30 | 88.40 | 84.71 | 1.22 | 637 | 917 | 839.87 | 66.93 |
| c3540 | 90.61 | 95.93 | 94.83 | 1.12 | 288 | 665 | 366.05 | 79.31 |
| c5315 | 98.29 | 99.37 | 99.21 | 0.17 | 67 | 182 | 83.77 | 18.26 |
| c6288 | 99.11 | 99.24 | 99.18 | 0.03 | 95 | 112 | 102.62 | 3.55 |
| c7552 | 94.94 | 96.34 | 95.71 | 0.27 | 553 | 764 | 647.76 | 40.66 |

**Table 6.7** Statistical results for ISCAS'85 circuits using a 100 random connections and both orientations (FBf and FBl) of the LFSR.

| Circuit | Coverage (%) | | | | Undetected faults | | | |
|---|---|---|---|---|---|---|---|---|
| name | Min. | Max. | Mean | Std-Dev | Min. | Max. | Mean | Std-Dev |
| c432 | 96.53 | 98.84 | 98.17 | 0.37 | 10 | 30 | 15.78 | 3.20 |
| c499 | 99.00 | 99.20 | 99.16 | 0.06 | 8 | 10 | 8.43 | 0.59 |
| c880 | 97.84 | 99.89 | 99.22 | 0.39 | 2 | 38 | 13.69 | 6.91 |
| c1355 | 98.15 | 99.08 | 98.68 | 0.17 | 25 | 50 | 35.81 | 4.54 |
| c1908 | 98.24 | 99.71 | 99.21 | 0.32 | 11 | 67 | 30.05 | 12.12 |
| c2670 | 83.21 | 88.40 | 84.68 | 1.21 | 637 | 922 | 841.52 | 66.46 |
| c3540 | 90.61 | 96.10 | 94.61 | 1.21 | 276 | 665 | 381.38 | 85.32 |
| c5315 | 98.29 | 99.38 | 99.20 | 0.19 | 66 | 182 | 85.20 | 19.98 |
| c6288 | 99.09 | 99.25 | 99.18 | 0.03 | 94 | 114 | 102.69 | 3.43 |
| c7552 | 94.91 | 96.37 | 95.71 | 0.26 | 548 | 769 | 648.20 | 39.98 |

**Table 6.8** Statistical results for ISCAS'85 circuits for results obtained using the first 5 options of the algorithm.

| Circuit | Coverage (%) | | | | Undetected faults | | | |
|---|---|---|---|---|---|---|---|---|
| name | Min. | Max. | Mean | Std-Dev | Min. | Max. | Mean | Std-Dev |
| c432 | 98.26 | 98.84 | 98.76 | 0.18 | 10 | 15 | 10.70 | 1.55 |
| c499 | 99.20 | 99.20 | 99.20 | 0.00 | 8 | 8 | 8.00 | 0.00 |
| c880 | 99.55 | 99.94 | 99.78 | 0.11 | 1 | 8 | 3.90 | 1.87 |
| c1355 | 98.97 | 99.04 | 99.00 | 0.04 | 26 | 28 | 27.00 | 1.00 |
| c1908 | 99.40 | 99.71 | 99.66 | 0.10 | 11 | 23 | 13.10 | 3.65 |
| c2670 | 89.13 | 94.23 | 91.12 | 1.74 | 317 | 597 | 487.90 | 95.59 |
| c3540 | 95.20 | 96.26 | 95.78 | 0.31 | 265 | 340 | 298.70 | 21.79 |
| c5315 | 99.24 | 99.41 | 99.39 | 0.05 | 63 | 81 | 65.20 | 5.40 |
| c6288 | 98.99 | 99.20 | 99.15 | 0.08 | 101 | 127 | 106.40 | 9.82 |
| c7552 | 97.24 | 98.17 | 97.62 | 0.28 | 277 | 417 | 359.10 | 41.86 |

Table 1.8 gives the statistical results for the first five set of options mentioned in section 1.1.2 (Table 1.2). The *transition fault* coverage results obtained using various combination of options for the algorithm will be referred to as **H-CXN** results. The *H* implies that heuristics were used to obtain the connections (CXN). Similarly, the best observed transition fault coverage results obtained using random connections will be referred to as R-CXN results. For comparison purposes, the results of Tables 1.7 and 1.8 are displayed in Figure 1.1 (page 16). There are ten groups of results in Figure 1.1, for the ten ISCAS'85 circuits. Each group has four bars. The first three are the minimum (R-CXN.Min), mean (R-CXN.Mean) and maximum (R-CXN.Max) observed transition fault coverage in the experiments for random connections. The fourth bar in each group indicates the maximum observed transition fault coverage (H-CXN.Max) using the connections obtained from the algorithm. It can be seen that for 5 out of 10 ISCAS'85 circuits, the H-CXN.Max fault coverage is greater than R-CXN.Max. For the circuits c432, c499 and c1908 the H-CXN.Max equals the R-CXN.Max. These could possibly be the maximum achievable transition fault coverage for the circuits. For the c1355 circuit, the H-CXN.Max is less than R-CXN.Max by only 0.04%. This difference is equivalent to only a single fault being undetected.

Thus the *transition fault* coverage results, for connections obtained using the algorithm (H-CXN) are not only as good as the best result for the random connections (R-CXN) but also better for many circuits. The obvious advantage over the random sample is that only a few simulations need to be run in order to find the connections which give the highest fault coverage. The connections for the options $ps_1s_2$ = "1-0" and $ps_1s_2$ = "210", that were expected to perform well in fact, performed very well for most of the ISCAS'85 circuits.

Although, in Table 1.4 the first 5 options appear to give very good performance, this does not mean that the other five do not do well. In fact, many results in Table 1.3 are seen to be quite close to the best observed results.

### Assessing Quality of Connections

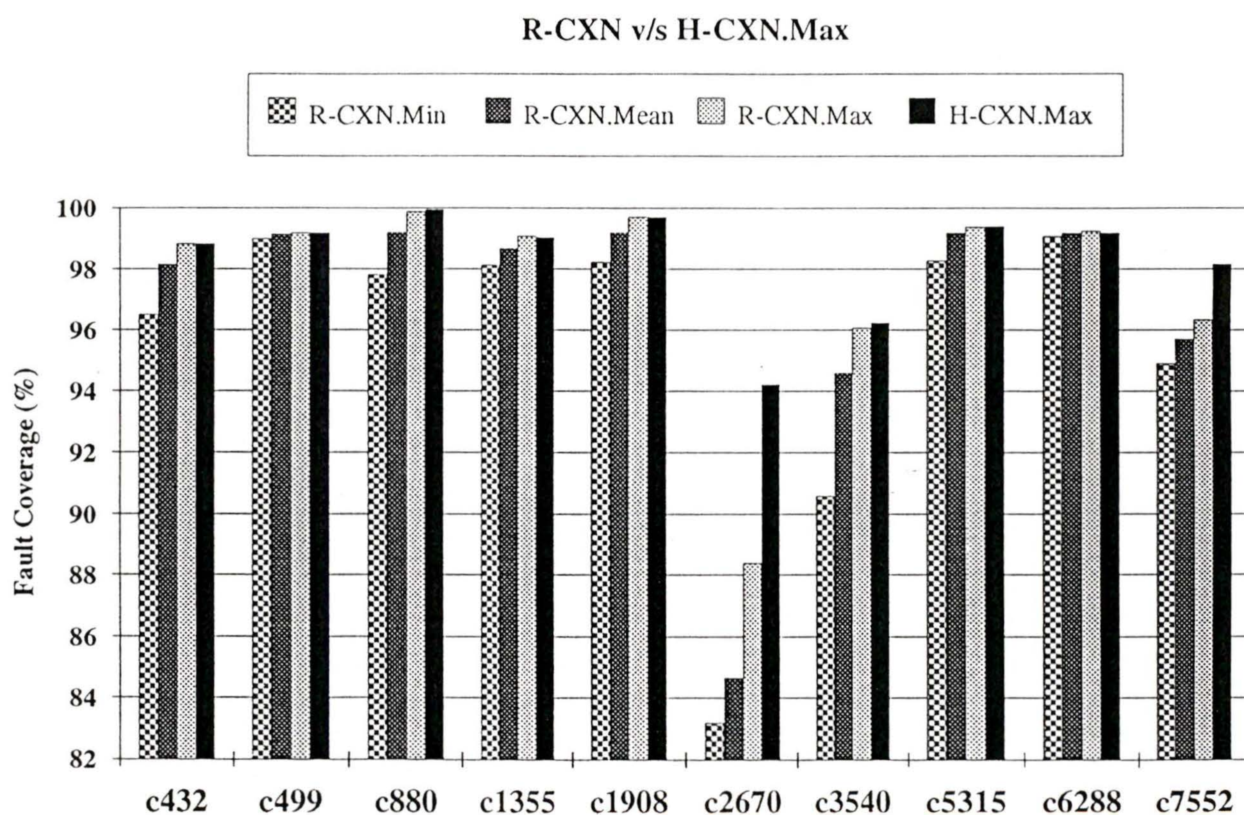
One way to assess the quality of the connections identified by our algorithm, would be to compare them to randomly selected connections. The idea is to generate some number of random connections and to use the distribution of that data as the basis for a statistical test to determine the probability of a randomly chosen set of connections yielding better fault coverage than the connections selected by the algorithm.

One hundred sets of connections were chosen at random for each of the 10 ISCAS'85 circuits. Using these connections, the fault coverage was determined for both orientations of the LFSR. These results show there is not a consistent distribution for the circuits considered. For example, the circuit c499 has only 3 distinct values, whereas the data for c2670 appears to follow an $f - distribution$. Only, two of the 10 ISCAS '85 circuits, namely c1355 and c7552, appear to yield normally distributed data.

Appropriate statistical tests can be applied to the fault coverage results of any circuit if its distribution is known. For example, the results for the c7552 circuit have a reasonable normal distribution and the test shows that the probability of a randomly selected set of connections having a better result than the best result of the algorithm is effectively 0 (zero).

The sample of 200 connections (100 randomly generated $\times$ both orientations of LFSR), used in the experiments, is very small compared to the $n!$ possible connections for a circuit with $n$ inputs, where $n > 20$. Since running simulations for all possible connections is impossible, an alternative would be to work with smaller circuits with less than 10 inputs. Unfortunately, because of the inconsistency in the distributions, the test results for individual circuits give no information for the other circuits, and small circuits provide no insight into the behaviour of large circuits.

**Figure 6.1** Comparing the results for R-CXN with the best observed results for H-CXN.



### 6.2.2 Comparing Results for the Algorithm with Other Methods

The *transition fault* coverage for the ISCAS'85 circuits obtained by using various methods were given in the earlier sections and chapters. In this section, the best observed fault coverage in each of these methods are compared with the best observed *transition fault* coverage for the 5 options for the algorithm.

Table 6.9 lists the best observed *transition fault* coverage in each scheme and also the mean (MEAN) *transition fault* coverage for random connections. The fault collapsing techniques were not applied on the sets of modeled faults. The other columns are for the *transition fault* coverage. The table contains results for :

- NP-CXN – Null Permutation connections ( i.e. Output $k$ of the LFSR connected to input $k$ of the CUT.)
- MEAN – The *mean* transition fault coverage for a sample of 200 random connections between the outputs of a LFSR and the inputs of a circuit.
- XLFSR – The cross over connections (section 4.5.1).
- ISUD – Input Separation Using Dummies method (section 4.5.3).
- H-CXN – Best observed coverage for connections obtained from the Algorithm.

#### Comparing NP-CXN with XLFSR

When the results for the cross over method (XLFSR) are compared with those for null permutation connections (NP-CXN), the XLFSR appears to perform extremely well. For 9 out of 10 ISCAS'85 circuits, the XLFSR performs much better than the NP-CXN. For one circuit, c6288, the fault coverage achieved using XLFSR was lower than that obtained for NP-CXN. The reason for such degradation in the performance of XLFSR was explained in section 4.5.2.

**Table 6.9** Comparing fault coverage of various methods.

| Circuit name | Transition faults‡ | Transition fault coverage (%) | | | | |
|---|---|---|---|---|---|---|
| | | NP-CXN | MEAN | XLFSR | ISUD | H-CXN |
| c432 | 864 | 97.22 | 98.17 | 98.84 | 98.84 | 98.84 |
| c499 | 998 | 99.10 | 99.16 | 99.20 | 99.20 | 99.20 |
| c880 | 1760 | 97.16 | 99.22 | 99.60 | 100 | 99.94 |
| c1355 | 2710 | 97.23 | 98.68 | 99.00 | 98.97 | 99.04 |
| c1908 | 3816 | 97.96 | 99.21 | 99.42 | 99.71 | 99.71 |
| c2670 | 5492 | 89.31 | 84.68 | 93.65 | 86.13 | 94.23 |
| c3540 | 7080 | 90.54 | 94.61 | 93.49 | 96.37 | 96.26 |
| c5315 | 10630 | 97.15 | 99.20 | 99.21 | 99.41 | 99.41 |
| c6288 | 12576 | 99.20 | 99.18 | 99.00 | 99.32 | 99.20 |
| c7552 | 15106 | 97.45 | 95.71 | 97.84 | 97.78 | 98.17 |

‡Fault collapsing techniques were not applied.

NP-CXN : Null Permutation Connections.

MEAN: *Mean* coverage for Random connections.

XLFSR: Cross Over Permutation.

ISUD : Input Separation Using Dummies method.

H-CXN: Best observed result for connections obtained using the algorithm.

### Comparing mean coverage for the R-CXN (MEAN) with XLFSR

When the results for the mean (MEAN) *transition fault* coverage for random connections are compared with the results for XLFSR, the difference between these two sets of results is not as significant as that between NP-CXN and XLFSR. In fact, for two circuits, c3540 and c6288, the MEAN results are better than the results for XLFSR.

**Table 6.10** Summary for the best observed fault coverage for ISCAS‘85 circuits using various methods.

| Circuit | NP-CXN | XLFSR | ISUD | H-CXN |
|---|---|---|---|---|
| c432 | | * | * | * |
| c499 | | * | * | * |
| c880 | | | * | |
| c1355 | | | | * |
| c1908 | | | * | * |
| c2670 | | | | * |
| c3540 | | | * | |
| c5315 | | | * | * |
| c6288 | | | * | |
| c7552 | | | | * |

* Best observed transition fault coverage.

### ISUD Results

The input separation using dummies method (ISUD) was observed to give the best fault coverage for 7 out of 10 ISCAS’85 circuits. Surprisingly, the ISUD method gives a very low fault coverage for c2670. But for all ISCAS’85 circuits, the results for the ISUD were better than the MEAN results for random connections. Table 6.11 shows the overhead and the delay fault coverage for the ISCAS ’85 circuits as given in [SR92]. It should be noted that these results differ from those for ISUD method in Table 6.9 because the later are the coverage results for the exact number of modelled faults, whereas the former may be for the equivalent classes. Moreover, the seed values in two experiments may not be the same.

**Table 6.11** Overhead and Coverage for Delay Fault Simulation of the ISCAS '85 Circuits as given in the [SR92].

| Cct. | PIs | POs | Dummies | Total‡ | Overhead % | Coverage % |
|---|---|---|---|---|---|---|
| c432 | 36 | 7 | 28 | 71 | 77 | 98.72 |
| c499 | 41 | 32 | 8 | 81 | 19 | 99.13 |
| c880 | 60 | 26 | 3 | 89 | 5 | 100 |
| c1355 | 41 | 32 | 8 | 81 | 19 | 98.83 |
| c1908 | 33 | 25 | 7 | 65 | 21 | 99.69 |
| c2670 | 233 | 140 | 0 | 244 | 0 | 86.48 |
| c3540 | 50 | 22 | 27 | 99 | 54 | 96.55 |
| c5315 | 178 | 123 | 0 | 179 | 0 | 99.30 |
| c6288 | 32 | 32 | 0 | 63 | 0 | 99.32 |
| c7522 | 207 | 108 | 74 | 389 | 35 | 95.42 |

‡Total number of latches required. These numbers were obtained by implementing the ISUD method.

### H-CXN results

The H-CXN results were found to be better than those for the NP-CXN and the XLFSR methods, and the MEAN results for random connections. It was indicated in section 6.2.1 that the H-CXN results for 5 out of 10 ISCAS'85 circuits were found to be better than the best observed results for R-CXN (for three circuits the H-CXN results were as good as the best observed R-CXN results and for the results of the remaining two circuits the difference was small.)

Table 6.10 gives a summary of the best observed transition fault coverage for the ISCAS'85 circuits using various methods (excluding the R-CXN method). The H-CXN results were the best observed results for 7 out of 10 ISCAS'85 circuits (for the remaining 3, the difference between the results for the ISUD method and the H-CXN was very small). Moreover, the H-CXN results for c1355, c2670 and c7552 were better than

those for ISUD method.

**Advantages of the Algorithm**

These results suggest that the connections obtained using the algorithm have a very good chance of giving a high transition fault coverage. The advantages of using the algorithm are:

1. the algorithm deterministically avoids or minimizes *not only* the collisions *but also* the effects of the collisions based on the structure of the CUT. In the XLFSR method, the set of connections are fixed and the implicit assumption is that the inputs to logic cones are all adjacent inputs of the circuit. This is not always true and therefore the XLFSR scheme does not work for all circuits. An example is the c6288 ISCAS'85 circuit. This was explained in section 4.5.2.

2. Unlike the ISUD method, the algorithm restricts the number of latches to be equal to the number of inputs in the circuit, thus keeping the cost low. Moreover in some special cases, a collision-free connection is possible using fewer latches (or using an LFSR of size less) than the number of inputs in the CUT.

3. Five sets of options for the algorithm were identified to generate connections that give a high fault coverage. The advantage over random connections is that only a few simulations need to be run in order find the connections that give a high fault coverage. When the *mean* fault coverage results for the 200 samples of random connections (Table 6.7, page 98) are compared with the *mean* fault coverage results for the 5 options (5 options $\times$ 2 *orientations of LFSR* = 10 sets of connections) obtained from the algorithm (Table 6.8, page 98), it can be seen that the later are better than the former. In Table 6.8, the small difference between the minimum and maximum fault coverage values, together with high mean fault coverage values are an indication of favourable results for all 5 options.

   Moreover, the *maximum fault coverage* values in the two tables suggest that the results for the *connections obtained from the algorithm* are not only as good as those for the random connections but in many cases better.

## 6.3 Effect of Seeds

In the previous sections and chapters, the major concern was the effect on the *transition fault* coverage by the connections between the SRLs and the inputs to the CUT. Also the effect of using both orientations of the LFSR (FBf and FBl) on the *transition fault* coverage was looked at. Trying both orientations of the LFSR can be considered as trying two different sets of connections between the outputs of the LFSR and the inputs of the circuit under test. The interesting part of trying both orientations is that, although each cell of the LFSR goes through the same sequence of binary values, the effect on the fault coverage can be significant. The orientations of the LFSR not only affects the fault coverage but also the test length. In certain cases, where the fault coverage is not affected by trying both orientations of the LFSR, the effect on the test length was observed to be significant.

After seeing the effect of connections between the SRLs to PIs and the effect of both orientations of a LFSR on the *transition fault* coverage, it was of interest to see the effect of different seed value. Therefore, some experiments were conducted by fixing:

1. the orientation of the LFSR; and
2. the connections between the LFSR and the inputs to the circuit.

The details of the experiments and the results are given in the following section.

### 6.3.1 Experiments

To see the effect of different seed value on the transition fault coverage and the test length, some experiments were conducted by running simulations using the following:

1. Thirty randomly generated, distinct, seeds were used for each of the 10 ISCAS'85 circuits;
2. The orientation of the LFSR was fixed such that the feed back goes into the first cell (or the left most cell) of the LFSR (the FBf orientation). Therefore, each pattern generated was obtained by a *right* shift of the previous one.

3. The connections between the LFSR and the inputs for each circuit were fixed. These connections were obtained by using the algorithm with option $ps_1s_2$ = '1-0' (*i.e.* only the second phase of the algorithm was executed and the inputs were sorted in decreasing order of their POsets before executing the second phase.) Note that, the option $ps_1s_2$ = '1-0' is the one which gave the best observed transition fault coverage for most of the ISCAS'85 circuits (Table 6.4, page 95).

4. The simulations were run for 100,000 patterns.

**Effect of Seeds on the Fault Coverage**

The statistical results are shown in Table 6.12. The results indicate that for many circuits, the seed value does not have much effect, if any, on the transition fault coverage.

**Table 6.12** Statistical results for ISCAS'85 circuits using 30 random seeds.

| Circuit | Coverage (%) | | | | Last effective pattern | | | |
|---|---|---|---|---|---|---|---|---|
| name | Min. | Max. | Mean | Std-Dev | Min. | Max. | Mean | Std-Dev |
| c432 | 98.84 | 98.84 | 98.84 | 0.00 | 2048 | 18944 | 7705.60 | 4079.62 |
| c499 | 99.20 | 99.20 | 99.20 | 0.00 | 1024 | 3584 | 2201.60 | 567.07 |
| c880 | 99.94 | 99.94 | 99.94 | 0.00 | 8192 | 59392 | 32110.93 | 10490.52 |
| c1355 | 98.67 | 99.08 | 98.90 | 0.10 | 52224 | 99584 | 88046.93 | 13027.35 |
| c1908 | 99.71 | 99.71 | 99.71 | 0.00 | 18688 | 95232 | 45175.47 | 17010.13 |
| c2670 | 83.65 | 88.53 | 85.19 | 1.29 | 56320 | 99584 | 89506.13 | 9643.08 |
| c3540 | 95.56 | 95.58 | 95.58 | 0.00 | 19200 | 95744 | 54980.27 | 19727.58 |
| c5315 | 99.41 | 99.41 | 99.41 | 0.00 | 3584 | 17920 | 7005.87 | 2652.60 |
| c6288 | 99.20 | 99.20 | 99.20 | 0.00 | 256 | 1024 | 622.93 | 194.78 |
| c7552 | 95.13 | 95.78 | 95.47 | 0.17 | 90368 | 100000 | 97031.47 | 2569.11 |

Fault collapsing techniques were not applied.

For the six ISCAS'85 circuits, c432, c499, c1908, c5315, c6288 and c880, it was seen that the fault coverage remained unchanged. For the c3540 circuit, the difference between the maximum and the minimum observed fault coverage was only 0.02%. This difference is equivalent to a single undetected fault.

A big difference in the minimum and maximum transition fault coverage was seen for the circuit c2670. A smaller, yet significant, difference in fault coverage was also seen for the c7552 circuit. Both circuits have over 200 inputs. Moreover, the number of inputs for some of the logic cones in these circuits exceed 100. On the other hand, for the remaining ISCAS'85 circuits, with the exception of c5315, the maximum number of inputs in a logic cone do not exceed 50. The c5315 circuit has some logic cones with 67 inputs. Hence, a test length of 100,000 patterns cannot be considered to be long enough for big circuits like c2670 and c7552. Thus, the seed value significantly affect the fault coverage for such big circuits.

**Effect of Seeds on the test-length**

For smaller circuits, although the choice of seed does not affect the fault coverage, it can be seen in Table 6.12 that the seed value significantly affect the test length. For example, the result for the circuit c880, indicate that the seed values do not affect the fault coverage (99.94%), but the difference between the maximum and the minimum test length was more than sevenfold (from 8,192 to 59,392).

Moreover, the *difference* between the *minimum* test length and the *mean* test length for most of the circuits can be seen in Table 6.12 to be *significant*. Hence, the choice of seed value is very important in keeping the test-length short.

Therefore, after identifying suitable connections between SRLs and the inputs of the CUT, to give a high fault coverage, it is essential to search for a suitable seed value to keep the test length short.

The advantage of having a short test-length in BIST is that the self-test can then be performed more frequently. The more frequently a circuit is tested, the higher is the probability of detecting *intermittent* faults and also the lower is the probability of more than one fault appearing in the circuit between two testing applications.

## 6.4 Summary

In this chapter, the simulation results for various methods, that try to enhance the *transition fault* coverage, were presented and compared. The results suggest that our method, which in addition to overcoming the disadvantages of other methods, is very effective in enhancing *transition fault* coverage.

The choice of *seed value* was identified to have very little or no effect on the *transition fault* coverage for most circuits. But, the variation in the *test length* was significant.

# Chapter 7

# Conclusions And Future Directions

In this thesis, an algorithm was developed that attempts to separate the inputs using a fixed number of shift register latches which is equal to (or in special cases less than) the number of inputs to the circuit under test. This algorithm was applied to the ISCAS '85 benchmark circuits and the simulation results suggest that the algorithm is very effective in enhancing transition fault coverage.

The advantage of using the algorithm over the *cross over permutation* (XLFSR) method is that the algorithm generates the connections between the outputs of the test-pattern generator and the circuit inputs based on the structure of the circuit under test. On the other hand, the *cross over permutation* is fixed and therefore does not work well for all circuits. This is especially true for circuits for which the inputs to the logic cones are not adjacent inputs of the circuit under test. Moreover, the simulation results for most of the ISCAS '85 circuits using our algorithm to separate the inputs were better than those for the XLFSR.

The advantage of using our algorithm over the ISUD method is that the number of SRLs, in the algorithm, is fixed to the number of inputs in the circuit. With our implementation of the ISUD method, the number of SRLs required to separate the inputs

of the ISCAS'85 circuits exceeds the number of inputs for all the circuits. The c5315 ISCAS '85 circuit is an *ALU and Selector* with 178 inputs and 123 outputs. The ISUD method requires 179 (one *more* than the number of inputs) SRLs to separate the inputs. On the other hand, our algorithm with option '201' requires only 133 (45 *less* than the number of inputs) SRLs to separate all the inputs that belong to the same cone. The simulation results for ISCAS '85 circuits indicate that our method performs as well as or, in some cases, better than the ISUD method.

Another contribution in this thesis was to identify the *effect of seed* values on the test-length. It was observed that for most circuits the seed value had very little or no effect on the fault coverage, but the variation in the test-length was significant. This is important because if the testing time is short, then the test can be repeated more frequently. The advantage of frequent testing is the high probability of detecting *intermittent* faults.

It was also pointed out that when the LFSR is used as the source of test-stimuli, the choice of *orientation* of a LFSR can affect the *transition fault* coverage for a given circuit under test. This is important because, although the two *orientations* are in fact two permutations of the connections between the LFSR and the inputs to the CUT, the routing overhead for one orientation (or set of connections) may be less than the other.

**Suggestions for Future Work**

As a result of our research work, there are some interesting problems to be investigated. These problems are:

- The algorithm introduced in this thesis uses a number of simple rules to make decisions. The question is can the algorithm be further improved to get better results by adding more rules or modifying the existing ones.

- An alternative method, the *ISSIA* method, was proposed but was not implemented. It will be of interest to see how the results of this method will compare with the results which we already have for the *ISSLA* and the *ISASLA* methods.

- The algorithm uses a set of simple rules to dictate the connections between the generator outputs and the inputs of the circuit under test (CUT). These connections

have been found to give high fault coverage. The problem to investigate is, given some random connections, is it possible to analyze the effect of these connections on the CUT and *predict* whether a high *transition fault* coverage is achievable or not. This will have an advantage over the time consuming simulation runs.

- The connections generated by the algorithm will increase the routing complexity and the following two problems need to be investigated;

  1. How much is the overhead?
  2. When a single SRL is allowed to feed multiple inputs, then how serious is the effect on the test speed due to increased load capacitance?

- The seed value was found to have a significant effect on the test length. The problem to investigate is how to search for seed values to keep the test length short.

Research on the above problems can further enhance the *transition fault* testing techniques without much increase in their implementation costs.

# Bibliography

[ABF90] M. Abramovici, M. A. Breuer, and A. D. Friedman. *Digital Systems Testing and Testable Deisgn.* Computer Science Press., 1990.

[BF85] F. Brglez and H. Fujiwara. Neutral netlist of ten combinational benchmark circuits and a target translator in FORTRAN. In *Proc., IEEE International Symposium. Circuits and Systems.*, June 1985.

[BMS87] P. H. Bardell, W. H. McAnney, and J. Savir. *Built-In Test for VLSI: Pseudorandom Techniques.* John Wiley & Sons, Inc., 1987.

[BR83] Z. Barzilai and B. K. Rosen. Comparison of AC Self-Testing Procedures. In *Proceedings International Test Conference*, pages 89–94, 1983.

[GJ79] Michael R. Garey and David S. Johnson. *COMPUTERS AND INTRACTABILITY, A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company., 1979.

[IRW90] V. S. Iyengar, B. K. Rosen, and S. A. Waicukauski. On Computing the Sizes Of Detected Delay Faults. *IEEE Transaction on CAD*, pages 299–312, March 1990.

[Joh89] Barry W. Johnson. *Design and Analysis of Fault Tolerant Digital Systems.* Addison-Wesley Publishing Company, 1989.

[Mil87] D. M. Miller. *Developments in Integrated Circuit Testing.* Academic Press Limited., 1987.

[PS92] S. Patil and J. Savir. Skewed-Load Transition Test: Part II, Coverage. In *Proceedings International Test Conference*, pages 714–722, October 1992.

[PUWM91] E. S. Park, B. Underwood, T. W. Williams, and M. R. Mercer. Delay Testing Qualities In Timing Optimized Designs. In *Proceedings International Test Conference*, pages 897–905, Oct 1991.

[RLP87] S. M. Reddy, C. J. Lin, and S. Patil. An Automatic Test Pattern Generator For Detection of Path Delay Faults. *ITC Computer Aided design.*, pages 284–287, November 1987.

[Sav92] J. Savir. Skewed-Load Transition Test: Part I, Calculus. In *Proceedings International Test Conference*, pages 705–713, October 1992.

[SM86] J. Savir and W. H. McAnney. Random Pattern testability Of Delay Faults. In *Proceedings International Test Conference*, pages 263–273, 1986.

[Smi85] Gordon L. Smith. Model For Delay Faults Based Upon Paths. In *Proceedings International Test Conference*, pages 342–349, November 1985.

[SR92] Jacob Savir and Berry Robert. AC Strength of a Pattern Generator. *JOURNAL OF ELECTRONIC TESTING : Theory and Applications*, 3:119–125, May 1992.

[SSMM90] M. Serra, T. Slater, J. C. Muzio, and D. M. Miller. The Analysis of one-dimensional Cellular Automata and their aliasing properties. *IEEE Transaction on CAD*, 9(7):767–778, July 1990.

[ZBM92] S. Zhang, R. Byrne, and D. M. Miller. LFSR and LHCA Generators and Nontraditional Fault Models. In *Proceedings of the 6th Workshop on new directions for testing*, pages 205–217, May 1992.

[ZM90] S. Zhang and D. M. Miller. A comparison of LFSR and cellular automata BIST. In *Proceedings of the Canadian Conference on VLSI*, pages 8.4.1–8.4.9, October 1990.

# VITA

| Surname: Kadri | Given Names: Fayaz |
| --- | --- |
| Place of Birth: Dar-Es-Salaam, Tanzania | Date of Birth: Jan 04, 1967 |

Educational Institutions Attended:

Middle East Technical University
Ankara, Turkey — 1987 to 1991

Degrees Awarded:

B.Sc. with High Honours
Middle East Technical University — July 1991

Honours and Awards:

| Award | Dates |
| --- | --- |
| University of Victoria Teaching Fellowship Award | Sept. 1992 - Apr. 1993 |
| University of Victoria Teaching Fellowship Award | Sept. 1991 - Apr. 1992 |
| Islamic Development Bank Scholarship (I.D.B. Jeddah, S. Arabia) | 1987 to 1991. |

# PARTIAL COPYRIGHT LICENSE

I hereby grant the right to lend my thesis to users of the University of Victoria Library, and to make single copies only for such users or in response to a request from the Library of any other university, or similar institution, on its behalf or for one of its users. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by me or a member of the University designated by me. It is understood that copying of this thesis for financial gain shall not be allowed without my written permission.

Title of Thesis:

Enhancing Transition Fault Coverage
in Built-In Self-Test

Author:

Fayaz M. Kadri

April 30, 1993