INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI

A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor MI 48106-1346 USA
313/761-4700 800/521-0600

Evaluation in Built-In Self-Test

by

Shujian Zhang

M.Sc., University of Victoria, 1993

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

DOCTOR OF PHILOSOPHY

in the Department of Computer Science

We accept this dissertation as conforming
to the required standard

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

Dr. J.C. Muzio, Co-Supervisor (Department of Computer Science)

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

Dr. G.F. McLean, Outside Member (Department of Mechanical Engineering)

Dr. A. Ivanov, External Examiner (Department of Electrical and Computer Engineering, University of British Columbia)

©Shujian Zhang, 1998

University of Victoria

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

Co-Supervisors: Dr. D.M. Miller and Dr. J.C. Muzio

Abstract

This dissertation addresses two major issues associated with a built-in self-test environment: (1) how to measure whether a given test vector generator is suitable for testing faults with sequential behavior, and (2) how to measure the safety of self-checking circuits.

Measuring the two-vector transition capability for a given test vector generator is a key to the selection of the generators for stimulating sequential faults. The dissertation studies general properties for the transitions and presents a novel, comprehensive analysis for the linear feedback shift registers and the linear hybrid cellular automata. As a result, the analysis solves the open problem as to "how to properly separate the inputs when the LHCA-based generator is used for detecting delay faults".

In general, a self-checking circuit has additional hardware redundancy than the original circuit and as a result, the self-checking circuit may have a higher failure rate than the original one. The dissertation proposes a fail-safe evaluation to predict the probability of the circuit not being in the fail-state. Compared with existing evaluation methods, the fail-safe evaluation is more practical because it estimates the safety of the circuit, which is decreasing as time goes on, instead of giving a constant probability measure.

Various other results about improving fault coverage for transition delay faults and testing in macro-based combinational circuits are derived as well.

Examiners:

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

Dr. J.C. Muzio, Co-Supervisor (Department of Computer Science)

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

Dr. G.F. McLean, Outside Member (Department of Mechanical Engineering)

Dr. A. Ivanov, External Examiner (Department of Electrical and Computer Engineering, University of British Columbia)

Contents

Abstract ii

Contents iv

List of Tables vii

List of Figures viii

Acknowledgements ix

1 Introduction 1

1.1 Motivation 1

1.2 Outline of the Dissertation 7

2 Linear Finite State Machines 9

2.1 Introduction 9

2.2 Linear Finite State Machines 11

2.2.1 Linear Feedback Shift Register 11

2.2.2 Linear Hybrid Cellular Automata 13

2.2.3 Remarks 14

2.3 General Properties for Transitions 16

2.4 Algorithms for Determining Partner Sets 25

3 Transitions of LHCA and LFSR **28**
3.1 Transitions of LHCA 28
3.2 Transitions of LFSR(I) 30
3.3 Transitions of LFSR(II) 32
3.4 Summary 34

4 Transition Faults in Combinational Circuits **38**
4.1 Introduction 38
4.2 Transition (Gate Delay) Fault Model 39
4.3 Experimental Results 41
4.4 Discussion and Conclusion 44

5 Transition Faults in Look-up Table Based FPGA Circuits **46**
5.1 Introduction 46
5.2 Transition Fault Model 48
5.3 Technology Mapping 50
5.3.1 Problem Formalization 51
5.3.2 4-RLMP/3-RLMP and 3-SAT 52
5.3.3 Remarks 56
5.3.4 Technology Mapping Experiments 56
5.4 Experimental Simulation Results 57
5.5 Detection Requirements 60
5.6 Summary 62

6 Performance Evaluation of Self-Checking Circuits **63**
6.1 Introduction 64
6.2 Totally Self-Checking Goal 65
6.2.1 Preliminaries 65
6.2.2 Definitions 66
6.2.3 General Designs for TSC Circuits 67
6.3 Existing Evaluations of Self-Checking Circuits 71
6.3.1 Related Work 71
6.3.2 Remarks 73

7 Fail-Safe Evaluation for Self-Checking Circuits **75**

7.1 Fail-Safe Evaluation 75

7.1.1 Generic Markov Model 76

7.1.2 Simplified Markov Model 78

7.1.3 Generalizations of Parameters and Model 83

7.2 Experimental Results and Comparisons 85

7.2.1 Results 85

7.2.2 Remarks 86

7.3 Summary 88

8 Conclusions **90**

8.1 Summary 90

8.1.1 Major Contributions 90

8.1.2 Remarks 91

8.2 Further Work 91

Bibliography **93**

Appendix **101**

A Minimum Cost Maximum Length LHCA for $n \leq 500$ 101

List of Tables

2.1 Test vectors produced by the LFSR(I), LFSR(II), and LHCA 15
2.2 An example for the number of the transitions. 18
2.3 An example of selecting cells and partners. 25

3.1 Number of selections for k-cell substate with 2^{2k} transitions for odd n. 36
3.2 Number of selections for k-cell substate with 2^{2k} transitions for even n. 37

4.1 A summary of *transition* fault simulation results for 100 different random connections. 43

5.1 An example of transition faults for a 3-input LUT. 50
5.2 Numbers of LUTs and numbers of faults for ISCAS85 circuits. 57
5.3 A summary of the fault simulation results for 100 randomly chosen connections. 59

6.1 Detection capability for the two-rail checker in Figure 6.2. 69
6.2 Truth table for the BCD-to-excess-3 code conversion. 71

List of Figures

2.1 (a) LFSR(I) and (b) LFSR(II) with characteristic polynomial x^5+x^2+1. 12

2.2 A 5-cell LHCA with characteristic polynomial x^5+x^2+1. 13

2.3 An example for the maximal matching and partner set. 22

2.4 An algorithm for determining a partner set $ps(\mathbf{w})$ for an LHCA. . . . 26

2.5 An algorithm for determining a partner set $ps(\mathbf{w})$ for an LFSR(II). . 27

3.1 A simplified structure of LHCA. 28

3.2 A simplified structure of LFSR(I). 30

3.3 A simplified structure of LFSR(II). 33

4.1 Simplified example for detecting transition fault. 41

5.1 An example of the construction for 4-RLMP from 3-SAT. 54

5.2 An example of the construction for 3-RLMP from 3-SAT. 55

5.3 An example of the detection requirement for the transition fault. . . . 60

6.1 A totally self-checking circuit. 68

6.2 A TSC two-rail code checker. 68

6.3 A BCD-to-excess-3 circuit. 70

6.4 A design for a self-checking BCD-to-excess-3 circuit. 72

7.1 A generic Markov model for the self-checking circuit in Figure 6.4. . . 76

7.2 A simplified Markov model for the self-checking circuit in Figure 6.4. 78

7.3 A simplified Markov model for the self-checking circuit in general. . . 83

7.4 Safety and reliability for the circuit in Figure 6.4 with $\lambda_C = \lambda_T = 10^{-5}$. 87

Acknowledgements

I would like to express my appreciation to my supervisors, Dr. D.M. Miller and Dr. J.C. Muzio, for their encouragement, invaluable guidance, and constructive discussions throughout my graduate studies at Victoria.

I would like to thank my committee and my external examiner, Dr. A. Ivanov, for their comments and suggestions.

Thanks to all my friends both at Victoria and at Ottawa for their invaluable friendship and profound concern for me.

I am greatly indebted to my parents for their unending support, understanding, encouragement, and immense love. I would like to express my gratitude to my sisters, Meiling, Meiqi, Meijue, and Meiyu, for their consistent encouragement and support, especially for taking care of our parents when I am studying abroad.

Finally, I would like to thank my lovely wife, Lan, for her support and encouragement.

The work was supported in part by a Postgraduate Scholarship from the Natural Sciences and Engineering Research Council of Canada and Scholarships from the University of Victoria.

Chapter 1

Introduction

1.1 Motivation

During its lifetime, a digital system is tested and diagnosed on numerous occasions. For the system to perform its intended mission with high availability, both testing and diagnosis must be performed quickly and effectively[AKS93].

The primary objective of testing digital circuits at the chip, board, or system level is to detect the presence of hardware failures induced by faults in the manufacturing processes or by operating stress or wear-out mechanisms. There is nothing fundamentally difficult in the functional testing of digital circuits. All that is necessary is to apply some input test vectors and observe the resulting digital output signals. The problem, basically, is the volume of data and resultant time to test. For circuits with a small number of logic gates, fully-exhaustive functional testing may be possible. As circuit size increases, techniques to ease this problem become increasingly necessary [Hur98].

Built-in self-test (BIST), *i.e.*, self-test implemented in the hardware itself, is a general approach to test a digital system [ABF90]. A widely accepted approach to BIST is to use a *pseudo-random* vector generator and a data compactor. The

generator produces the test vectors applied to a circuit under test and the compactor reduces the response to these vectors to a single value (*e.g.*, 16 or 32 bits) known as a signature.

In general, LFSMs (Linear Finite State Machines) [Sto73], such as an LFSR (Linear Feedback Shift Register) [BMS87] and an LHCA (Linear Hybrid Cellular Automata) [HMP+89, HMC89, SSMM90, ZBM92] are used as BIST test vector generators. Note that it is not practical to apply all $2^n - 1$ vectors to an n-input circuit with n over 25 when using an n-cell generator because it takes too much time. Therefore, minimizing the number of test vectors applied to the circuit for detecting all possible faults is one of the major concerns.

On the other hand, an important problem one faces during the design of a BIST circuit is not only to detect stuck-at faults but also to detect delay faults, normally due to random variation in process parameters that often cause propagation delays to exceed their limits. That is, we need to consider the effectiveness of the test vector generator for detecting faults with *sequential* behavior in the circuit under test.

A general approach for measuring the quality of a generator is based on fault simulation to see how many faults considered can be detected when a test vector sequence, produced by the generator, is applied to the circuit. Simulation is a useful approach for comparing the performance of different BIST generators for different fault models. However, without analyzing the fault models considered and the properties of the generators, it is not easy to draw any general conclusion.

In a BIST environment, a checker, which compares two values to see whether they are the same, is required. For example, the signature generated via the test should be checked to see if it is correct. Self-checking techniques should be applied to such a checker, so that any fault in the checker can be detected as well. In general, self-checking circuits are tested during normal operation. Therefore, evaluating the

quality of the self-checking circuit should be under the normal operation of the circuit.

Although a considerable amount of work has been done regarding performance evaluation and improvement in BIST, there still exist gaps between models (or methodologies), used in the evaluation and the performance improvement, and the realities of testing in circuits. Consequently, researchers are continuously working in this area in order to expose new approaches which are more reliable in practice. This dissertation addresses the following issues about the evaluation and the performance improvement in BIST.

(a) Quantitative Measures of Test Vector Generators

In a BIST environment, choosing a test vector generator for stimulating defects in a circuit under test is a complex issue. Since it is not easy to abstract fault models from physical defects, researchers use stuck-at faults as a general fault model to measure the effectiveness of the stimulation source used. However, there are serious limitations in predicting defect levels based on stuck-at fault coverage [PNK+94]. Faults with sequential behavior, such as delay faults, are an increasingly frequent problem with modern circuits and have been subject to considerable study.

We place emphasis on measuring the effectiveness of a generator for stimulating delay faults and are particularly interested in attempting to develop a sound theoretical approach to compare generators which are suitable for testing sequential type faults (such as delay faults). Such faults require a pair of vectors for stimulation, the first to *set-up* the fault, and the second to *propagate* the fault to a circuit output.

A comprehensive description of the development of delay testing is given in [Sav92] and an approach to the selection of a proper test vector genera-

tor to apply to a circuit under test is proposed in [Sav95, Sav97]. It is noted that the two-vector testing capabilities for the generators are playing a key role in testing the delay faults.

A method for assessing the two-vector testing capabilities of linear test vector generator circuits is given in [FM91], where transition coverage is introduced as a metric. In [ZBM92, Zha93], an analysis of transition coverage is presented, which derives the exact number of distinct transitions for a linear feedback shift register (LFSR), a linear hybrid cellular automata (LHCA), and two modified generators, XLFSR and XLHCA, for a number of specific substate vectors. For a $2n$-cell LHCA, an approach is given in [NVCC94] to select an n-cell substate vector such that the corresponding vector sequence has 2^{2n} transitions.

Our goal is to concentrate on any k-cell substate vector and evaluate the number of distinct k-cell substate vectors which produce the maximum number 2^{2k}, $1 \leq k \leq \lfloor n/2 \rfloor$, of distinct transitions for any n-cell LHCA and LFSR with maximum length cycles. We derive a general theoretical answer to the question as to why the LHCA are better than the LFSR as BIST generators for sequential faults.

We develop an approach to calculate the number of transitions for a given substate vector for an LFSR or LHCA. The approach is efficient because it directly examines the feedback for each cell in the substate vector instead of evaluating the rank of the corresponding submatrix. In such a way, we can derive the numbers of distinct k-cell substate vectors that have 2^{2k} transitions, which gives an indication as to how good the test vector generator is for detecting sequential faults. As a result, the analysis solves the open problem of "how to properly separate the inputs when an LHCA-based generator is used for detecting delay faults", proposed in [Sav95, Sav97].

(b) Improving Fault Coverage for Transition Delay Faults

For a given n-input circuit under test, we need to apply a test sequence produced by an n-cell test vector generator for detecting *sequential faults*. In general, we have $n!$ different ways to assign connections between the circuit input and the generator output. An interesting open question is proposed in [ZBM92, Zha93] as to whether a proper connection can be identified based on specific knowledge of the circuit topology or the functionality of the circuit on knowing the *sequential faults* considered and the transition property for a BIST vector generator used, so that the fault coverage is maximized.

Considerable work toward improving stuck-at fault coverage has been presented recently (*e.g.*, [Avr94, HK93, KT94, LGB94, MMR94, TM94, HRT+95, MV95]). However, not much attention is being paid to the improvement of fault coverage for sequential faults. One of the major reasons is that the latter problem is much harder than the former. For example, if all transition delay faults are detected, then all stuck-at faults are covered automatically.

To meet the needs of current application requirements, however, we attempt to derive the best possible result using limited computer resources and specific knowledge about the problem. We present our results based on random connections between the circuits and test vector generators, which look promising.

(c) Testing in Macro-Based Combinational Circuits

Macro-based combinational circuits considered are those constructed by look-up table (LUT) based field-programmable gate arrays (FPGA). It is mentioned in [Tri92] that FPGAs can be fully tested after manufacture, so users' designs do not require test program generation, automatic test pattern generation, and design for testability. In fact, the possibility still exists that the circuit may not work properly, resulting from component defects, *e.g.*, component wear-

out. Therefore, in [PR94], the problem of testing delay faults in macro-based combinational circuits is considered and two delay fault models are proposed.

Macro-based circuits are obtained as a result of technology mapping. It is instructive to apply logic synthesis techniques to the whole design in which the testing issues are included. Considering the key issues, such as circuit representation, technology mapping, and testing techniques used, we intend to analyze the problems deeply and examine a general testing strategy for macro-based circuits.

(d) Safety of Self-Checking Circuits

Self-checking circuits can detect the presence of transient and permanent faults because they are designed with additional sub-circuits that are used to monitor whether the circuits work correctly during normal operation. However, self-checking circuits may not necessarily be safe because some faults in the circuits may not necessarily be detected. Therefore, we need a proper evaluation for such circuits.

A primary difficulty with self-checking circuits is that, while both a circuit and a checker may be totally self-checking, the resulting composite circuit is not necessarily totally self-checking. That is, it is possible for there to be erroneous outputs from the circuit which are not caught by the checker. This is caused by the presence of one or more faults in the checker which have not been exposed by an appropriate stimulus from the circuit. The problem is that one of the two primary assumptions for totally self-checking circuits (*i.e.*, faults occur one at a time, and the time interval between occurrences of any two faults is long enough for all input codewords to be applied to the circuit) is often not met during normal operation of the circuit.

The design of self-checking circuits has been discussed for more than twenty

years. However, comparatively little has been done on the analytical evaluation of their performance. The existing evaluations are based on determining whether a given circuit satisfies the totally self-checking goal or calculating how much of the goal has been achieved by the given circuit [LM84, FMM84, FM87, LF93].

Consider the evaluation of the safety of a self-checking circuit with combinational logic. Since the circuit is tested under normal operation, as time goes on, it may be in a different state from a perfect state in which any erroneous output can be detected. It could be in an unstable state in which an erroneous output may be detected or may not, a safe-state when the erroneous output has been caught, or a fail-state because the erroneous output is undetected. Consequently, we propose a fail-safe evaluation, using a Markov model to describe the state transitions and predict the probability of the circuit not being in the fail-state.

1.2 Outline of the Dissertation

The chapters of this dissertation cover general properties for transitions for linear finite state machines (Chapter 2), analysis of the transition property for linear feedback shift register and linear hybrid cellular automata (Chapter 3), transition faults in combinational circuits (Chapter 4), transition faults in look-up table based FPGA circuits (Chapter 5), the safety of self-checking circuits (Chapter 6 and Chapter 7), and general conclusions and further work (Chapter 8).

Chapter 2 provides knowledge of linear finite state machines and their general properties for transitions.

Chapter 3 presents a method of evaluating the effectiveness of the LHCA and LFSR as test vector generators for stimulating faults requiring a pair of vectors.

Chapter 4 reviews the transition fault model and detection requirements. Then, it provides empirical comparisons to show that the analysis of the transition property presented in Chapter 3 is a reasonable metric of the effectiveness of the test vector generator in testing delay faults.

Chapter 5 examines the testing of look-up table based FPGA circuits, and proposes the transition fault models and detection requirements for such circuits. It also discusses the technology mapping problem and proves the NP completeness of the K-RLMP (Restricted K-LUT Minimization Problem) problem for $K = 3$ and 4.

Chapter 6 examines performance evaluation of self-checking circuits, reviews the existing methods, and discusses potential problems with them.

Chapter 7 develops a new approach to evaluate the safety of self-checking circuits. Compared with existing evaluation methods, the proposed approach is more practical because it estimates the safety of the circuits, which is decreasing as time goes on, instead of giving a constant probability measure.

Chapter 8 summarizes the major contributions of the dissertation and raises several related problems as further work.

Chapter 2

Linear Finite State Machines

This chapter presents a combinatorial method of evaluating the effectiveness of linear hybrid cellular automata (LHCA) and linear feedback shift registers (LFSR) as test vector generators for stimulating faults requiring a pair of vectors. We provide a theoretical analysis and empirical comparisons to see why the LHCA are better than the LFSRs as the generators for sequential-type faults in a built-in self-test (BIST) environment. Based on the concept of a partner set. the method derives the number of distinct k-cell substate vectors which have 2^{2k}. $1 \leq k \leq \lfloor n/2 \rfloor$. transition capability for an n-cell LHCA and an n-cell LFSR with maximum length cycles.

2.1 Introduction

In this chapter, we place emphasis on measuring the effectiveness of a generator for stimulating faults with *sequential* behavior. *e.g.*, *delay* faults. in a combinational circuit. since those faults are an increasingly serious problem with modern circuits and have been subject to considerable study. Usually linear finite state machines (LFSM). *e.g.*, linear feedback shift registers (LFSR) [BMS87] or linear hybrid cellular automata (LHCA) [HMP+89. HMC89, SSMM90, ZBM92], are used as BIST

generators. Such a generator is sequenced through a number of states with each state serving as a test vector. The coverage of combinational faults, such as *stuck-at* faults, depends entirely on the inclusion of the appropriate vectors within the sequence generated. It is well known that the performance of LFSM in this context is determined only by the characteristic polynomial of the LFSM, and actual implementation (whether it is as an LFSR, an LHCA or some others) is largely irrelevant.

On the other hand, the question of generators for stimulating *sequential* faults is much more interesting. We are particularly interested in attempting to develop a sound theoretical metric for comparing generators which are suitable for testing faults with *sequential* behavior, *e.g.*, *delay* faults, which require a pair of vectors for stimulation, the first to *set-up* the fault, and the second to *propagate* the fault to a circuit output. The effectiveness of a particular test vector generator in stimulating *sequential* faults is dependent upon its state transition sequence, *i.e.*, the number of distinct pairs of vectors produced by the generator.

A method of assessing the two-vector testing capabilities of linear test vector generator circuits is given in [FM91], where transition coverage is introduced as a metric. In [ZBM92], an analysis of transition coverage is presented, which derives the exact number of distinct transitions for the LFSR, LHCA, and two modified generators, XLFSR and XLHCA, for a number of specific substate vectors. For a $2n$-cell LHCA, an approach is given in [NVCC94] to select an n-cell substate vector such that the corresponding vector sequence has 2^{2n} transitions.

We concentrate on any k-cell substate vector and evaluate the number of distinct k-cell substate vectors which produce the maximum number 2^{2k}, $1 \leq k \leq \lfloor n/2 \rfloor$, distinct transitions for any n-cell LHCA and LFSR with maximum length cycles and give a general theoretical answer to the question as to why LHCA are better than the LFSRs as BIST generators for sequential faults.

The rest of the chapter is organized as follows. Section 2.2 reviews LFSR and LHCA. Section 2.3 introduces general definitions for transitions and their properties. In Section 2.4. we present algorithms of determining the partner sets.

In the next chapter. we give details of derivation for the number of different k-cell substate vectors which have 2^{2k} transition capability for the LHCA and LFSR and claim that LHCA have a much higher transition space than LFSR. and summarize our major results.

2.2 Linear Finite State Machines

Linear finite state machines (LFSM) that are of interest are called autonomous linear machines [Sto73]. *i.e.*. they have no inputs. In the remainder of this chapter. LFSM refers to an autonomous linear machine. In general. the next state function of an LFSM is represented by a state transition matrix T. For an n-cell state vector $\mathbf{s}$. the next state vector $\mathbf{s}^+$ is given by $\mathbf{s}^+ = \mathbf{s} \cdot T$. where all operations are carried out over $GF(2)$. *the Galois Field of order 2* [Sto73].

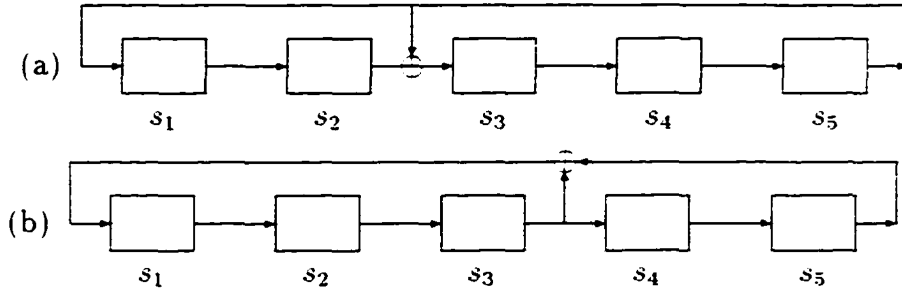
In this section. we briefly review practical linear finite state machines. *viz.*. linear feedback shift registers and linear hybrid one-dimensional cellular automata. Since they are specific instances of LFSM. they have all the general properties of LFSM. The only difference is that they have different state transition matrices which are described as follows.

2.2.1 Linear Feedback Shift Register

A linear feedback shift register (LFSR) [BMS87] is an LFSM. Each state is uniquely determined from the previous state. There are two configurations for an LFSR: a Type I LFSR which has the exclusive-OR gates between the cells. and a Type II LFSR which has exclusive-OR gates on the feedback path. We assume that shifting

and numbering of the cells are from left to right. For convenience. the Type I and Type II LFSRs are simply named LFSR(I) and LFSR(II). respectively.

Figure 2.1 (a) LFSR(I) and (b) LFSR(II) with characteristic polynomial x^5+x^2+1.



Example 2.1 *Figure 2.1 shows the two types of LFSRs derived from the same polynomial $x^5 + x^2 + 1$. We can write a set of state transition equations for any LFSR. For example. the equations for the LFSR(I) of Figure 2.1(a) are:*

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

where s_i is the present state of the cell i and s_i^+ is the next state with all operations being carried out over $GF(2)$. Hence

$$(s_1^+ . s_2^+ . s_3^+ . s_4^+ . s_5^+) = (s_1 . s_2 . s_3 . s_4 . s_5) \cdot T_I.$$

where T_I is the state transition matrix for the LFSR(I) of Figure 2.1(a) as follows:

$$T_I = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \end{pmatrix}.$$

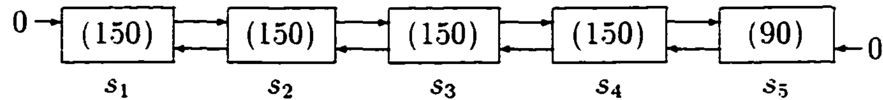
Similarly, the state transition matrix for the LFSR(II) of Figure 2.1(b) is given by

$$T_{II} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix}.$$ □

2.2.2 Linear Hybrid Cellular Automata

The linear hybrid cellular automata (LHCA) considered are LFSM each composed of a one-dimensional array of cells, which only communicate with their immediate neighbors [SSMM90]. Figure 2.2 is an example of an LHCA. The LHCA is said to have *null boundary conditions* since the two end connections are fixed at 0.

Figure 2.2 A 5-cell LHCA with characteristic polynomial $x^5 + x^2 + 1$.



Definition 2.1 *Let s_i, $1 \leq i \leq n$, be the present state of cell i of the LHCA identified by the n-tuple $(c_1, c_2, \ldots, c_n)$. Its next state, s_i^+, is given by*

$$s_i^+ = s_{i-1} + c_i s_i + s_{i+1},$$

where $s_0 = s_{n+1} = 0$, $c_i = 0$ for a Rule 90 cell and $c_i = 1$ for a Rule 150 cell.

We only consider LHCA composed of *Rule* 90 and *Rule* 150 cells since it has been shown in [SSMM90] that this is a necessary condition for the LHCA to have maximum length cycle. These have been termed *hybrid* since the cells are not all the same. Throughout this chapter we are concerned with one-dimensional linear hybrid rule 90/150 cellular automata and we refer to them simply as LHCA for brevity. A general state transition matrix for the LHCA is given by

$$T = \begin{pmatrix} c_1 & 1 & 0 & \cdots & 0 \\ 1 & c_2 & 1 & & 0 \\ & \ddots & \ddots & \ddots & \\ 0 & & 1 & c_{n-1} & 1 \\ 0 & \cdots & 0 & 1 & c_n \end{pmatrix}. \tag{2.1}$$

For example, for the LHCA shown in Figure 2.2, the state transition matrix is

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

2.2.3 Remarks

When an LFSM is used as the generator in BIST, it is sequenced through a number of states with each state serving as a test vector. Table 2.1 shows the test vectors produced by the LFSR(I) and LFSR(II) of Figure 2.1, and LHCA of Figure 2.2. The test vectors produced cover all possible nonzero states, beginning

Table 2.1 Test vectors produced by the LFSR(I), LFSR(II), and LHCA

Time	*LFSR(I)*	*LFSR(II)*	*LHCA*	*Time*	*LFSR(I)*	*LFSR(II)*	*LHCA*
0	00001	00001	00001	16	00110	00111	10001
1	10100	10000	00010	17	00011	00011	11010
2	01010	01000	00111	18	10101	10001	00011
3	00101	00100	01011	19	11110	11000	00101
4	10110	10010	11001	20	01111	01100	01100
5	01011	01001	00110	21	10011	10110	10010
6	10001	10100	01001	22	11101	11011	11111
7	11100	11010	11110	23	11010	11101	01111
8	01110	01101	01101	24	01101	01110	10111
9	00111	00110	10000	25	10010	10111	10011
10	10111	10011	11000	26	01001	01011	11101
11	11111	11001	00100	27	10000	10101	01000
12	11011	11100	01110	28	01000	01010	11100
13	11001	11110	10101	29	00100	00101	01010
14	11000	11111	10100	30	00010	00010	11011
15	01100	01111	10110	31	00001	00001	00001

from a nonzero state. Thus, they are all maximum length cycles (with length $2^n - 1 = 2^5 - 1 = 31$). Moreover, since the LFSR(I), LFSR(II), and LHCA have the same primitive characteristic polynomial, they produce the same output stream in each bit position apart from the appropriate phase shift, flowing from each single cell of the generators, *e.g.*, starting at 0 marked for state s_1 in Table 2.1, we can see that three sequences on s_1 for the LFSR(I), LFSR(II) and LHCA are identical. The calculation of the phase-shift between the bit streams can be found in [Bar92a].

The LFSR and LHCA associated with primitive polynomials are more desirable because they generate all possible nonzero test vectors, beginning from any nonzero state. There has been much recent work concerning LHCA (*e.g.*, [SSMM90, SS90, DGD+90, CM91, Bar92a, CM96, Cat95]). In general, we minimize the hardware cost of an LFSR implementation by using minimal weight primitive polynomials. Such polynomials for the LFSR of degree through 500 are contained in [BMS87, Bar92b].

whereas the minimal cost LHCA of degree through 500 are available in [ZMM91, CZ95] (Also see Appendix A).

The algorithm of determining whether a given n-degree LHCA has maximum length cycle is as follows.

(**a**) Compute the characteristic polynomial of the LHCA using the recurrence relation in [SSMM90];

(**b**) Check if the characteristic polynomial is primitive; if so, the LHCA has maximum length cycle.

For each degree, we first generate all of the LHCA with a single rule 150 cell. If this is not successful, we then generate all of the LHCA with a pair of rule 150 cells. For each degree, the search is stopped at the first LHCA with maximum length cycle. This search has never failed, meaning that for each degree up to 500, there is an LHCA with maximum length cycle that has either one or two rule 150 cells. More details can be found in [ZMM91, CZ95].

2.3 General Properties for Transitions

For *sequential* faults, the actual test vector sequence produced by a generator is critical since a fault requires a pair of vectors for stimulation. We are thus concerned with the transitions produced by the generator. Moreover, as noted above, it is not sufficient to just consider transitions of n-cell vectors, but we must also look at transitions of k-cell substate vectors for $k < n$. We start by defining these k-cell substate vectors, and the corresponding transitions.

Definition 2.2 *For a given n-cell LFSM state vector* $\mathbf{s} = (s_1, s_2, \ldots, s_n)$, $s_p \in \{0, 1\}$, $1 \le p \le n$, *a k-cell substate vector* $\mathbf{w}$ *of* $\mathbf{s}$ *is defined by*

$$\mathbf{w} = (s_{i_1}, s_{i_2}, \ldots, s_{i_k}),$$

and a transition corresponding to $\mathbf{w}$ *is defined as*

$$\langle(s_{i_1}, s_{i_2}, \ldots, s_{i_k}), (s_{i_1}^{+}, s_{i_2}^{+}, \ldots, s_{i_k}^{+})\rangle,$$

where $1 \le i_j < i_l \le n$ *for* $1 \le j < l \le k$.

For notational convenience, $\overline{\mathbf{w}}$ is used to denote a substate vector of $\mathbf{s}$ with cells which are not in $\mathbf{w}$. We count one transition even if $(s_{i_1}, s_{i_2}, \ldots, s_{i_k}) = (s_{i_1}^{+}, s_{i_2}^{+}, \ldots, s_{i_k}^{+})$ because it simplifies the derivation of a general equation to evaluate the number of transitions for a given substate vector. That is, for a k-cell substate vector, we use $2^k \times 2^k$ as its maximum number of transitions, instead of $2^k \times (2^k - 1)$.

Example 2.2 *For the three LFSMs in Table 2.1, if* $\mathbf{s} = (s_1, s_2, s_3, s_4, s_5) = (00111)$ *and* $\mathbf{w} = (s_2, s_3, s_4)$, *then the transition corresponding to* $\mathbf{w}$ *is*

$$\langle(s_2, s_3, s_4), (s_2^{+}, s_3^{+}, s_4^{+})\rangle,$$

which is

$$\langle(011), (011)\rangle \qquad \text{for the } LFSR(I) \text{ because } s^{+} = (10111),$$

$$\langle(011), (001)\rangle \qquad \text{for the } LFSR(II) \text{ because } s^{+} = (00011),$$

$$\langle(011), (101)\rangle \qquad \text{for the } LHCA \text{ because } s^{+} = (01011).$$

For any particular substate vector, we can count the total number of transitions for the given substate vector for the LFSM. For example, for $\mathbf{w} = (s_3, s_4)$ *for the LFSR(I) in Table 2.1, we have the following transitions:* $\langle(00), (10)\rangle$, $\langle(10), (01)\rangle$, $\langle(01), (10)\rangle$, $\langle(10), (11)\rangle$, $\langle(11), (01)\rangle$, $\langle(01), (00)\rangle$, $\langle(11), (11)\rangle$, $\langle(00), (00)\rangle$, *giving a total of 8 transitions. Obviously, the maximum number of transitions in this case is 16 (four possible choices for the first vector of the pair, and four for the second), so this generator only produces half of the maximum possible number of transitions for this substate. The complete list of all the transitions for 2-cell substate vectors for the LFSMs from Table 2.1 is given in Table 2.2.* □

Table 2.2 An example for the number of the transitions.

Substate Vector	LFSR(I)	LFSR(II)	LHCA
(s_1, s_2)	8	8	8
(s_1, s_3)	16	16	16
(s_1, s_4)	16	16	16
(s_1, s_5)	8	16	16
(s_2, s_3)	16	8	16
(s_2, s_4)	16	16	16
(s_2, s_5)	16	16	16
(s_3, s_4)	8	8	16
(s_3, s_5)	16	16	16
(s_4, s_5)	8	8	8

As shown in Table 2.2, if we consider different k-cell substate vectors, there are some differences between the numbers of distinct transitions generated by the LHCA and the LFSR. Here we briefly review the method, based on evaluating the rank of a specific submatrix of the transition matrix, in [FM91] and propose a different approach to select a k-cell substate vector $\mathbf{w}$ which can achieve 2^{2k} distinct transitions for $1 \leq k \leq \lfloor n/2 \rfloor$. First we give an example using the transition matrix to evaluate the number of distinct transitions.

Example 2.3 *Let* $\mathbf{s} = (s_1, s_2, s_3, s_4, s_5)$ *be the state vector. We want to know the number of distinct transitions for a 2-cell substate vector* $\mathbf{w} = (s_1, s_3)$*. By definitions, we let* $\overline{\mathbf{w}} = (s_2, s_4, s_5)$ *and* $\mathbf{w}^+ = (s_1^+, s_3^+)$*. For the LFSR(II) shown in*

Figure 2.1(b), we have the following next state function for $\mathbf{w}$

$$
\begin{aligned}
(s_1^+, s_3^+) &= (s_1, s_2, s_3, s_4, s_5) \cdot \begin{pmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 0 \\ 1 & 0 \end{pmatrix} \\
&= (s_1, s_3) \cdot \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} + (s_2, s_4, s_5) \cdot \begin{pmatrix} 0 & 1 \\ 0 & 0 \\ 1 & 0 \end{pmatrix}.
\end{aligned}
$$

In general, we have

$$\mathbf{w}^+ = \mathbf{w} \cdot T_w + \overline{\mathbf{w}} \cdot T_{\overline{w}}. \tag{2.2}$$

In particular,

$$rank(T_{\overline{w}}) = rank(\begin{pmatrix} 0 & 1 \\ 0 & 0 \\ 1 & 0 \end{pmatrix}) = 2.$$

Therefore, based on Theorem 1 in [FM91], for each value of (s_1, s_3), *there are* 2^2 *distinct next values* (s_1^+, s_3^+), *and the total number of distinct transitions for the given substate vector is* $2^2 2^2 = 2^4$. □

For convenience, Theorem 1 in [FM91] is stated as follows.

Theorem 2.1 [FM91] *For a k-cell substate vector* $\mathbf{w}$ *with next state function written in Eqn* (2.2), 2^r *distinct transitions occur from every k-cell substate vector, where* $r = rank(T_{\overline{w}})$, *while the whole state vector* $\mathbf{s}$ *goes through all possible* 2^n *states.*

Theorem 2.2 [ZBM92] **(upper and lower bounds)** *Consider any n-cell LFSM vector generator with a maximum length cycle. Let* $\mathcal{F}(k)$ *be the maximal number of*

distinct transitions corresponding to a k-cell substate vector $\mathbf{w}$. *We have*

$$\text{upper bound:} \quad \mathcal{F}(k) \leq \begin{cases} 2^{2k}, & 1 \leq k < \lceil n/2 \rceil, \\ 2^n - 1, & \lceil n/2 \rceil \leq k \leq n. \end{cases}$$

$$\text{lower bound:} \quad \mathcal{F}(k) \geq \begin{cases} 2^{k}, & 1 \leq k < n. \\ 2^n - 1, & k = n. \end{cases}$$

Based on the above theorem, it can be seen that for any k-cell substate vector $\mathbf{w}$ of an n-cell LFSM state vector $\mathbf{s}$ with $k = \lfloor n/2 \rfloor$, if n is odd, the maximum possible number of distinct transitions produced by $\mathbf{w}$ is 2^{2k}, and if n is even, the maximum possible number of distinct transitions produced by $\mathbf{w}$ is $2^{2k} - 1 = 2^n - 1$ because the all zeros state is not included in the sequence produced by the LFSM. Without loss of generality, when n is even and $k = \lfloor n/2 \rfloor$, we assume that $\mathbf{w}$ can produce 2^{2k} transitions in the best case, *i.e.*, the corresponding $T_{\overline{w}}$ has rank k.

It is proposed in [FM91] that transition coverage for $k = \lfloor n/2 \rfloor$ could be used as a universal metric of transition capability of an LFSM because of the following reasons. Assume a k-cell substate vector $\mathbf{w}$ of $\mathbf{s} = (s_1, s_2, \ldots, s_n)$ can produce 2^{2k} transitions. Then it follows that

(a) Any m-cell substate vector $\mathbf{x}$ of $\mathbf{w}$ can produce 2^{2m} transitions:

(b) Any m-cell substate vector $\mathbf{x}$, which includes $\mathbf{w}$, of $\mathbf{s}$ can produce at least 2^{2k} transitions.

In the next section, we derive the number of different k-cell substate vectors, which have 2^{2k} transition capability, with $k \leq \lfloor n/2 \rfloor$ for the LHCA and LFSR in order to compare the performance of the LHCA and LFSR concerning testing faults requiring a pair of vectors. A key to determining whether a given k-cell substate vector $\mathbf{w}$ has 2^{2k} transition capability is to check if the corresponding $T_{\overline{w}}$ has rank k. However, to avoid computing the rank of $T_{\overline{w}}$, we introduce the idea of a partner set and prove that the cardinality of this set gives us the rank of $T_{\overline{w}}$.

Definition 2.3 *Let* $\mathbf{w}$ *be a substate vector of a state vector* $\mathbf{s} = (s_1, s_2, \ldots, s_n)$ *for an n-cell LFSM. The next state* s_i^+ *corresponding to* s_i *in* $\mathbf{w}$ *is given by*

$$s_i^+ = \sum_{s_j \in K_i} s_j, \qquad \text{for some subset } K_i \subseteq \{s_1, s_2, \ldots, s_n\}.$$

(Note that the subset K_i *for a specific* s_i *depends on the transition matrix for the given LFSM.) All such* s_j, *which are not in* $\mathbf{w}$, *are eligible partners for* s_i.

Example 2.4 *For the LHCA shown in Figure 2.2, let* $\mathbf{w} = (s_2, s_4)$. *The next state functions for* s_2 *and* s_4 *are given by*

$$s_2^+ = s_1 + s_2 + s_3,$$
$$s_4^+ = s_3 + s_4 + s_5.$$

That is, $K_2 = \{s_1, s_2, s_3\}$ *and* $K_4 = \{s_3, s_4, s_5\}$. *By definition 2.3, the eligible partners are* s_1 *and* s_3 *for* s_2, *and* s_3 *and* s_5 *for* s_4. □

In other words, partners are those states that are not in $\mathbf{w}$, but have an immediate effect on the next state function corresponding to the state in $\mathbf{w}$.

Before defining a partner set, we review a maximal matching problem in a bipartite graph[CLR90]. Let $G = (V, E)$ be a bipartite graph with V partitioned as $X \cup Y$ (Each edge of E has the form (x, y) with $x \in X$ and $y \in Y$).

(a) A *matching* of G is a subset of E such that no two edges share a common vertex in X or Y.

(b) A *maximal matching* in G is one that matches as many vertices in X as possible with vertices in Y.

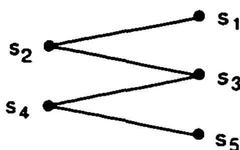
We use the concept of the maximal matching to define the partner set as follows.

Definition 2.4 *Let* $\mathbf{w}$ *be a k-cell substate vector of a state vector and* $G = (V, E)$ *be a bipartite graph with* V *partitioned as* $X \cup Y$, *where*

$$\begin{aligned} V &= \{s_1, s_2, \ldots, s_n\}, \\ X &= \{\ s_i \mid s_i \textit{ is in } \mathbf{w}\ \}, \\ Y &= \{\ s_j \mid s_j \textit{ is in } \overline{\mathbf{w}}\ \}, \\ E &= \{\ (s_i, s_j) \mid s_i \in X, s_j \in Y, \textit{ and } s_j \textit{ is an eligible partner of } s_i\ \}. \end{aligned}$$

If M, $M \subseteq E$, *is a maximal matching of* G, *then* $ps(\mathbf{w}) = \{\ s_j \mid (s_i, s_j) \in M\ \}$ *is a partner set of* $\mathbf{w}$ *and* $|ps(\mathbf{w})| = |M|$.

Figure 2.3 An example for the maximal matching and partner set.



Example 2.5 *For the* $\mathbf{w} = (s_2, s_4)$ *given in example 2.4, a corresponding bipartite graph* G *is shown in Figure 2.3 with* $X = \{s_2, s_4\}$, $Y = \{s_1, s_3, s_5\}$, *and* $E = \{(s_2, s_1), (s_2, s_3), (s_4, s_3), (s_4, s_5)\}$. *The possible maximal matchings for this example are* $\{(s_2, s_1), (s_4, s_3)\}$, $\{(s_2, s_1), (s_4, s_5)\}$, *and* $\{(s_2, s_3), (s_4, s_5)\}$. *Thus, the corresponding partner sets are* $\{s_1, s_3\}, \{s_1, s_5\}$, *and* $\{s_3, s_5\}$. □

Note that for a given $\mathbf{w}$, the choice of partners may not be unique, which may result in different partner sets.

Theorem 2.3 *Let* $\mathbf{w}$ *be a k-cell substate vector of a state vector* $\mathbf{s}$ *for an n-cell LHCA or LFSR with a maximum length cycle.* $|ps(\mathbf{w})| = k$ *if and only if* $rank(T_{\overline{w}}) = k$, i.e., $\mathbf{w}$ *can produce* 2^{2k} *distinct transitions.*

Proof. ($\Leftarrow$) It is obvious.

($\Rightarrow$) The trick of the proof is to find that k columns in $T_{\overline{w}}$ are linearly independent. Here, we show the key to the proof for the LHCA. For the LFSR, the proof is similar.

Consider an n-cell LHCA with an $n \times n$ state transition matrix T in Eqn (2.1) and a k-cell substate vector $\mathbf{w}$. For simplicity, if an element of T is one, we call it a 1-element; otherwise, a 0-element.

Because of the nature of the state transition matrix T of the LHCA, for the $(n-k) \times k$ submatrix $T_{\overline{w}}$ of T, we have the following:

(a) $T_{\overline{w}}$ is given by

$$T_{\overline{w}} = \begin{pmatrix} (T_{\overline{w}(1)}) & 0 & \cdots & 0 \\ 0 & (T_{\overline{w}(2)}) & & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & (T_{\overline{w}(u)}) \end{pmatrix}.$$

where $T_{\overline{w}(q)}$, $1 \le q \le u$, has k_q columns and $\sum_{q=1}^{u} k_q = k$;

(b) Each column/row in $T_{\overline{w}}$ has at most two 1-elements, which must be adjacent;

(c) If an element t_{ij}, $1 \le i \le n-k$ and $1 \le j \le k$, in $T_{\overline{w}}$ is a 1-element, then any element $t_{i'j'}$, ($1 \le i' < i$ and $j < j' \le k$) or ($i < i' \le n-k$ and $1 \le j' < j$), must be a 0-element. It implies that any two columns/rows in $T_{\overline{w}}$ which have two 1-elements are not identical.

Moreover since $|ps(\mathbf{w})| = k$, it guarantees that

(d) Each column of $T_{\overline{w}}$ has at least one 1-element; and

(e) Any two columns of $T_{\overline{w}}$ which have exactly one 1-element are not identical.

We can conclude from properties (b)-(e) that $T_{\overline{w}(q)}$, $1 \leq q \leq u$, must be with one of four structures as follows (without loss of comprehensibility, all 0-elements are not displayed).

$$T_1 = \begin{pmatrix} 1 & 1 & & & \\ & 1 & \ddots & & \\ & & \ddots & 1 & \\ & & & 1 & 1 \\ & & & & 1 \end{pmatrix}, \quad T_2 = \begin{pmatrix} 1 & & & & \\ 1 & 1 & & & \\ & 1 & \ddots & & \\ & & \ddots & 1 & \\ & & & 1 & 1 \end{pmatrix},$$

$$T_3 = \begin{pmatrix} 1 & & & & \\ 1 & 1 & & & \\ & 1 & \ddots & & \\ & & \ddots & 1 & \\ & & & 1 & 1 \\ & & & & 1 \end{pmatrix}, \quad T_4 = (1).$$

Note that

$$\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}, \text{ and } \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

are special cases of T_1, T_2, and T_3 respectively.

It is straightforward to see that all columns in $T_{\overline{w}(q)}$ are linearly independent. Consequently, all columns in $T_{\overline{w}}$ are linearly independent. ∎

Note that Theorem 2.3 does not hold for a general LFSM.

Example 2.6 *For a 5-cell LFSR(II), as shown in Figure 2.1(b), by Theorem 2.3, we select all 2-cell substate vectors, reported in Table 2.3, which produce* $2^{2\times 2} = 16$ *transitions. By referring to Tables 2.2 and 2.3, we can see that the selections in Table 2.3 are correct and no other 2-cell substate vector can be chosen to achieve* 16 *transitions.* □

Table 2.3 An example of selecting cells and partners.

w	$ps(\mathbf{w})$	Notes
(s_1, s_3)	$\{s_5, s_2\}$	s_5 is a partner of s_1 and s_2 is s_3's
(s_1, s_4)	$\{s_5, s_3\}$	s_5 is a partner of s_1 and s_3 is s_4's
(s_1, s_5)	$\{s_3, s_4\}$	s_3 is a partner of s_1 and s_4 is s_5's
(s_2, s_4)	$\{s_1, s_3\}$	s_1 is a partner of s_2 and s_3 is s_4's
(s_2, s_5)	$\{s_1, s_4\}$	s_1 is a partner of s_2 and s_4 is s_5's
(s_3, s_5)	$\{s_2, s_4\}$	s_2 is a partner of s_3 and s_4 is s_5's

In Theorem 2.3 we only consider a special case, in which each element in **w** has its own partner. In fact, we can easily extend the theorem to a general case: $|ps(\mathbf{w})| = r$ (*i.e.*, there exist at most r of k elements in **w**, each having its own partner), if and only if $rank(T_{\overline{w}}) = r$, where $r \leq k$.

2.4 Algorithms for Determining Partner Sets

If we want to evaluate the number of transitions for a given k-cell substate vector **w** of a state vector **s** for an n-cell LFSR or LHCA with a maximum length cycle, we only need to construct a partner set $ps(\mathbf{w})$, and then evaluate the number of transitions 2^{k+r}, where $r = |ps(\mathbf{w})|$. Since the LHCA and LFSR have straightforward next state functions, it is easy to design algorithms which determine the partner set for the given substate vector **w** for them.

We first consider an n-cell LHCA with a maximum length cycle. By observing the next state function in Definition 2.1, we can see the following cases are a key to designing a algorithm for determining the $ps(\mathbf{w})$.

(a) If s_1 is in **w**, only s_2 may be used as a partner of s_1;

(b) For $1 < p < n$ and s_p is in **w**, if s_{p-1} has already been taken (*i.e.*, s_{p-1} is

Figure 2.4 An algorithm for determining a partner set $ps(\mathbf{w})$ for an LHCA.

$C = \{\ i \mid s_i\ is\ in\ \mathbf{w}\ \}$;

$D = \emptyset$;

for $p = 1$ **to** n **do**

if $p \in C$ **then**

begin

  if $(p = 1)$ **or** $(p - 1 \in D$ **and** $p \neq n)$

    then $D = D \cup \{p, p + 1\}$

    else $D = D \cup \{p - 1, p\}$

end;

$ps(\mathbf{w}) = \{\ s_j \mid j \in D - C\ \}$.

in **w** or s_{p-1} is a partner of s_{p-2}), then s_{p+1} may be used as a partner of s_p; otherwise, s_{p-1} is chosen as a partner of s_p;

(c) If s_n is in **w**, only s_{n-1} may be used as a partner of s_n.

Formalizing the above cases, we describe the algorithm for the LHCA in Figure 2.4.

For an n-cell LFSR(II) with a maximum length cycle, an algorithm for determining a partner set $ps(\mathbf{w})$ is shown in Figure 2.5. In a similar fashion, the algorithm for the LFSR(I) can be easily designed.

Example 2.7 *Consider an n-cell LHCA with a maximum length cycle, where n is even. We want to derive a partner set* $ps(\mathbf{w})$ *for* $\mathbf{w} = (s_1, s_3, \ldots, s_{n-1})$, i.e., **w** *includes all odd cells and* $k = n/2$. *By Algorithm 1, we get* $ps(\mathbf{w}) = \{s_2, s_4, \ldots, s_n\}$.

Figure 2.5 An algorithm for determining a partner set $ps(\mathbf{w})$ for an LFSR(II).

/* Let $P(x) = x^n + c_{n-1}x^{n-1} + \cdots + c_2x^2 + c_1x^1 + 1$ */

$C = \{ i \mid s_i \textit{ is in } \mathbf{w} \}$;

$F = \{ n - i \mid 1 \leq i < n \textit{ and } c_i \textit{ is } 1 \textit{ in } P(x) \} \cup \{ n \}$;

$D = \{ p - 1, p \mid p \in C \textit{ and } p \neq 1 \}$;

if $(1 \in C)$ **then**

begin

$D = D \cup \{ 1 \}$;

if $\exists(1 < i \leq n$ **and** $i \notin D$ **and** $i \in F)$

then $D = D \cup \{ i \}$;

end;

$ps(\mathbf{w}) = \{ s_j \mid j \in D - C \}$.

---

i.e., $ps(\mathbf{w})$ *includes all even cells and* $|ps(\mathbf{w})| = k$. *By Theorem 2.3, we conclude that the given* $\mathbf{w}$ *has the maximum transition capability. This is a simple approach to prove Theorem 5 in* [ZBM92]. *In fact, all of the theorems for transition properties, given in* [ZBM92] *can be easily proved by the algorithms and theorems presented in this chapter.* □

Chapter 3

Transitions of LHCA and LFSR

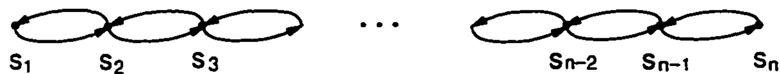
In this chapter. we use a combinatorial approach to derive the number of different k-cell substate vectors. which have 2^{2k} transition capability. with $k \leq \lfloor n/2 \rfloor$ for the LHCA and LFSR.

Note that a deep understanding of the material presented in this chapter is useful in selecting a set of cells such that the corresponding test vector sequence covers the maximal transition.

3.1 Transitions of LHCA

Consider an n-cell LHCA with a maximum length cycle. as shown in Figure 3.1.

Figure 3.1 A simplified structure of LHCA.



Theorem 3.1 *Let $f_1(n)$ be the total number of distinct k-cell substate vectors which produce 2^{2k} transitions with $k = \lfloor n/2 \rfloor$ for the n-cell LHCA. We have*

$$f_1(n) = \begin{cases} 2f_1(n-2), & \text{even } n \text{ and } n \geq 4, \\ 2f_1(n-2) + f_1(n-3), & \text{odd } n \text{ and } n \geq 5. \end{cases} \quad (3.1)$$
$$f_1(1) = 1, \quad f_1(2) = 2, \quad f_1(3) = 3.$$

Proof. The key to the proof is to determine how many selections there are for $\mathbf{w}$ such that each element in $\mathbf{w}$ has its own partner, i.e., $|ps(\mathbf{w})| = k$. For $n = 1, 2,$ and 3, it is obvious.

When n is even and $n \geq 4$, if we select k cells for $\mathbf{w}$, each must have a unique partner from the remaining k cells. That is,

(a) If s_1 is included in $\mathbf{w}$, s_2 must be used as a partner of s_1. There are $f_1(n-2)$ ways to choose $k-1$ cells from $\{s_3, s_4, \ldots, s_n\}$;

(b) If s_2 is included in $\mathbf{w}$, s_1 must be used as a partner of s_2. There are $f_1(n-2)$ ways to choose $k-1$ cells from $\{s_3, s_4, \ldots, s_n\}$.

Therefore, for the case of even n, the total number of selections is $2f_1(n-2)$.

When n is odd and $n \geq 5$, if we select k cells for $\mathbf{w}$, we need k cells as partners of $\mathbf{w}$ and have one cell which is neither included in $\mathbf{w}$ nor a partner. That is,

(a) If s_1 is included in $\mathbf{w}$, s_2 must be used as a partner of s_1. Then we consider selecting $\lfloor n/2 \rfloor - 1$ elements from $\{s_3, s_4, \ldots, s_n\}$ such that each has its own partner. The total number of selections is $f_1(n-2)$;

(b) If s_2 is included in $\mathbf{w}$ and s_1 is counted as a partner of s_2, the total number of selections is $f_1(n-2) - f_1(n-3)$, where $f_1(n-3)$ is the number of selections if s_2 is in $\mathbf{w}$ and s_3 is a partner of s_2;

(c) If s_1 is neither included nor a partner of any cell in $\mathbf{w}$, there are $f_1(n-1) = 2f_1(n-3)$ selections.

Hence, the total number of selections for the case of odd n is $2f_1(n-2)+f_1(n-3)$.

■

It is easy to solve the recurrence relation in Eqn (3.1) as follows.

$$f_1(n) = \begin{cases} 2^{\lfloor n/2 \rfloor}, & \textit{even } n, \\ (\lfloor n/2 \rfloor + 2)2^{(n-3)/2}, & \textit{odd } n. \end{cases}$$

Theorem 3.2 *Let $f_2(n,k)$ be the total number of distinct k-cell substate vectors which produce 2^{2k} transitions with $k \leq \lfloor n/2 \rfloor$ for the n-cell LHCA. We have*

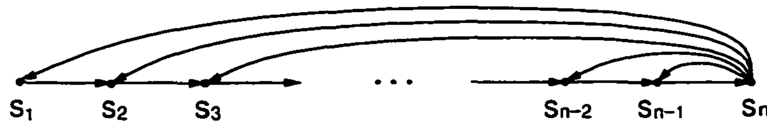
$$f_2(n,k) = \begin{cases} 2f_2(n-2,k-1) + f_2(n-1,k) - f_2(n-3,k-1), & k < \lfloor n/2 \rfloor, \\ f_1(n), & k = \lfloor n/2 \rfloor, \end{cases}$$
$$f_2(n,1) = n.$$

Proof. It is similar to the proof of Theorem 3.1. ■

3.2 Transitions of LFSR(I)

Consider an n-cell LFSR(I) with a maximum length cycle and assume[1] that each cell s_i, $1 \leq i < n$, gets feedback from cell s_n, as shown in Figure 3.2.

Figure 3.2 A simplified structure of LFSR(I).



[1] We use this assumption to determine an upper bound on the number of selections for the LFSR(I) because there is no general equation for any LFSR(I) concerning the number of different selections.

Theorem 3.3 *Let $f_1(n)$ be the total number of distinct k-cell substate vectors which produce 2^{2k} transitions with $k = \lfloor n/2 \rfloor$ for the n-cell LFSR(I). We have*

$$f_1(n) = \begin{cases} f_1(n-2)+1, & \text{even } n \text{ and } n \geq 4, \\ f_1(n-2)+f_1(n-1)+\lfloor n/2 \rfloor - 1, & \text{odd } n \text{ and } n \geq 5. \end{cases} \quad (3.2)$$

$$f_1(1) = 1, \quad f_1(2) = 2, \quad f_1(3) = 3.$$

Proof. The key to the proof is to select k cells to construct $\mathbf{w}$ such that each element in $\mathbf{w}$ has its own partner, i.e., $|ps(\mathbf{w})| = k$. For $n = 1, 2,$ and 3, it is obvious.

When n is even and $n \geq 4$, each cell should be considered as either an element in $\mathbf{w}$ or a partner of an element in $\mathbf{w}$.

(a) If s_1 is included in $\mathbf{w}$, s_n must be used as a partner of s_1. We only have one way to select $\lfloor n/2 \rfloor - 1$ cells from $\{s_2, s_3, \ldots, s_{n-1}\}$ in order to produce 2^{2k} transitions, i.e., we select $\mathbf{w} = (s_1, s_3, \ldots, s_{n-1})$ such that each element in $\mathbf{w}$ has its own partner;

(b) If s_2 is included in $\mathbf{w}$, s_1 must be used as a partner of s_2 because if s_n is used as a partner of s_2, there does not exist a proper selection. The number of selecting $\lfloor n/2 \rfloor - 1$ cells from $\{s_3, s_4, \ldots, s_n\}$ is $f_1(n-2)$.

Therefore, the total number of selections for the case of even n is $f_1(n-2)+1$.

When n is odd and $n \geq 5$, we have one cell which is neither included in $\mathbf{w}$ nor a partner.

(a) If s_1 is included in $\mathbf{w}$, s_n must be used as a partner of s_1, with the result that $s_2, s_3, \ldots, s_{n-1}$ cannot consider s_n as a partner. Thus there are $\lfloor n/2 \rfloor$ selections in this case;

(b) If s_2 is included in $\mathbf{w}$ and s_1 is counted as a partner of s_2, there are $f_1(n-2)-1$ selections, w here the -1 term corresponds to a selection of using s_n as a partner of s_2;

(c) If s_1 is neither included not a partner of any cell in **w**. there are $f_1(n-1)$ selections.

Thus the total number of selections for the case of odd n is $f_1(n-2)+f_1(n-1)+\lfloor n/2 \rfloor - 1$. ■

The recurrence relation in Eqn (3.2) can be solved as follows:

$$f_1(n) = \begin{cases} n/2+1, & \textit{even } n, \\ (n^2+3)/4, & \textit{odd } n. \end{cases}$$

Theorem 3.4 *Let $f_2(n,k)$ be the total number of distinct k-cell substate vectors which produce 2^{2k} transitions with $k \le \lfloor n/2 \rfloor$ for the n-cell LFSR(I). We have*

$$f_2(n,k) = \begin{cases} g(n-2,k-1) - g(n-3,k-1) \\ \qquad +f_2(n-2,k-1)+f_2(n-1,k), & k < \lfloor n/2 \rfloor, \\ f_1(n), & k = \lfloor n/2 \rfloor, \end{cases}$$

$$f_2(n,1) = n,$$

where $g(n,k)$ is defined as follows:

$$g(n,k) = \begin{cases} g(n-2,k-1)+g(n-1,k), & k < \lfloor n/2 \rfloor, \\ 1, & k = \lfloor n/2 \rfloor \textit{ and even } n, \\ \lfloor n/2 \rfloor + 1, & k = \lfloor n/2 \rfloor \textit{ and odd } n, \end{cases}$$

$$g(n,1) = n-1.$$

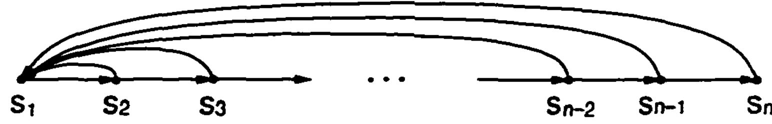
Proof. Similar to the proof of Theorem 3.3. ■

3.3 Transitions of LFSR(II)

Consider an n-cell LFSR(II) with a maximum length cycle and assume[2] that cell s_1 receives feedback from cells $s_2, s_3, \ldots, s_n$, as shown in Figure 3.3.

[2]The assumption is used to determine an upper bound on the number of selections for the LFSR(II) because there is no general equation for any LFSR(II).

Figure 3.3 A simplified structure of LFSR(II).



Theorem 3.5 *Let $f_1(n)$ be the total number of distinct k-cell substate vectors which produce 2^{2k} transitions with $k = \lfloor n/2 \rfloor$ for the n-cell LFSR(II). We have*

$$f_1(n) = \begin{cases} f_1(n-2)+1, & \text{even } n \text{ and } n \geq 4, \\ f_1(n-1)+f_1(n-2), & \text{odd } n \text{ and } n \geq 3, \end{cases} \tag{3.3}$$
$$f_1(1) = 1, \quad f_1(2) = 2.$$

Proof. The proof is similar to the proof of Theorem 3.3 in using Theorem 2.3 to construct a k-cell substate vector $\mathbf{w}$ such that each element in $\mathbf{w}$ has its own partner. For $n = 1$ and 2, it is obvious.

When n is even and $n \geq 4$, we have the following:

(a) If s_n is included in $\mathbf{w}$, s_{n-1} must be used as a partner of s_n. There are $f_1(n-2)$ ways of choosing $\lfloor n/2 \rfloor - 1$ cells from $\{s_1, s_2, \ldots, s_{n-2}\}$;

(b) If s_n is not included in $\mathbf{w}$, there is only one selection, i.e., $\mathbf{w} = (s_1, s_3, \ldots, s_{n-1})$.

Therefore, the total number of selections for the case of even n is $f_1(n-2)+1$.

When n is odd and $n \geq 3$, we have:

(a) If s_n is included in $\mathbf{w}$, s_{n-1} must be used as a partner of s_n. There are $f_1(n-2)$ ways of choosing $\lfloor n/2 \rfloor - 1$ cells from $\{s_1, s_2, \ldots, s_{n-2}\}$;

(b) If s_n is not included in $\mathbf{w}$, s_n can be ignored. There are $f_1(n-1)$ ways of choosing $\lfloor n/2 \rfloor$ cells from $\{s_1, s_2, \ldots, s_{n-1}\}$.

Hence, the total number of selections for the case of odd n is $f_1(n-1)+f_1(n-2)$.

■

It is easy to solve the recurrence relation in Eqn (3.3) as follows:

$$f_1(n) = \begin{cases} n/2 + 1, & \textit{even } n, \\ (n+3)(n+1)/8, & \textit{odd } n. \end{cases}$$

Theorem 3.6 *Let $f_2(n,k)$ be the total number of distinct k-cell substate vectors which produce 2^{2k} transitions with $k \leq \lfloor n/2 \rfloor$ for the n-cell LFSR(II). We have*

$$f_2(n,k) = \begin{cases} f_2(n-2,k-1) + f_2(n-1,k), & k < \lfloor n/2 \rfloor, \\ f_1(n), & k = \lfloor n/2 \rfloor, \end{cases}$$

$$f_2(n,1) = n.$$

Proof. Similar to the proof of Theorem 3.5. ∎

3.4 Summary

We have developed an approach to calculate the number of transitions for a given substate vector for an LHCA or LFSR. The approach is efficient because it directly examines the feedback, *called a partner*, for each cell in the substate vector instead of evaluating the rank of the corresponding submatrix. In this way, we can derive the number of distinct k-cell substate vectors that have 2^{2k} transitions, which gives an indication, to a certain extent, about how good the test vector generator used is in detecting *sequential* faults.

On the basis of Theorems 3.1, 3.3, and 3.5, we list the number of different k-cell substate vectors, which have 2^{2k} transition capability, with $k = \lfloor n/2 \rfloor$ for LHCA and LFSR from degree 5 to 50 in Table 3.1 for odd n and in Table 3.2 for even n. We can see that for the LHCA, the number of possible selections increases exponentially while for the LFSR the number increases linearly or quadratically.

It is very clear from the theoretical results that the LHCA have a much higher transition space than the LFSRs, and should consequently perform much better as

generators. To investigate whether this is true in practice. we consider experiments with the ISCAS85 benchmark circuits in the next chapter.

Table 3.1 Number of selections for k-cell substate with 2^{2k} transitions for odd n.

n	$k = \lfloor n/2 \rfloor$	LHCA	LFSR(I)*	LFSR(II)*
5	2	8	7	6
7	3	20	13	10
9	4	48	21	15
11	5	112	31	21
13	6	256	43	28
15	7	576	57	36
17	8	1280	73	45
19	9	2816	91	55
21	10	6144	111	66
23	11	13312	133	78
25	12	28672	157	91
27	13	61440	183	105
29	14	131072	211	120
31	15	278528	241	136
33	16	589824	273	153
35	17	1245184	307	171
37	18	2621440	343	190
39	19	5505024	381	210
41	20	11534336	421	231
43	21	24117248	463	253
45	22	50331648	507	276
47	23	104857600	553	300
49	24	218103808	601	325

* The numbers are upper bounds on the number of selections.

Table 3.2 Number of selections for k-cell substate with 2^{2k} transitions for even n.

n	$k = \lfloor n/2 \rfloor$	LHCA	LFSR(I)*	LFSR(II)*
6	3	8	4	4
8	4	16	5	5
10	5	32	6	6
12	6	64	7	7
14	7	128	8	8
16	8	256	9	9
18	9	512	10	10
20	10	1024	11	11
22	11	2048	12	12
24	12	4096	13	13
26	13	8192	14	14
28	14	16384	15	15
30	15	32768	16	16
32	16	65536	17	17
34	17	131072	18	18
36	18	262144	19	19
38	19	524288	20	20
40	20	1048576	21	21
42	21	2097152	22	22
44	22	4194304	23	23
46	23	8388608	24	24
48	24	16777216	25	25
50	25	33554432	26	26

* The numbers are upper bounds on the number of selections.

Chapter 4

Transition Faults in Combinational Circuits

4.1 Introduction

Built-in self-test(BIST) refers to those techniques where additional hardware is added to a design so that testing is accomplished without the need for external special purpose testing hardware. A widely accepted approach to BIST is to use a *pseudorandom* vector generator and a data compactor. The generator produces the test vectors to be applied to a circuit under test and the compactor reduces the response to these vectors to a single value (*e.g.*, 16 or 32 bits) known as the signature (*See* Chapter 2).

LFSR are the predominant choice in the BIST literature, both for the test vector generator and for the data compactor[BMS87]. Recently, alternative test vector generators based on LHCA have been considered and shown to be superior to LFSR based generators for stimulating faults with *sequential behavior*. LHCA have also been proposed as the data compactor in BIST. We examine practical issues concerning the use of LHCA, comparing with LFSR.

In BIST, to determine whether a given test vector generator is "good", a practical measure is to to see how many faults considered can be *stimulated* when the given test vector is applied to a circuit under test. In general, the LFSR and LHCA test vector generators offer virtually identical single *stuck-at* fault coverage. However, the LHCA have substantial promise in BIST for the more complex fault models.

We review the transition (gate delay) fault model and detection requirements because this is useful information in understanding the importance of the transition property for the given test vector generator discussed in the previous chapter. Then, we provide empirical comparisons to show that our analysis of the transition property presented in the previous chapter is a reasonable metric of the effectiveness of the test vector generator.

4.2 Transition (Gate Delay) Fault Model

Fault models allow us to specifically define the types of faults considered and their behavior. In addition, fault models allow us to represent the behavior of physical occurrences. Fault models attempt to cover the types of faults that can occur although they are not completely accurate in practice.

A single *stuck-at* fault assumes a circuit failure corresponds to a line of the circuit being permanently fixed at 0 or at 1. A circuit with p lines has $2p$ possible single *stuck-at* faults. We do simple fault collapsing on single gates, *e.g.*, an input of an AND gate *stuck-at* 0 fault is equivalent to the output of the gate *stuck-at* 0 fault.

A delay test of a combinational circuit in a clocked environment is defined to be a test of the ability of the combinational logic to propagate data in time for clocking into the next stage of latches (*see* [Smi85]). Two different delay fault models, called the *gate delay fault* (or *transition fault*) model and *path delay fault* model, respectively, have been proposed and are frequently used. We are concentrating on

the *transition fault* model in this chapter.

The *transition* faults considered are termed *slow-to-rise* and *slow-to-fall* faults. Let $c_1, c_2, \ldots, c_q$ be the correct (*expected*) bit sequence for a line of a circuit over some time period. The *transition* faults yield the sequence $d_i, 0 < i \leq q$, defined as follows:

(a) *slow-to-rise*

$$d_i = \begin{cases} 1, & c_{i-1} = 1 \text{ and } c_i = 1 \\ 0, & \text{otherwise} \end{cases}$$

(b) *slow-to-fall*

$$d_i = \begin{cases} 0, & c_{i-1} = 0 \text{ and } c_i = 0 \\ 1, & \text{otherwise} \end{cases}$$

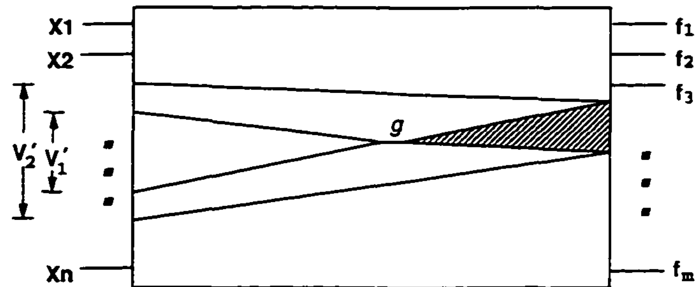
where $c_0 = c_1$. A circuit with p lines has $2p$ potential single *transition* faults. *Transition* fault equivalence rules have been partly considered in [WLRI87]. We remove *slow-to-rise* and *slow-to-fall* faults for the NOT gate output and also remove *slow-to-rise* faults for NOR and AND gate outputs and *slow-to-fall* faults for NAND and OR gate outputs since these can be shown to be equivalent to gate input *transition* faults [Zha93].

Transition faults cause combinational circuits to behave sequentially, and they can not be modeled as classical (*e.g.*, *stuck-at*) faults. Hence, they are called *sequential* faults.

A single *transition* fault on a given line g requires a pair of vectors for detection: the first to 'set-up' the value, and the second to 'propagate' the fault effect to a circuit output. When a pair of vectors is applied, it produces one of four possible different sequences on the line g: 00, 01, 10, and 11. The sequences 01 and 10 are used to detect *slow-to-rise* and *slow-to-fall* faults on line g since *slow-to-rise* changes 01 to 00 and *slow-to-fall* changes 10 to 11 on the given line g. Note that only the

second vector is used to propagate the fault effect to the circuit output because the fault does not change the first value produced by the first vector.

Figure 4.1 Simplified example for detecting transition fault.



Consider a circuit illustrated diagrammatically in Figure 4.1. where we are interested in detecting a *slow-to-rise* fault on line g. First we need vector $\mathbf{V_1}$ to set 0 on line g, and then use vector $\mathbf{V_2}$ to propagate the effect of the fault through the shadow area to at least one of the m outputs. This vector $\mathbf{V_2}$, of course, also detects a *stuck-at* 0 fault on line g. However, while both $\mathbf{V_1}$ and $\mathbf{V_2}$ are n-cell vectors, that is, they assign values to all the input variables, in general not all of those inputs are relevant (or even connected) to g. That is, only a subvector $\mathbf{V_1'}$ of $\mathbf{V_1}$ and a subvector $\mathbf{V_2'}$ of $\mathbf{V_2}$ are actually used (It is obvious that $|\mathbf{V_1'}| \leq |\mathbf{V_2'}|$.) From the above discussion, we can see that the transition property presented in the previous chapter is a good metric for comparing generators when stimulating faults requiring a pair of vectors.

4.3 Experimental Results

Given that an LHCA has a much larger number of distinct k-cell substate vectors, $k \leq \lfloor n/2 \rfloor$, which have 2^{2k} transition capability, than an LFSR, we would expect a higher fault coverage. In this section, we use simulation experiments on the ISCAS85 benchmark circuits in [BF85] to observe the effectiveness of test vectors generated

by the LHCA and LFSR. The experiments are based on *transition faults*, *i.e.*, *slow-to-rise* and *slow-to-fall* fault simulation [WLRI87]. Comprehensive *transition* fault equivalence rules are applied. A fault is detected if the output data stream in the presence of the fault differs from the output data stream in the fault-free case. All redundancies in the circuits have been removed by an approach in [TvdG91], *i.e.*, there are no untestable *transition* faults in the circuits.

The simulation results for *transition* faults are shown in Table 4.1. The results are collected by performing the fault simulation using test sequences of length 102,000. We simulated each circuit by using the LHCA [CZ95, ZMM91] and LFSR [BMS87] test vector generators with degree corresponding to the number of the circuit inputs. For each circuit, we used 100 different random connections between the circuit inputs and the test vector generator outputs. In the table, we report the best (and the number of times the best result is achieved in 100 random connections), worst (and the number of times the worst result is achieved in 100 random connections), as well as the median, and average results with respect to the number of undetected faults. It can be seen that, the LHCA have better fault coverage than the LFSRs and have greater chances to cover all faults considered. For example, for the LHCA, more than half of the 100 connections detect all faults for circuit C880, whereas for the LFSR, none of the 100 connections achieve 100% fault coverage for the same circuit. Hence, the simulation results provide evidence that our analysis of transition properties presented in the preceding chapter is indeed a proper metric of the effectiveness of LHCA and LFSR test vector generators.

From our analysis of the transition properties and the empirical results, it can be seen that an LHCA has more potential to achieve a higher fault coverage for stimulating faults requiring a pair of vectors. However, an LFSR still has the possibility to catch all faults considered, and infrequently, it performs better, *e.g.*, for circuit C7552nr (with 207 inputs) in Table 4.1, resulting from the circuit topology

Table 4.1 A summary of *transition* fault simulation results for 100 different random connections.

circuit name	number of faults	undetected faults for LHCA					
		best	(#)	worst	(#)	median	average
C432nr	658	0	(73)	4	(2)	0	0.38
C499nr	844	0	(99)	1	(1)	0	0.01
C880	1305	0	(56)	5	(3)	0	0.74
C1355nr	2076	14	(1)	28	(2)	22	22.18
C1908nr	2468	0	(71)	18	(1)	0	0.99
C2670nr	2532	54	(2)	119	(2)	85	84.91
C3540nr	4290	0	(27)	155	(1)	2	10.51
C5315nr	7222	0	(91)	3	(1)	0	0.12
C6288nr	9987	0	(40)	11	(1)	1	1.21
C7552nr	9104	85	(2)	132	(1)	109	109.01

circuit name	number of faults	undetected faults for LFSR(II)					
		best	(#)	worst	(#)	median	average
C432nr	658	0	(2)	16	(1)	5	5.25
C499nr	844	0	(67)	2	(2)	0	0.35
C880	1305	1	(1)	30	(1)	12	13.61
C1355nr	2076	17	(1)	36	(1)	26	26.02
C1908nr	2468	1	(3)	35	(1)	11	11.81
C2670nr	2532	115	(1)	191	(1)	146	145.16
C3540nr	4290	21	(1)	250	(1)	70	88.01
C5315nr	7222	4	(2)	73	(1)	14	17.48
C6288nr	9987	9	(1)	28	(1)	18	17.82
C7552nr	9104	65	(1)	115	(1)	91	90.82

and only a portion of the total cycle being generated. Thus. specific knowledge of a circuit under test is useful while choosing a test vector generator.

It is certain that. for any circuit. we can use simulation to get the best connection between the circuit inputs and the generator outputs. However. simulating a complex circuit is time consuming. An alternative way is to analyze the circuit topology and produce reasonable connections in order to avoid unnecessary simulations. *i.e.*. discarding any connection which may not produce a good fault coverage.

4.4 Discussion and Conclusion

We have seen that the number of distinct k-cell substate vectors which have the maximum transition capability for $1 \leq k \leq \lfloor n/2 \rfloor$ is a good metric for choosing an n-cell LFSM based test vector generator for detecting faults requiring a pair of vectors in a BIST environment. The comparisons of transition capabilities and simulation results provide information that is useful in understanding details of the test vectors produced by the LFSR and LHCA. We have noticed that the LHCA have a higher transition space than the LFSR. and consequently perform better as generators in general.

For a given n-input circuit under test. we need to apply a test sequence produced by an n-cell test vector generator for detecting *sequential faults.* In general. we have $n!$ different ways to assign connections between the circuit input and the generator output. An interesting question is proposed in [ZBM92. Zha93] as to whether a proper connection can be identified based on specific knowledge of the circuit topology or the functionality of the circuit on knowing the *sequential faults* considered and the transition property for a BIST vector generator used. such that the fault coverage is maximized.

A considerable work about improving stuck-at fault coverage has been presented

recently (*e.g.*, [Avr94, HK93, KT94, LGB94, MMR94, TM94, HRT+95]). However, not much attention is being paid about how to improve fault coverage for *sequential* faults. One of the major reasons is that the latter problem is harder than the former. For example, if all gate delay (transition) faults are detected, then all corresponding stuck-at faults will be covered automatically.

Some heuristic approaches based on the circuit input separation are introduced in [Kad93, SB92] to improve the fault coverage for the *delay* faults, using LFSR as a test vector generator. Our work on the choice of substate vectors for LHCA or LFSR to achieve maximum transition count provides necessary information to determine the appropriate connection of the circuit inputs to the LFSM generator outputs in order to maximize the fault coverage for *sequential* faults. The work of determining the proper connection is a promising area for further research.

However, when doing this, we must understand that any method proposed which works in principle for a set of small circuits does not necessarily mean that it works for the general case. For example, for a given n-input circuit under test, we may use a $2n$-cell LHCA (or LFSR) test vector generator such that we can apply all pairs of vectors to the circuit. Unfortunately, such an approach is not practical because in the BIST environment, the fault considered (corresponding to the existing defects) should be detected in limited time. Perhaps a choice based on random connections and simulation can easily achieve the expected result (at least our preliminary studies indicate this to be so).

Chapter 5

Transition Faults in Look-up Table Based FPGA Circuits

Considering the memory elements involved in the implementation of look-up table based FPGA (Field Programmable Gate Arrays) combinational circuits. we propose a transition fault model for such circuits. Experimental simulation results are provided and detection requirements for the transition faults are discussed. We claim that technology mapping for testability is an important issue when the circuit is realized using the look-up table based FPGA.

5.1 Introduction

It is mentioned in [Tri92] that Field Programmable Gate Arrays (FPGA) can be fully tested after manufacture. so users' designs do not require test program generation. automatic test pattern generation. and design for testability. In fact. the possibility still exists that the circuit may not work properly, resulting from component defects. *e.g.*. component wear-out.

As applications of FPGA are increased, researchers are paying attention to test-

ing issues regarding such circuits. For example, with specific hardware embedded in the Xilinx FPGA, testing can be done by scanning in and scanning out the appropriate binary signals. Alternatively, two fault models are proposed in [PR94], which are useful in deriving test sets when the special test hardware is not present, or when the test sets through scan are very large.

In this chapter, we examine look-up table (LUT) based FPGA. In general, 2^K memory elements are involved in the implementation of a K-input LUT, with each corresponding to one of 2^K K-tuple inputs to the LUT. Thus, a K-input LUT can represent any Boolean function of up to K variables. We can easily implement a general Boolean function using LUT-based FPGA because it provides programmable LUTs and programmable interconnections among the LUTs.

To observe the testing for a circuit with such an implementation, we need reasonable fault models such that we can evaluate the testing approaches applied. In this chapter, we emphasize fault models for BIST techniques although the fault models proposed may be suitable for other testing strategies.

The rest of this chapter is organized as follows. In Section 5.2, we introduce transition faults in look-up table based FPGA combinational circuits. In Section 5.3, we review the technology mapping problem for LUT-based FPGA and produce the mapping results for ISCAS85 benchmark circuits. In Section 5.4, we present experimental simulation results for the transition faults proposed in Section 5.2 using linear feedback shift registers and linear hybrid cellular automata as test vector generators. Detection requirements for testing the transition faults are discussed in Section 5.5. Finally, in Section 5.6, we summarize our major results and propose some possible further research directions.

5.2 Transition Fault Model

A circuit implemented as a collection of LUTs is a memory-based circuit. However, it should be noted that during the circuit operation (1) there is no writing operation involved; (2) each address decoder of the LUTs may not be controllable directly from the circuit inputs and (3) each value of the memory elements in the LUTs may not be observable directly at the circuit outputs. Therefore, methods for memory testing in [vdG91] may not be suitable for testing memory-based circuits.

In this section, we propose a transition fault model that results from access time failure caused by overly-slow signal transition in the address decoder [Coc75] of the LUT.

For convenience, we use the following notation in defining transition faults.

- $\mathcal{L}(x)$ - a K-input function corresponding to a K-input LUT;
- $\mathcal{L}_f(x_p, x_c)$ - a K-input function, depending on the previous input x_p and the current input x_c, regarding the presence of a transition fault in the LUT. That is, the presence of a transition fault in the LUT may affect the output of the function;
- $I = \{0, 1\}^K$ - a set of inputs to function $\mathcal{L}(x)$;
- $I_0 = \{x \in I \;:\; \mathcal{L}(x) = 0\}$;
- $I_1 = \{x \in I \;:\; \mathcal{L}(x) = 1\}$;
- $I_0 \cap I_1 = \emptyset$ and $I_0 \cup I_1 = I$.

For example, if a 5-input LUT represents a 5-input AND function, then

$$\mathcal{L}(x) = \begin{cases} 1, & x \in I_1, \\ 0, & x \in I_0, \end{cases}$$

where $I_1 = \{(1,1,1,1,1)\}$ and $I_0 = \{0,1\}^5 - \{(1,1,1,1,1)\}$.

For a given K-input LUT, the transition faults considered are termed as *slow-to-fall* and *slow-to-rise* faults.

(a) The *slow-to-fall* fault corresponding to input x_c ($\in I_0$) is defined as

$$\mathcal{L}_f(x_p, x_c) = \begin{cases} 1, & x_p \in I_1 \\ 0, & \text{otherwise.} \end{cases}$$

(b) The *slow-to-rise* fault corresponding to input x_c ($\in I_1$) is defined as

$$\mathcal{L}_f(x_p, x_c) = \begin{cases} 0, & x_p \in I_0, \\ 1, & \text{otherwise.} \end{cases}$$

Note that $\mathcal{L}_f(x_p, x_c) = \mathcal{L}(x_c)$ if no fault occurs. The number of *slow-to-rise* faults is $|I_1|$ and the number of *slow-to-fall* faults is $|I_0|$. The total number of the faults is $|I_1| + |I_0| = 2^K$ for the K-input function corresponding to the LUT. Moreover, each of the 2^K faults must be treated separately because each one corresponds to one memory element and no fault equivalent rules can be applied.

For example, suppose that $\mathcal{L}(x)$ is a 3-input function corresponding to a 3-input LUT with

$$\begin{aligned} I_0 &= \{(0,0,0), (0,1,1), (1,1,0), (1,1,1)\}, \\ I_1 &= \{(0,0,1), (0,1,0), (1,0,0), (1,0,1)\}. \end{aligned}$$

Table 5.1 shows results when no fault occurring and results because of the presence of the transition faults.

It should be mentioned that a K-input LUT can have 2^K different inputs. For each of the inputs, a transition fault (*slow-to-rise* or *slow-to-fall*) is defined. However, in the FPGA-based circuit, some LUTs may not see the all possible inputs since the inputs to the LUT are not necessarily the primary inputs of the circuit. As a result, some transition faults defined in the given LUT do not manifest themselves.

It can be seen that we need a pair of vectors for detecting a transition fault. In general, it is harder to detect the transition faults in an LUT than to detect a stuck-at fault and a transition fault in gates, because the second vector applied to a circuit under test must produce a specific input to the LUT for accessing a specific memory element examined. In Section 5.5, we derive the exact detection requirements for the given transition faults.

Table 5.1 An example of transition faults for a 3-input LUT.

input	fault-free	transition-fault	
x_c	$\mathcal{L}(x_c)$	x_p	$\mathcal{L}_f(x_p, x_c)$
(0,0,0)	0	$\in I_1$	1
(0,0,1)	1	$\in I_0$	0
(0,1,0)	1	$\in I_0$	0
(0,1,1)	0	$\in I_1$	1
(1,0,0)	1	$\in I_0$	0
(1,0,1)	1	$\in I_0$	0
(1,1,0)	0	$\in I_1$	1
(1,1,1)	0	$\in I_1$	1

5.3 Technology Mapping

To produce some experimental results for the fault model proposed, we need some benchmark circuits with reasonable sizes (regarding the numbers of inputs and outputs for the circuits). However, LUT-based benchmarks do not exist. Therefore, in this section, we review the technology mapping problem and produce some mapping results for the ISCAS85 benchmark circuits.

5.3.1 Problem Formalization

The technology mapping problem for LUT-based FPGA is to transform a general Boolean network into a functionally equivalent K-input LUT-based network. We briefly review the formalization of the LUT-based FPGA technology mapping problem based on the representations in [FS94, CD94].

We represent a Boolean network as a directed acyclic graph $G(V, E)$ where each node in V represents a logic gate and a directed edge (u, v) in E represents that node u is an input of node v. A primary input node has no incoming edge and a primary output node has no outgoing edge. The set of all input nodes of v is denoted by $Input(v)$, while the set of all output nodes of v is denoted by $Output(v)$. For classifying different nodes in V, we let

$$V \doteq V_{in} \cup V_m \cup V_{out},$$

where V_{in} is the set of primary input nodes which need not be assigned a lookup-table (LUT). V_{out} is the set of output nodes in which each node must be assigned an LUT, and V_m is the set of intermediate nodes which may or may not be assigned LUTs. Note that (a) the intersection of any two of them is an empty set; and (b) if none of the output nodes is used as the input of any node, then V_{out} includes only all primary output nodes.

(1) Restricted K-LUT Minimization (Mapping) Problem for $K \geq 2$ (K-RLMP)

Instance: A directed acyclic graph $G(V, E)$, positive integers $K \geq |Input(v)|$ for each node $v \in V$ and $B \leq |V - V_{in}|$.

Question: Is there a subset D with $V_{out} \subseteq D \subseteq V - V_{in}$ and $|D| \leq B$ such that for each node $v \in D$, there exists a set $Fence(v)$ with $Fence(v) \subseteq V_{in} \cup D$ and $|Fence(v)| \leq K$ such that (a) for each node $u \in Fence(v)$, there is a path

from u to v in G; and (b) for each node $u \in V_{in} - Fence(v)$, any path from u to v must pass through at least one node in $Fence(v)$?

Note that on describing the problem, we have used the following the assumptions:

(a) the number of inputs for each node in the Boolean network is less than or equal to K; and

(b) the Boolean property in the network is ignored.

It is easy to see that if we assign a K-input LUT to each node v in D and use nodes in $Fence(v)$ as inputs to v, we can find a feasible K-input LUT decomposition for G.

Farrahi and Sarrafzadeh in [FS94] show that the restricted cases of the lookup-table minimization problem, called K-RLMP, for FPGA technology mapping are NP-complete [GJ79] for $K \geq 5$. We claim that 4-RLMP and 3-RLMP are also NP-complete [ZMM96]. However, whether 2-RLMP is NP-complete remains an open problem.

In the following proofs, for simplicity, we ignore the construction of set $Fence(v)$ since it is visible. For convenience, we restate the 3-Satisfiability Problem [GJ79] as follows.

(2) 3-Satisfiability Problem (3-SAT)

Instance: Set U of variables, collection C of clauses over U such that each clause $c \in C$ has $|c| = 3$.

Question: Is there a satisfying truth assignment for C?

5.3.2 4-RLMP/3-RLMP and 3-SAT

Theorem 5.1 *4-RLMP problem is NP-complete.*

Proof. It is easy to see that 4-RLMP $\in$ NP since a nondeterministic algorithm only needs guess an assignment for a given number of 4-input LUTs and check in polynomial time whether the assignment decomposes the given graph which describes a Boolean network.

We transform 3-SAT to 4-RLMP. Let $X = \{x_1, x_2, \ldots, x_n\}$ be the set of variables and $C = \{c_1, c_2, \ldots, c_m\}$ be the set of clauses in an arbitrary instance of 3-SAT. We must construct a directed acyclic graph $G(V, E)$, which represents a Boolean network with $Input(v) \leq 4$ for each node $v \in V$, such that G can be decomposed using $2n + 3m$ 4-input LUTs if and only if C is satisfiable.

Without loss of generality, for each variable $x_i \in X$, let $\bar{x}_i$ be equivalent to x_{i+n}, and for each clause $c_j \in C$, if $\bar{x}_i \in c_j$, we replace it with x_{i+n} in c_j such that $c_j = \{x_{ju}, x_{jv}, x_{jw}\}$, where $1 \leq ju < jv < jw \leq 2n$. Note that for $y \in \{ju, jv, jw\}$, if $n < y \leq 2n$, then $x_y = \bar{x}_{y-n}$.

The construction for $V = V_{in} \cup V_m \cup V_{out}$ and E of G is as follows.

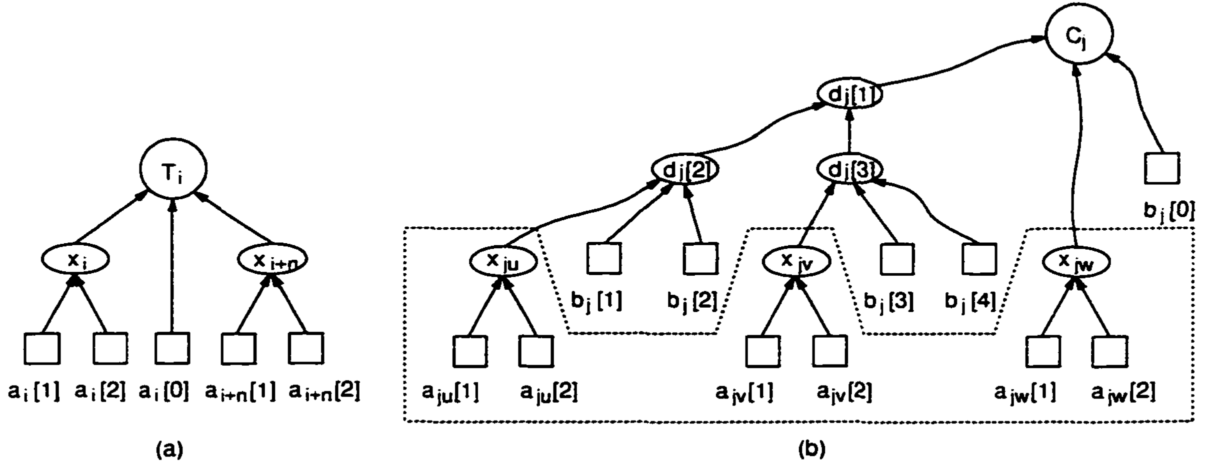
$$\begin{aligned}
V_{out} &= \{T_i \mid x_i \in X\} \cup \{C_j \mid c_j \in C\}, \\
V_m &= \{x_i, x_{i+n} \mid x_i \in X\} \cup \{d_j[1], d_j[2], d_j[3] \mid c_j \in C\}, \\
V_{in} &= \{a_i[0], a_i[1], a_i[2], a_{i+n}[1], a_{i+n}[2] \mid x_i \in X\} \\
&\quad \cup \{b_j[0], b_j[1], b_j[2], b_j[3], b_j[4] \mid c_j \in C\}.
\end{aligned}$$

and

$$\begin{aligned}
E = \; & \{(a_i[0], T_i), (x_i, T_i), (x_{i+n}, T_i), (a_i[k], x_i), \\
& \quad (a_{i+n}[k], x_{i+n}) \mid x_i \in X \text{ and } 1 \leq k \leq 2\} \\
& \cup \{(d_j[2], d_j[1]), (d_j[3], d_j[1]), (d_j[1], C_j), (b_j[0], C_j), (b_j[1], d_j[2]), (b_j[2], d_j[2]), \\
& \quad (b_j[3], d_j[3]), (b_j[4], d_j[3]), (x_{ju}, d_j[2]), (x_{jv}, d_j[3]), (x_{jw}, C_j) \mid c_j \in C\}.
\end{aligned}$$

We show an example of the construction for $x_i \in X$ in Figure 5.1(a) and for $c_j \in C$ in Figure 5.1(b), where a square denotes a node in V_{in}, a circle denotes a

Figure 5.1 An example of the construction for 4-RLMP from 3-SAT.



node in V_{out}, and an oval denotes a node in V_m. Note that in Figure 5.1(b), all nodes and edges in a polygon surrounded by the dash line are constructed in the case for $x_i \in X$ (*referring to* Figure 5.1(a)).

It is easy to see how the construction can be accomplished in polynomial time. It remains to be shown that G is decomposable by $2n + 3m$ 4-input LUTs if and only if C is satisfiable.

First, suppose that $t : X \rightarrow \{T, F\}$ is a satisfying truth assignment for C. The corresponding decomposition D for G includes the following elements:

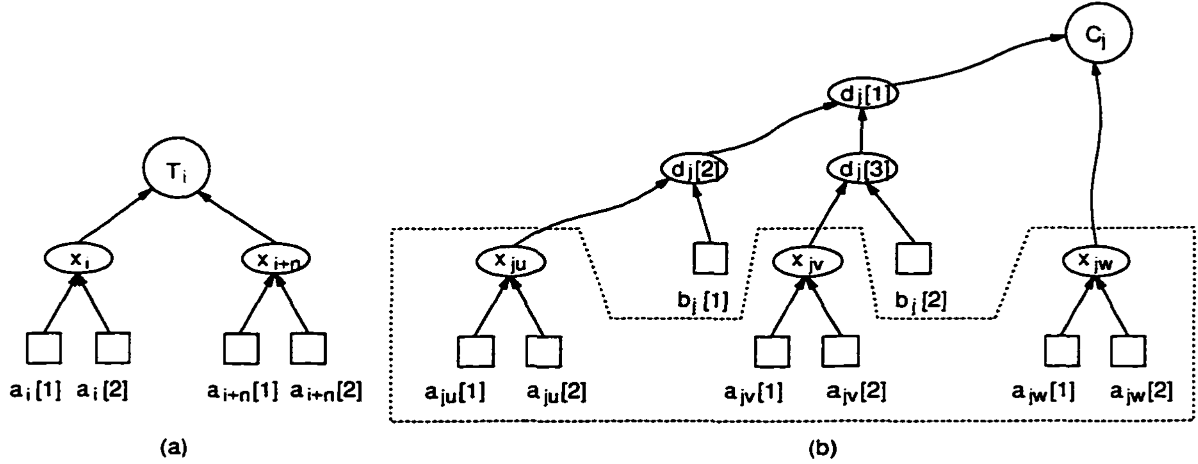
(1) T_i for each variable $x_i \in X$ (the number of such elements is n).

(2) either x_i, if $x_i = T$, or x_{i+n}, if $x_i = F$, for each variable $x_i \in X$ (the number of such elements is n),

(3) C_j for each clause $c_j \in C$ (the number of such elements is m).

(4) either $d_j[2]$ and $d_j[3]$ if $x_{jw} = T$, or $d_j[1]$ and $d_j[2]$ if $x_{jv} = T$, or $d_j[1]$ and $d_j[3]$ if $x_{ju} = T$ for each clause $c_j \in C$ with $c_j = \{x_{ju}, x_{jv}, x_{jw}\}$ (the number of such elements is $2m$).

Figure 5.2 An example of the construction for 3-RLMP from 3-SAT.



Obviously, D ($|D| = 2n + 3m$) is a decomposition solution for G.

Conversely, suppose D is a decomposition solution for G with $|D| \leq 2n + 3m$. By the previous construction, we have the following facts.

(1) $V_{out} \subseteq D$ and $|V_{out}| = n + m$.

(2) D must include one of two nodes, x_i and x_{i+n}, for each variable x_i in X. This gives n nodes.

(3) Finally, the remaining $2m$ nodes must be used as follows. For each clause $c_j \in C$ with $c_j = \{x_{ju}, x_{jv}, x_{jw}\}$, we need at least two of the three nodes, $d_j[1]$, $d_j[2]$, and $d_j[3]$, to be included in D on the condition that at least one of the three nodes, x_{ju}, x_{jv}, and x_{jw}, of c_j is in D. In other words, if none of the elements in c_j is in D, D has to include $d_j[1]$, $d_j[2]$, and $d_j[3]$, resulting in $|D| > 2n + 3m$, which is in contradiction with $|D| \leq 2n + 3m$.

Based on the above discussion, it is easy to find a satisfying truth assignment for C from D. ∎

We can use a similar approach to transform 3-SAT to 3-RLMP. Figure 5.2 shows an example of constructing 3-RLMP from 3-SAT for each variable $x_i \in X$ and each clause $c_j \in C$. Therefore, we have

Theorem 5.2 *3-RLMP problem is NP-complete.*

5.3.3 Remarks

We can see, from the transformation applied in the proofs, that 4-RLMP and 3-RLMP remain NP-complete even for 4-level Boolean networks (i.e., the number of edges on the longest path from any primary input node to any primary output node is 4), and for networks without reconvergent path. Moreover, if we use the restricted satisfiability problem [CD94] in the transformation, we find that 4-RLMP and 3-RLMP remain NP-complete for 3-bounded fanout networks (i.e., $|Output(v)| \leq 3$ for each node v in V).

If we apply the transformation used in the proofs to K-RLMP with $K > 4$, we can find a different NP-completeness proof from that presented by Farrahi and Sarrafzadeh in [FS94]. Note that the transformation in the proofs can also be used to prove NP-completeness of the technology mapping from the BDD (Binary Decision Diagram) based Boolean network [Bry86, Mil93, ZMM95a] to the 5-input LUT-based Boolean network.

We have proved that 4-RLMP and 3-RLMP are NP-complete. The remaining open problem is whether 2-RLMP is NP-complete. Neither the approach in [FS94] nor the approach in this section can be applied to the 2-RLMP.

5.3.4 Technology Mapping Experiments

Although the K-input LUT minimization problem for FPGA technology mapping is NP-complete, a number of heuristic algorithms have been developed to solve the

problem sub-optimally. We use a greedy algorithm similar to the one given in [Mil93] to do some experiments for the 5-input LUT-based FPGA technology mapping for the ISCAS85 benchmark circuits.

For each circuit we list. in Table 5.2. the original properties of the circuit regarding the number of gates used. the number of inputs. and the number of outputs. as well as the number of LUTs needed and the number of the possible transition faults existed after the technology mapping.

Table 5.2 Numbers of LUTs and numbers of faults for ISCAS85 circuits.

circuits	gates	inputs	outputs	LUTs	faults
c432	160	36	7	80	1122
c499	202	41	32	80	1280
c880	383	60	26	121	2476
c1355	546	41	32	117	2152
c1908	880	33	25	154	3488
c2670	1193	233	140	337	6206
c3540	1669	50	22	519	13674
c5315	2307	178	123	635	10748
c6288	2406	32	32	841	19168
c7552	3512	207	108	706	16160

In this dissertation. we have no intent to do any comparison between our technology mapping results and others obtained in different approaches since eight of ten circuits have never been used as benchmarks of the technology mapping. Moreover. the technology mapping is not a major issue of the dissertation.

5.4 Experimental Simulation Results

In Chapters 3 and 4, we have discussed the effectiveness of using LHCA and LFSR as the BIST generator for sequential-type faults. *e.g.*, gate delay faults. in combi-

national circuits. The presentation was based on a theoretical analysis as well as comparisons of experimental results. In this section, we examine the use of LHCA and LFSR as BIST generators for testing transition faults in the LUT-based FPGA combinational circuits.

Simulation of transition faults for LUT-based FPGA combinational circuits is similar to the one given in [WLRI87]. A fault is detected if the output data stream in the presence of the fault differs from that in the fault-free case.

Table 5.3 shows results collected by performing fault simulation for the transition faults using 102,000 test vectors. We simulated each circuit using three different test vector generators: the minimal cost LHCA in [CZ95], the minimal cost LFSR(II) in [BMS87, Bar92b], and the LFSR(I) with half taps evenly distributed, produced by the author. Each generator has degree corresponding to the number of inputs in the circuit under test.

For a fair comparison, we used 100 different trials for each circuit and test vector generator applied. The differences for each trial are (1) randomly chosen connections between the circuit inputs and the test vector generator outputs; and (2) randomly chosen initial state for the generator.

In the table, we report the best, the worst, the median, and the average results about the number of undetected faults. In each row, we show the difference between the results produced by the LHCA and the LFSR(II), as well as between the LHCA and the LFSR(I). Evidence that LHCA yield better fault coverage than LFSR is consistent with results of the theoretical analysis in Chapters 3 and 4.

We may notice, from Table 5.3, that a number of the faults are undetected after 102,000 test vectors are applied. One of the reasons is mentioned in Section 5.2 that the transition faults defined in the given q-input LUT ($2 \leq q \leq 5$) may not exist because the LUT only can get a subset of 2^q different inputs, resulting from the circuit topology or the circuit functionality. Another reason is that the faults

Table 5.3 A summary of the fault simulation results for 100 randomly chosen connections.

circuit	undetected faults at best			undetected faults at worst		
name	LHCA	LFSR(I)	LFSR(II)	LHCA	LFSR(I)	LFSR(II)
c432	235	238	240	256	273	288
c499	28	28	28	36	28	28
c880	321	342	338	432	533	541
c1355	560	560	560	576	564	568
c1908	797	821	807	999	1144	1204
c2670	2756	2758	2796	2844	2957	2976
c3540	3968	4072	4231	5665	6675	7235
c5315	2377	2381	2385	2407	2540	2702
c6288	8580	8586	8612	8618	8865	8850
c7552	4562	4576	4574	4703	4704	4762
comparison	1.000	1.007	1.016	1.000	1.066	1.099

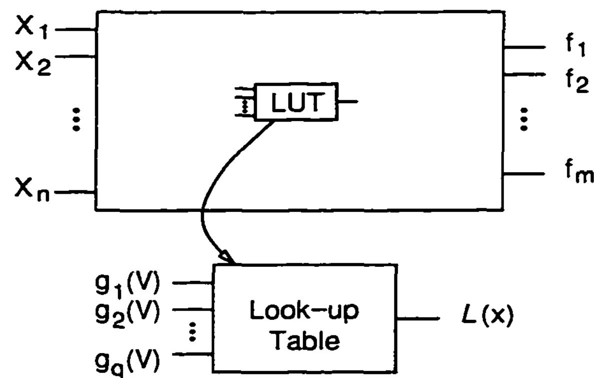
circuit	undetected faults at median			undetected faults on average		
name	LHCA	LFSR(I)	LFSR(II)	LHCA	LFSR(I)	LFSR(II)
c432	240	247	253	241.1	248.8	254.8
c499	28	28	28	28.2	28.0	28.0
c880	347	393	421	350.3	401.9	427.7
c1355	560	560	560	560.2	560.5	561.2
c1908	888	912	936	891.0	924.5	953.6
c2670	2815	2822	2830	2812.2	2823.5	2834.2
c3540	4077	4718	5108	4159.2	4916.8	5357.2
c5315	2392	2404	2427	2391.7	2409.8	2441.5
c6288	8586	8630	8680	8590.4	8639.9	8687.6
c7552	4615	4628	4644	4617.1	4631.6	4648.6
comparison	1.000	1.032	1.055	1.000	1.038	1.063

may be untestable, *i.e.*, they remain undetected even if all the possible pairs of test vectors are applied to the circuit.

5.5 Detection Requirements

We can see, from the simulation results, that a number of the faults considered are not detected using the given length of test vectors. It is instructive to know whether the given fault is testable. Thus, we use an illustration in Figure 5.3 to derive the detection requirements for given transition faults.

Figure 5.3 An example of the detection requirement for the transition fault.



Suppose that a K-input LUT-based FPGA combinational circuit, shown in Figure 5.3, has n inputs, $V = (X_1, X_2, X_3, ..., X_n)$, and m outputs, $f_1, f_2, ..., f_m$. Let $\mathcal{L}(x)$ be the function corresponding to a K-input LUT being observed with input $x = (g_1(V), g_2(V), ..., g_q(V))$, where $2 \leq q \leq K$ and $g_i, 1 \leq i \leq q$, is a function of input V. That is, the input to the LUT depends on the circuit input V. Moreover, let output $f_i = F_i(\mathcal{L}(x), V)$ for $1 \leq i \leq m$, i.e., f_i does depend on not only the circuit input, but also the output of the observed LUT.

Observing the definition of the *slow-to-fall* fault for the LUT in (a) of Section 5.2, we have the following result.

(a) The *slow-to-fall* fault corresponding to x_c ($\in I_0$), an input to the LUT resulting in the output of the LUT being 0, is testable if and only if there exist n-tuple vectors V_1 and V_2 such that

$$x_p = (g_1(V_1), g_2(V_1), \ldots, g_q(V_1)) \text{ and } x_p \in I_1, \tag{5.1}$$

$$x_c = (g_1(V_2), g_2(V_2), \ldots, g_q(V_2)), \tag{5.2}$$

$$\exists_{1 \leq i \leq m}[F_i(1, V_2) \oplus F_i(0, V_2) = 1]. \tag{5.3}$$

In other words, we need two vectors V_1 and V_2 to detect the transition fault. The first vector V_1 is to set the output of the LUT. In the case being observed, $\mathcal{L}(x_p) = 1$, i.e., the output of the LUT for input V_1 is 1, because $x_p = (g_1(V_1), g_2(V_1), \ldots, g_q(V_1)) \in I_1$, given in Equation (5.1). The second vector V_2 is to satisfy

$$x_c = (g_1(V_2), g_2(V_2), \ldots, g_q(V_2)),$$

which results in $\mathcal{L}(x_c) = 0$, i.e., the output of the LUT being 0, shown in Equation (5.2). And at the same time, the V_2 satisfies that if the output of the LUT is not changed from the previous value 1 to the current value 0 because of the *slow-to-fall* fault, then at least one of the m outputs of the circuit produces the value differing from that in the fault-free case, displayed in Equation (5.3).

In a similar approach, we can determine the necessary and sufficient conditions for detecting the *slow-to-rise* faults in the LUT, defined in (b) of Section 5.2, as follows.

(b) The *slow-to-rise* fault corresponding to x_c ($\in I_1$), an input to the LUT resulting in the output of the LUT being 1, is testable if and only if there exist n-tuple vectors V_1 and V_2 such that

$$x_p = (g_1(V_1), g_2(V_1), \ldots, g_q(V_1)) \text{ and } x_p \in I_0, \tag{5.4}$$

$$x_c = (g_1(V_2), g_2(V_2), \ldots, g_q(V_2)), \tag{5.5}$$

$$\exists_{1 \leq i \leq m}[F_i(1, V_2) \oplus F_i(0, V_2) = 1]. \tag{5.6}$$

Knowing the detection requirements for the transition faults considered, we can determine whether the given fault is testable. We are investigating this issue and trying to find an efficient algorithm for determining untestable transition faults.

5.6 Summary

Testing a memory-based circuit is different from testing memory because in the former, memory writing is not involved during the circuit operation and the memory elements may not be accessible directly from the circuit input or output. Considering the memory elements involved in LUT-based FPGA, we proposed the transition fault model, which can be used to evaluate the testing approaches applied to the circuit. Unlike the fault models proposed for the gate level, transition faults examined are more reliable since they are based on the final realization of the circuit.

Typically, the basic logic block in LUT-based FPGA has 5 inputs and one output. A circuit based on such a structure may have a considerable number of untestable transition faults. The detection requirements derived for the faults considered are definitely useful in identifying the untestable faults as well as in finding a good technology mapping algorithm which produces LUT-based FPGA circuits with fewer untestable faults or no untestable faults. The technology mapping for testability is a promising area for further research.

Chapter 6

Performance Evaluation of Self-Checking Circuits

In this and the following chapters. we consider the evaluation of the safety of a self-checking circuit with combinational logic. Since the circuit is tested under normal operation. it may stay in different states such as a perfect state in which any erroneous output can be detected. unstable states in which an erroneous output may be detected or may not. a safe-state when the erroneous output has been caught. and a fail-state because the erroneous output is undetected. as time goes on. Consequently. we propose a fail-safe evaluation. using a Markov model to describe the state transitions and predict the probability of the circuit not being in the fail-state.

We include a comparison with existing evaluation methods. the proposed approach being more practical because it estimates the safety of the circuit. which decreases as time goes on. instead of giving a constant probability measure.

6.1 Introduction

Self-checking circuits can detect the presence of transient and permanent faults because the circuits are designed with additional sub-circuits that are used to monitor whether the circuits work correctly during normal operation. However, without a proper evaluation, we may be misled into thinking that any faults in the self-checking circuits can be detected because the circuits are being tested under normal operation or they are totally self-checking and therefore, the self-checking circuits are safe. In this chapter, we try to expose critical problems in evaluating the safety of the self-checking circuits and state our initial work in this area.

A primary difficulty with self-checking circuits is that, while both a circuit and a checker may be totally self-checking, the resulting composite circuit is not. That is, it is possible for there to be erroneous outputs from the circuit which are not caught by the checker. This is caused by the presence of one or more faults in the checker which have not been exposed (and indeed may never be exposed) by an appropriate stimulus from the circuit (this is considered in detail in Section 6.2.3 below). The problem is that one of the two primary assumptions for totally self-checking circuits is often not met.

Designing self-checking circuits has been discussed for more than twenty years. However, not much work has been done in the area of evaluating the self-checking circuits. The existing evaluations are based on determining whether a given circuit satisfies the totally self-checking goal or calculating how much of that goal has been achieved by the given circuit[LM84, FMM84, FM87, LF93].

We propose fail-safe evaluation based on a Markov model regarding the testing procedure under normal operation of the circuit. The chapter begins with reviewing the development of totally self-checking circuits in Section 6.2, discusses the existing evaluations in Section 6.3. In the following chapter, we introduce our fail-safe evaluation in Section 7.1, give experimental results and comparisons in Section 7.2

and summarize the major results in Section 7.3.

6.2 Totally Self-Checking Goal

The goal to be reached by self-checking circuits is often called a totally self-checking goal. *i.e.*. the first erroneous output of the functional circuit is caught by the checker. We give a brief review of the concepts of totally self-checking circuits so that we can illustrate the difficulties of evaluating how closely a circuit meets this goal.

6.2.1 Preliminaries

In this chapter. we consider multiple-input and multiple-output combinational circuits. If a circuit has r primary inputs and q primary outputs, then the 2^r binary vectors of length r form the input space and the 2^q binary vectors of length q form the output space of the circuit. During normal. *i.e.*. fault-free. operation. the circuit receives a codeword from the input codeword space. a subset of the input space. and produces a codeword belonging to the output codeword space. a subset of the output space. Members of the input(output) space which are not in the input(output) codeword space are called the input(output) non-codewords.

To formalize our discussion, the term *fault* refers to one line of a circuit stuck-at 0 or stuck-at 1. A fault that involves only one stuck line is called a *single fault*. An *error* is defined as the incorrect logic signals caused by a fault. It should be noted that a circuit with a fault does not necessarily produce an error for a particular input.

For notational convenience. we define the following notations to be used hereafter.

- X – an input codeword space for a given circuit;

- Z - an output codeword space for a given circuit:
- F - a set of faults to be considered in a given circuit:
- $z(x)$ - a logic function for a given circuit;
- $\hat{z}_f(x)$ - a logic function for a given circuit with fault f.

6.2.2 Definitions

The concept of self-checking circuits is formally defined by Anderson and Metze in [AM73]. The following definitions are proposed to describe self-checking circuits.

Definition 6.1 *A circuit is fault-secure with respect to F if*

$$\hat{z}_f(x) \notin Z \quad or \quad \hat{z}_f(x) = z(x)$$

for any $x \in X$ and $f \in F$.

This definition implies that $\hat{z}_f(x)$ never takes an incorrect output value in Z. Conversely. when $\hat{z}_f(x)$ belongs to the output codeword space. it necessarily takes a correct value.

Definition 6.2 *A circuit is self-testing with respect to F if*

$$\forall f \in F. \ \exists x \in X. \quad \hat{z}_f(x) \notin Z.$$

This means that any fault in F can be detected by at least one input codeword.

Definition 6.3 *A circuit is totally self-checking (TSC) if it is fault-secure and self-testing.*

The following definition concerns TSC checkers.

Definition 6.4 *A circuit is code-disjoint if*

$$\forall x \in X, \ z(x) \in Z \quad and \quad \forall x \notin X, \ z(x) \notin Z.$$

Code disjointness can be viewed as a formalization of the capability to detect input errors.

Definition 6.5 *A checker is TSC if it is fault-secure, self-testing, and code-disjoint.*

A TSC checker usually has two outputs. A (0,0) or (1,1) on its outputs indicates errors, whereas (0,1) or (1,0) indicates fault free operation. The reason we need at least two outputs from the checker is that if it had only one output, the output could get stuck-at the correct value and an error indication would never be given.

The effectiveness of TSC circuits is based on two fundamental assumptions in [SM78]:

A_1) Each failure can be modeled as a member of F;

A_2) Faults occur one at a time, and the time interval between occurrences of any two faults is long enough for all input codewords to be applied the circuit.

6.2.3 General Designs for TSC Circuits

Anderson [And71] gives the model of Figure 6.1 of a TSC circuit consisting of a functional circuit and a checker that are both TSC. In terms of the notation presented here, the functional circuit has a fault-free function which is surjection from an input codeword space X_C onto an output codeword space Z_C, while the checker has a normal input codeword space $X_T = Z_C$ and an output codeword space $Z_T = \{(01), (10)\}$. It is easy to show that the entire circuit is TSC.

Much effort has been concentrated on the designs of TSC checkers by researchers. The most common TSC checker is the two rail checker. The two rail checker is used to compare two words that should normally be complementary.

A simple design of a TSC two-rail checker is shown in Figure 6.2 where each of two input words is two bits. The first input word is (x_0, x_1), and the second

Figure 6.1 A totally self-checking circuit.

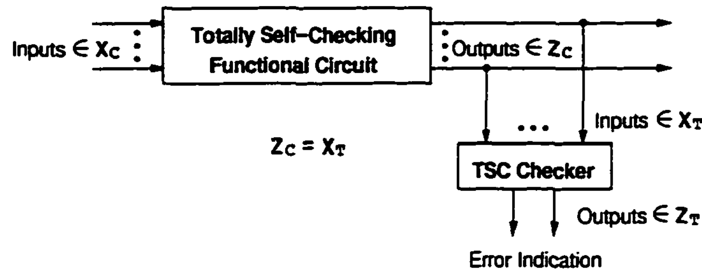
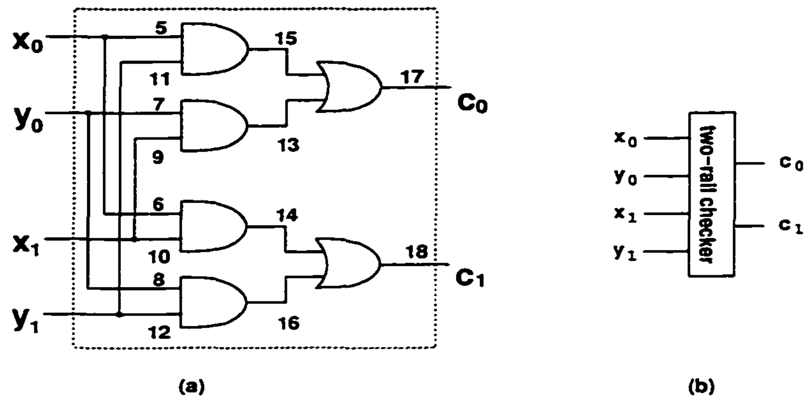


Figure 6.2 A TSC two-rail code checker.



(y_0, y_1). Valid codewords on the inputs have $x_0 \neq y_0$ and $x_1 \neq y_1$. If we consider (x_0, y_0, x_1, y_1) as an input codeword for the given checker, a set of input codewords X is $\{(0101), (0110), (1001), (1010)\}$. In Table 6.1, the detections of this checker for all the possible stuck-at 0 (line/0) and stuck-at 1 (line/1) faults[1] are listed. There are 16 faults in total: 12 of them are detected by only one input, *i.e.*, the detection probability is 0.25, and the remaining four are detected by two input codewords, *i.e.*, the detection probability is 0.5.

Although a two-rail checker for an arbitrary number of input pairs may be designed using two-level AND-OR logic, it is more efficiently realized as a tree by interconnecting the checker modules with two input pairs[AM73].

[1] The fault equivalence rules have been applied.

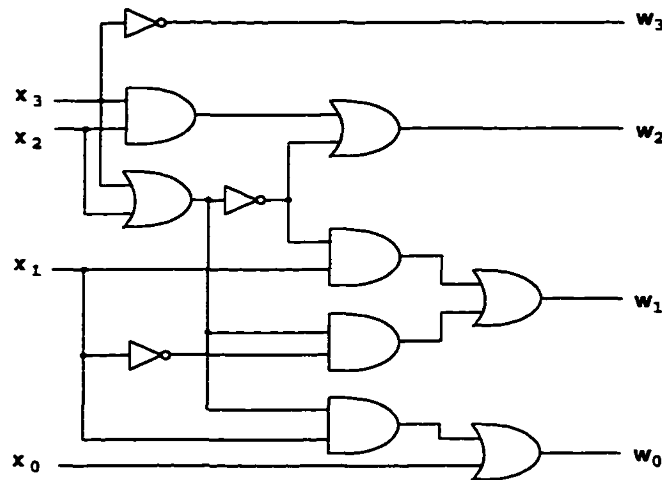
Table 6.1 Detection capability for the two-rail checker in Figure 6.2.

i	fault	0101	0110	1001	1010	*detection*
–	fault-free	01	10	10	01	—
1	5/1	11	10	10	01	0.25
2	6/1	01	11	10	01	0.25
3	7/1	01	10	10	11	0.25
4	8/1	01	10	11	01	0.25
5	9/1	11	10	10	01	0.25
6	10/1	01	10	11	01	0.25
7	11/1	01	10	10	11	0.25
8	12/1	01	11	10	01	0.25
9	13/0	01	00	10	01	0.25
10	14/0	01	10	10	00	0.25
11	15/0	01	10	00	01	0.25
12	16/0	00	10	10	01	0.25
13	17/0	01	00	00	01	0.50
14	17/1	11	10	10	11	0.50
15	18/0	00	10	10	00	0.50
16	18/1	01	11	11	01	0.50

A considerable amount of work has been done in the area of TSC checker design for different codes. However. not as much attention has been paid to the design of functional circuits [JW93]. Since designing a TSC functional circuit is not major issue of this chapter. we employ the simplest way. *i.e.*. a duplex structure. to design a TSC functional circuit. An example used is a BCD-to-excess-3 circuit from [Man91]. depicted in Figure 6.3. and the truth table relating the input and output variables is listed in Table 6.2.

In Figure 6.4. we show a design for a self-checking BCD-to-excess-3 circuit using this duplication structure. It is easy to see that both the functional circuit and the checker are TSC. However, the entire circuit is not TSC because the output codeword

Figure 6.3 A BCD-to-excess-3 circuit.



space Z_C of the functional circuit is a proper subset of the input codeword space X_T of the checker. *i.e.*. $Z_C \subset X_T$ ($|Z_C| = 10$ and $|X_T| = 16$). resulting in some faults in the checker not being detectable by only using the codewords from the input codeword space of the entire circuit.

It seems that the design in Figure 6.4 satisfies the TSC goal because the first erroneous output of the functional circuit is caught by the checker (if the checker works correctly). However. if the checker has a fault which is not detectable (by only using the codewords from the input codeword space of the entire circuit). the erroneous output from the functional circuit may not be reported. Perhaps a special design for such a circuit can be found to achieve the TSC goal (*e.g.*. the entire circuit is TSC). But. the checker may have a fault which cannot be detected before the first erroneous output occurs in the functional circuit. *i.e.*. one of the two fundamental assumptions. A_2. is not realistic.

In practice. a structure similar to that in Figure 6.4 is often used to design self-checking circuits with the property that both the functional circuit and the checker are TSC [JW93. ABF90]. But the question now arises to how we can evaluate

Table 6.2 Truth table for the BCD-to-excess-3 code conversion.

Input BCD				Output Excess-3 code			
x_0	x_1	x_2	x_3	w_0	w_1	w_2	w_3
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0

the performance for such a design with respect to the concurrent error detecting capability. In the following sections. we discuss the existing evaluation methods and propose the fail-safe model.

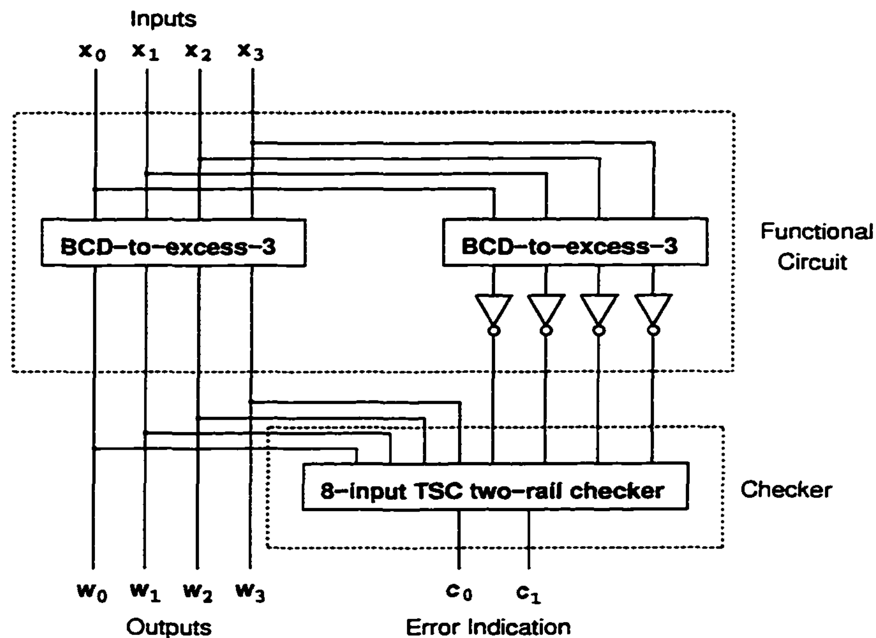
It should be noted that the circuit in Figure 6.4 is used solely as an illustration and the results derived later in Chapter 7 are applicable to any circuit which follows the same structural model.

6.3 Existing Evaluations of Self-Checking Circuits

6.3.1 Related Work

Only a handful of works with respect to the performance of self-checking circuits have been reported in the last ten years. Lu and McCluskey in [LM84] propose *quantitative measures* of self-checking power for evaluation, comparison. and design

Figure 6.4 A design for a self-checking BCD-to-excess-3 circuit.



of self-checking circuits. Fujiwara *et al.* in [FMM84. FM87] suggest a measure called *checker fault coverage* to describe the fault detection capability of a TSC circuit. Lo and Fujiwara in [LF93] deliver a *probabilistic measurement* for self-checking circuits. We give brief reviews of these existing evaluations below.

Quantitative measures of the self-testing and fault-secure properties are defined to allow comparisons between various self-checking implementations of a circuit function. These measures are the *testing input fraction* (TIF) and the *secure input fraction* (SIF) [LM84]. The values of TIF and SIF can be averaged over the fault set F to provide figures of merit, which are named as the TIFBAR and SIFBAR. By the above evaluation. it can be seen that the circuit is fault-secure if the $SIFBAR = 1$. However. it should be noted that if the $TIFBAR$ of circuit A is greater than that of circuit B. it does not mean that if circuit B is self-testing. then circuit A is also self-testing.

In general. the self-checking circuit consists of two parts: a functional circuit

and a checker. Fujiwara *et al.* in [FMM84. FM87] suggest a measure called *checker fault coverage.* defined as the ratio of the number of single stuck-at faults in both the functional circuit and the checker which are detected with at least one codeword from the functional circuit to the total number of single stuck-at faults in both the functional circuit and the checker. In short. it is a fault coverage on using the codewords from the input codeword space of the entire circuit. Clearly. if a circuit has 100% *checker fault coverage.* it must be self-testing.

In [LF93]. Lo and Fujiwara introduce a measurement named Probability of Achieving TSC Goal (PATG). which is the probability that a circuit still possesses concurrent error detecting capability and guarantees that no error is propagated. The PATG is defined as the probability that the circuit has no fault and that the circuit has a fault which is detected before a second fault occurs. The PATG gives the worst case probability. A TSC circuit may be still achieving the TSC goal with the accumulation of two faults. When a second fault occurs before the first fault is detected. the circuit is *unsafe* but the actual violation of the TSC goal has to wait until an appropriate input to provoke that [LF93].

6.3.2 Remarks

In short. the existing evaluations are based on simulation and statistics. To a certain extent. they can be applied to compare how two different designs meet the TSC Goal. The secure input fraction is a quantitative measure for the fault-secure property. and the checker fault coverage. the testing input fraction. and the PATG are quantitative measures for the self-testing property. A major problem of the above evaluations is that they compute a single value. which is not a function of t (time). for the self-checking circuit under normal operation.

Since the circuit is tested during normal operation. we should consider the following points:

(a) A fault may occur at any moment in the functional circuit or the checker under normal operation;

(b) The checker has different abilities of detecting errors when it has i faults, $0 \leq i \leq m$, where m is the number of faults considered in the checker.

These lead us to propose a *fail-safe* evaluation in the following chapter, based on a Markov model [Joh89].

Chapter 7

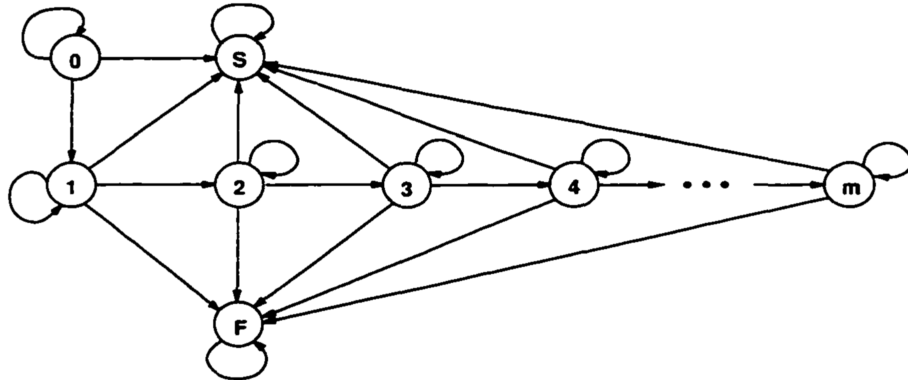
Fail-Safe Evaluation for Self-Checking Circuits

7.1 Fail-Safe Evaluation

Before introducing our evaluation, we should note that our work is concerned with the fail-safe evaluation for the self-checking circuits proposed by researchers working in the digital circuit testing field. Here, the safety measure is different from the steady-state measures in [GLT87, TMWH92]. To avoid a misunderstanding of our measure, we cite a definition of safety from [Joh89].

Safety is the probability that a system either performs its functions correctly or discontinues the function in a manner that causes no harm [Joh89]. To improve the safety of a circuit, a general approach is to design the circuit which can be tested under normal operation, *i.e.*, the self-checking circuit. However, without a general analysis for the safety issue about the self-checking circuit, it may not be easy to say how much safety is improved for the given self-checking circuit.

Figure 7.1 A generic Markov model for the self-checking circuit in Figure 6.4.



7.1.1 Generic Markov Model

In general, a self-checking circuit consists of two parts: a functional circuit and a checker. Without loss of generality, we assume that both the functional circuit and the checker are TSC respectively. Based on this assumption, we want to show that even if we put our best effort into designing the self-checking circuit, we still have to consider the *unsafe* case for the circuit under normal operation. Note that when the functional circuit works properly, we still can get correct outputs even if there are some faults in the checker. However, when the functional circuit has a fault resulting in an erroneous output, the circuit fails if the checker cannot find it (it only happens when the checker already has some faults.)

For a theoretical analysis, the entire circuit is regarded as a system with $|F| = n + m$ components, n components in the functional circuit and m components in the checker. Each component in the functional circuit has a unique failure rate λ_C and each in the checker has a unique failure rate λ_T. Each fault in the functional circuit (the checker) corresponds to the failure of one component in the functional circuit (the checker). A failure of any component in the functional circuit results in the system failure if it cannot be detected by the checker, whereas a failure of any component in the checker only influences the detecting ability of the checker in

finding any fault. which results in an erroneous output. in the functional circuit. For convenience. we still use the *fault occurring* instead of the the *component failure*. Theoretically. we can define a Markov model as shown in Figure 7.1 to describe the circuit depicted in Figure 6.4. The model has the following states.

state 0. both the functional circuit and the checker are working correctly.

state i. $1 \leq i \leq m$. the checker has i faults which have not been detected by the checker. where m is the number of faults considered in the checker.

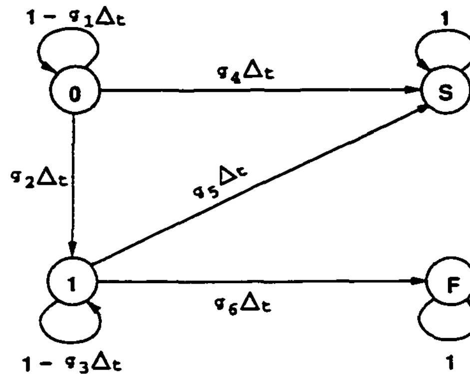
state S. there is one fault in the functional circuit or some faults in the checker. which has(have) been detected by the checker.

state F. the functional circuit has one fault. resulting in an erroneous output which is not detected by the checker.

As we know. if both the functional circuit and the checker are TSC. an erroneous output which results from a fault in the functional circuit must be detected by the checker in **state 0**. The system goes to the safe state. *i.e.*. **state S**. However. if the system is in **state i**. an erroneous output from the functional circuit may or may not be detected by the checker. As a result. the system may go to the failed state. *i.e.*. **state F**.

To build equations corresponding to the model in Figure 7.1. we need to work out the state transition probabilities. As the number of faults increases. we have great difficulty in precisely evaluating the abilities of detecting any fault in the checker and erroneous output from the functional circuit in **state i**. $0 \leq i \leq m$. However. we can simplify the model by making a number of assumptions so that it is amenable to analysis. In so doing it is critical that we ensure that any such assumptions can be proved to lead to results that are consistently pessimistic rather than optimistic - in this manner we can ensure that our results never lead to false optimism. but that

Figure 7.2 A simplified Markov model for the self-checking circuit in Figure 6.4.



they are slightly more pessimistic than the actual performance. This is explored in the section below.

7.1.2 Simplified Markov Model

The simplified Markov model resulting from that in Figure 7.1 is shown in Figure 7.2. with the following assumptions:

(a) The fault in the checker can only be detected at the moment of its occurrence:

(b) When the checker has the second fault in **state 1**. the system goes to **state S** if the fault is detected or **state F** if the fault is not detected.

The assumptions are pessimistic because if the fault in the checker is detectable by using the codewords from the input codeword space of the entire circuit. it may be detected not only at the moment of its occurrence. but also at any moment before an erroneous output of the functional circuit occurs.

The system is assumed to begin in the *fault-free* state. *i.e.*, **state 0**. There are two paths by which the system can exit **state 0**. The first is shown as a transition

to **state S** and corresponds to the checker detecting the erroneous output from the functional circuit or a fault occurring in the checker. The second transition that can occur from **state 0** is the transition to **state 1**, which corresponds to a fault occurring in the checker and going undetected at the moment of the fault occurrence.

Similar transitions exist from **state 1** to **state S**, the safe state, and **state F**, the failed state. **state 1** corresponds to the checker having one fault which has not been detected. While in **state 1**, if the checker can detect either a fault occurring in the functional circuit which results in the erroneous output or the second fault occurring in the checker, the system goes to **state S**. If the checker cannot detect the erroneous output from the functional circuit resulting from the fault occurring, or the second fault occurring in the checker, the system goes to the failed state, **state F**.

Based on the above discussion, we define g_i, $1 \leq i \leq 6$, in Figure 7.2 as follows.

$$\begin{aligned}
g_2 &= (1 - p_x C_{T_1}) m \lambda_T, \\
g_4 &= n\lambda_C + p_x C_{T_1} m \lambda_T, \\
g_5 &= n C_C \lambda_C + (m-1) p_x C_{T_2} \lambda_T, \\
g_6 &= n(1 - C_C)\lambda_C + (m-1)(1 - p_x C_{T_2})\lambda_T, \\
g_1 &= g_2 + g_4, \\
g_3 &= g_5 + g_6,
\end{aligned}$$

where

- n is the number of faults considered in the functional circuit.
- m is the number of faults considered in the checker.
- λ_C is the unique rate of the fault occurring in the functional circuit.
- λ_T is the unique rate of the fault occurring in the checker.

- p_x is the probability that the functional circuit produces all input codewords to the checker. which is calculated by the number of output codewords of the functional circuit over the number of input codewords to the checker.

- C_{T_1} (C_{T_2}) is the probability of detecting the first (second) fault in the checker when all possible codewords are applied. Computing C_{T_1}. for each fault in the checker. by applying all input codewords. we obtain the detection probability which is similar to that shown in Table 6.1. Then. C_{T_1} is calculated by the sum of the detection probabilities over the number of faults considered. For the checker shown in Figure 6.2. we have

$$\begin{aligned} C_{T_1} &= (0.25 \times 12 + 0.50 \times 4)/16 \\ &= 0.31. \end{aligned}$$

  In a similar fashion. we can obtain $C_{T_2} = 0.44$ for the checker in Figure 6.2 by considering two faults in the checker each time.

- C_C is the probability that the checker detects an erroneous output from the functional circuit when the checker has one fault. Computing C_C is similar to computing C_{T_1}. However. in this case. we apply all input codewords with one bit error instead of all input codewords without error. For the checker shown in Figure 6.2. we have $C_C = 0.75$. That is. if the checker has one fault. it only has a probability of 0.75 to catch the erroneous output from the functional circuit. Moreover. for this example. if the checker has two faults. the probability is reduced to 0.61. Note that when the checker has no fault. the probability is 1.0. *i.e.*, any erroneous output from the functional circuit is detectable by the checker.

For the Markov model in Figure 7.2, we have the following equations.

$$p_0(t + \Delta t) = (1 - g_1 \Delta t) p_0(t),$$

$$\begin{aligned}
p_1(t+\Delta t) &= g_2\Delta t p_0(t) + (1 - g_3\Delta t)p_1(t), \\
p_S(t+\Delta t) &= g_4\Delta t p_0(t) + g_5\Delta t p_1(t) + p_S(t), \\
p_F(t+\Delta t) &= g_6\Delta t p_1(t) + p_F(t).
\end{aligned}$$

The Markov model considered is a *discrete-time* model in which state transitions occur at fixed intervals Δt. The continuous-time equations can be derived from the discrete-time equations by allowing the time interval Δt to approach zero. The equations of the discrete-time Markov model can be rewritten as

$$\begin{aligned}
\frac{p_0(t+\Delta t) - p_0(t)}{\Delta t} &= -g_1 p_0(t), \\
\frac{p_1(t+\Delta t) - p_1(t)}{\Delta t} &= g_2 p_0(t) - g_3 p_1(t), \\
\frac{p_S(t+\Delta t) - p_S(t)}{\Delta t} &= g_4 p_0(t) + g_5 p_1(t), \\
\frac{p_F(t+\Delta t) - p_F(t)}{\Delta t} &= g_6 p_1(t).
\end{aligned}$$

Taking the limit as Δt approaches zero results in a set of differential equations given by

$$\begin{aligned}
\frac{dp_0(t)}{dt} &= -g_1 p_0(t), \\
\frac{dp_1(t)}{dt} &= g_2 p_0(t) - g_3 p_1(t), \\
\frac{dp_S(t)}{dt} &= g_4 p_0(t) + g_5 p_1(t), \\
\frac{dp_F(t)}{dt} &= g_6 p_1(t).
\end{aligned}$$

Using Laplace transforms, we have

$$\begin{aligned}
sP_0(s) - p_0(0) &= -g_1 P_0(s), \\
sP_1(s) - p_1(0) &= g_2 P_0(s) - g_3 P_1(s), \\
sP_S(s) - p_S(0) &= g_4 P_0(s) + g_5 P_1(s), \\
sP_F(s) - p_F(0) &= g_6 P_1(s).
\end{aligned}$$

We assume in this analysis, that the system starts in the perfect state, *i.e.*, **state 0**, at time $t = 0$, so $p_0(0) = 1$, and $p_1(0) = p_S(0) = p_F(0) = 0$. Consequently, the Laplace transform equations can be written as

$$\begin{aligned}
P_0(s) &= \frac{1}{s+g_1}, \\
P_1(s) &= \frac{g_2}{g_3-g_1}\left(\frac{1}{s+g_1} - \frac{1}{s+g_3}\right), \\
P_S(s) &= \frac{1}{g_1}\left(g_4 + \frac{g_2 g_5}{g_3-g_1}\right)\left(\frac{1}{s} - \frac{1}{s+g_1}\right) - \frac{g_2 g_5}{g_3(g_3-g_1)}\left(\frac{1}{s} - \frac{1}{s+g_3}\right), \\
P_F(s) &= \frac{g_2 g_6}{g_3-g_1}\left[\frac{1}{g_1}\left(\frac{1}{s} - \frac{1}{s+g_1}\right) - \frac{1}{g_3}\left(\frac{1}{s} - \frac{1}{s+g_3}\right)\right].
\end{aligned}$$

Taking the inverse Laplace transform results in the solution which is given by

$$\begin{aligned}
p_0(t) &= e^{-g_1 t}, \\
p_1(t) &= \frac{g_2}{g_3-g_1}\left(e^{-g_1 t} - e^{-g_3 t}\right), \\
p_S(t) &= \frac{1}{g_1}\left(g_4 + \frac{g_2 g_5}{g_3-g_1}\right)\left(1 - e^{-g_1 t}\right) - \frac{g_2 g_5}{g_3(g_3-g_1)}\left(1 - e^{-g_3 t}\right), \\
p_F(t) &= \frac{g_2 g_6}{g_3-g_1}\left[\frac{1}{g_1}\left(1 - e^{-g_1 t}\right) - \frac{1}{g_3}\left(1 - e^{-g_3 t}\right)\right].
\end{aligned}$$

The safety of the system is written as

$$S(t) = p_0(t) + p_1(t) + p_S(t). \tag{7.1}$$

Meanwhile, we can get a side product from our analysis, the reliability of the system which is given by

$$R(t) = p_0(t) + p_1(t). \tag{7.2}$$

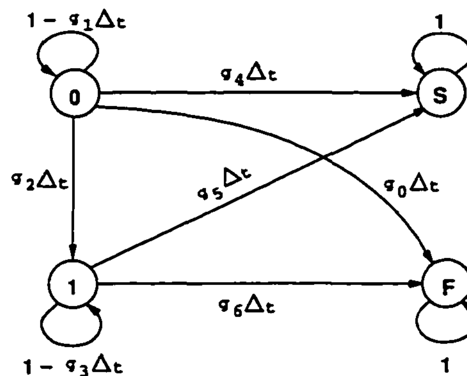
Note that although we use the circuit in Figure 6.4 to derive equations (7.1) and (7.2), in fact, they are applicable to all circuits with a similar structure to that in Figure 6.4.

7.1.3 Generalizations of Parameters and Model

In the previous subsection, when computing C_{T_1}, C_{T_2}, and C_C, we assume that the output codewords produced by the functional circuits are uniformly distributed. In general, it may not be true, *i.e.*, some output codewords may appear more frequently than others. Therefore, we should consider using all output codewords produced by applying all inputs to the entire circuit to get the detecting capability of C_{T_1} and C_{T_2}. For computing C_C, considering all possible combinations of two faults, one in the functional circuit and another in the checker, we can get C_C using results of the fault simulation. Note that, if we apply the above methods to get C_{T_1}, C_{T_2}, and C_C, we do not need parameter p_x.

Furthermore, even if we consider a more general case for the self-checking circuit (*e.g.*, the functional circuit is not *fault-secure*), we still can estimate the safety of the circuit by modifying the model in Figure 7.2 into one shown in Figure 7.3, where a transition from **state 0** to **state F** results from the presence of one fault in the functional circuit producing an incorrect output codeword that cannot be detected by the checker.

Figure 7.3 A simplified Markov model for the self-checking circuit in general.



Since we use a different approach to get parameters C_{T_1}, C_{T_2}, and C_C, and

examine the general case, we redefine g_i, $0 \leq i \leq 6$, in Figure 7.3 as follows.

$$\begin{aligned}
g_0 &= n(1 - C_{C_0})\lambda_C, \\
g_2 &= (1 - C_{T_1})m\lambda_T, \\
g_4 &= nC_{C_0}\lambda_C + C_{T_1}m\lambda_T, \\
g_5 &= nC_C\lambda_C + (m-1)C_{T_2}\lambda_T, \\
g_6 &= n(1 - C_C)\lambda_C + (m-1)(1 - C_{T_2})\lambda_T, \\
g_1 &= g_0 + g_2 + g_4, \\
g_3 &= g_5 + g_6,
\end{aligned}$$

where m, n, λ_C, λ_T, C_{T_1}, C_{T_2}, and C_C are the same as those in Section 7.1.2 but we use the methods discussed in this subsection to obtain C_{T_1}, C_{T_2}, and C_C. C_{C_0} is the probability that the checker without faults detects an erroneous output from the functional circuit, which can be obtained by simulating faults one by one in the functional circuit. If both the functional circuit and the checker are TSC, then C_{C_0} is 1, with the result that the model in Figure 7.3 degenerates to the one in Figure 7.2.

For the Markov model in Figure 7.3, we have the following equations.

$$\begin{aligned}
p_0(t + \Delta t) &= (1 - g_1\Delta t)p_0(t), \\
p_1(t + \Delta t) &= g_2\Delta t p_0(t) + (1 - g_3\Delta t)p_1(t), \\
p_S(t + \Delta t) &= g_4\Delta t p_0(t) + g_5\Delta t p_1(t) + p_S(t), \\
p_F(t + \Delta t) &= g_0\Delta t p_0(t) + g_6\Delta t p_1(t) + p_F(t).
\end{aligned}$$

In a similar method to that used to solve the equations in Section 7.1.2, we have

$$\begin{aligned}
p_0(t) &= e^{-g_1 t}, \\
p_1(t) &= \frac{g_2}{g_3 - g_1}(e^{-g_1 t} - e^{-g_3 t}),
\end{aligned}$$

$$p_S(t) = \frac{1}{g_1}(g_4 + \frac{g_2g_5}{g_3 - g_1})(1 - e^{-g_1t}) - \frac{g_2g_5}{g_3(g_3 - g_1)}(1 - e^{-g_3t}).$$
$$p_F(t) = \frac{1}{g_1}(g_0 + \frac{g_2g_6}{g_3 - g_1})(1 - e^{-g_1t}) - \frac{g_2g_6}{g_3(g_3 - g_1)}(1 - e^{-g_3t}).$$

Then, we can derive equations for the safety and reliability which are similar to (7.1) and (7.2).

7.2 Experimental Results and Comparisons

It is instructive to compare the different evaluations by experimental results. Since the main purpose of the comparisons is to see which evaluation is practical, it is sufficient to use the circuit in Figure 6.4 as an example.

7.2.1 Results

For the circuit shown in Figure 6.4, the functional circuit has 56 faults ($n = 56$) and the checker has 48 faults ($m = 48$). We calculate results from the existing evaluation methods in abbreviated form to give us some comparison with the fail-safe method. The complete details of these methods are in [LM84, FMM84, FM87, LF93].

(a) quantitative measures in [LM84].

The average *testing input fraction* ($TIFBAR$) and the average *secure input fraction* ($SIFBAR$) for the circuit shown in Figure 6.4 are given as follows.

$$TIFBAR = \frac{1}{104}(34.6) = 0.33,$$
$$SIFBAR = \frac{1}{104}(104 \times 1) = 1.$$

It can be seen that the circuit is fault-secure because $SIFBAR = 1$. However, we do not know how to precisely explain the result $TIFBAR = 0.33$ because if the $TIFBAR$ of circuit A is greater than that of circuit B, it does not mean that if circuit B is self-testing, then circuit A is also self-testing.

(b) checker fault coverage in [FMM84, FM87].

The total number of faults in the circuit is 104, and 101 of them are detected by at least one input codeword and 3 of them are undetectable. Thus,

$$\text{checker-fault-coverage} = \frac{101}{104} \times 100\% = 97.12\%.$$

This means that the circuit is not self-checking.

(c) probability measurement in [LF93].

Assuming failure rate $\lambda_g = 10^{-5}$, we have

$$PATG = 99.93\%.$$

That is, the probability that the circuit has no fault and the circuit has a single fault which is detected before a second fault occurs is 99.93%.

(d) fail-safe evaluation.

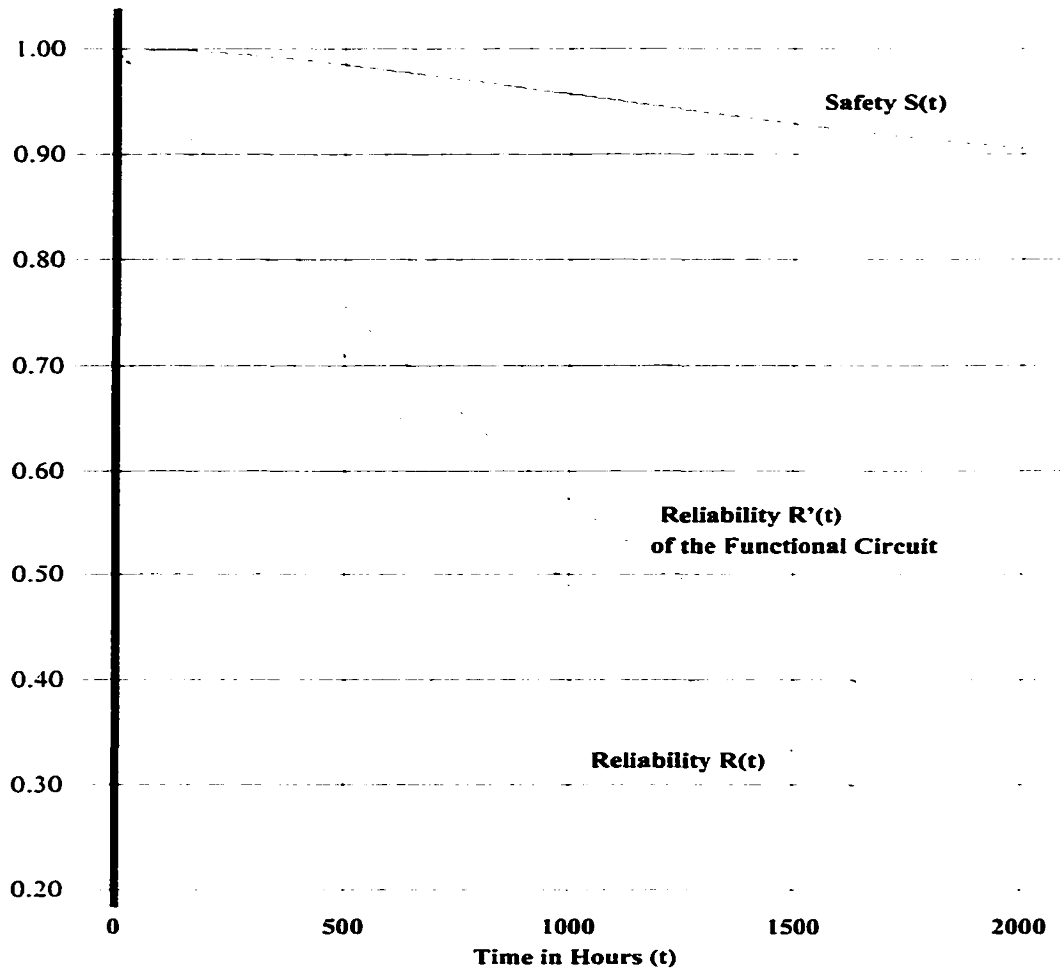
For the circuit shown in Figure 6.4, we have $n = 56$ and $m = 48$. By simulation and observation, we have $p_x = 10/16 = 0.63$, $C_C = 0.83$, $C_{T_1} = 0.31$, and $C_{T_2} = 0.46$. By (7.1) and (7.2), the safety $S(t)$ and the reliability $R(t)$ are shown in Figure 7.4, where $\lambda_C = \lambda_T = 10^{-5}$ failures per hour.

Compared with the reliability, $R'(t) = e^{-n\lambda_C t}$, of the functional circuit without the checker, it can be seen that using the checker can improve the safety but reduce the reliability of the system because the system goes to the safe state if it detects any fault in the checker although the fault in the checker does not influence the output of the functional circuit. Moreover, compared with the reliability of the functional circuit, the reliability of the original circuit (BCD-to-excess-3) is $e^{-\frac{n}{2}\lambda_C t} = [R'(t)]^{\frac{1}{2}}$, which is also the safety of the original circuit.

7.2.2 Remarks

A significant difference between the fail-safe evaluation and the existing evaluations is that the former estimates the safety, which decreases as time goes on, of the

Figure 7.4 Safety and reliability for the circuit in Figure 6.4 with $\lambda_C = \lambda_T = 10^{-5}$.



circuit under normal operation rather than giving a single value. However, some of the results from the existing evaluations are still needed in the fail-safe evaluation. In other words, the fail-safe evaluation incorporates the property that the self-checking circuit is tested under normal operation with the existing evaluations on the basis of the Markov model.

7.3 Summary

We have reviewed the existing evaluations concerning satisfying the TSC goal and proposed the fail-safe evaluation based on a Markov model. The proposed evaluation can be used to predict the safety of the system (the entire circuit). Compared with the existing evaluations. our analysis is more practical since it considers the testing procedure under normal operation of the circuit.

In our analysis. we note that when designing a self-checking circuit. we should not assume that the TSC checker can detect all errors produced by the functional circuit because the functional circuit may not produce all input codewords for the checker such that some faults in the checker may not be detectable by only using the input codewords for the whole circuit. Even for the TSC circuit. there still exists some possibility that the error produced by the functional circuit cannot be detected if there is/are some fault(s) in the checker which cannot be detected before the fault occurs in the functional circuit.

We have given a general idea as to the use of the Markov model to predict the safety of the self-checking circuit. It is thought that this is a very promising approach to the derivation of an effective measure of the safety of such a circuit. and research is continuing to minimize the pessimistic assumptions so that the results of the evaluations are more accurate for the self-checking circuit.

Self-checking circuits can detect the presence of both transient and permanent faults. Since the existing evaluation methods for the self-checking circuits are based on the stuck-at fault model. our current work is to use the same fault model but apply a different evaluation approach and do some comparisons between our method and the existing evaluations. According to field measurements on several operational computer systems, considerable detected errors/failures are transient in nature [SKM+78. MST79. IBM82]. Therefore, we will work on the evaluation for the transient faults in future.

It should be noted that in general it is not easy to find relationships between the fault models and physical defects in the circuit under test. Consequently, the result of the evaluation is not an exact but a rough estimate.

Chapter 8

Conclusions

8.1 Summary

8.1.1 Major Contributions

This dissertation discusses issues of evaluation and performance improvement in a BIST environment. The following contributions are:

(1) We have developed a method of analyzing the effectiveness of a test vector generator for detecting delay faults. Most of the results presented in Chapter 2, 3, and 4 are published in [ZBMM94a, ZBMM94b, ZBMM95, ZMM95b].

The comprehensive analysis regarding the two-vector testing capabilities of LFSR and LHCA plays a key role in analyzing delay fault testing. We solved the open problem as to "how to properly separate the inputs when an LHCA-based generator is used for testing the delay faults" proposed in [Sav95, Sav97].

(2) We have proposed a transition delay fault model for look-up table based circuits and formalized the technology mapping problem in Chapter 5. Some of the results are presented in [ZMM96], which solves an open problem listed in [FS94, CD94].

(3) We have identified a problem with evaluating self-checking circuits and proposed a fail-safe evaluation. Our initial results were published in [ZM95].

To produce the experimental results presented in this dissertation. we implemented a transition fault simulation program for gate and LUT-based combinational circuits. We also implemented a technology mapping program used to map the ISCAS85 benchmark circuits to the 5-input LUT-based circuits for our experimental results regarding transition faults for the LUT-based circuits.

8.1.2 Remarks

Note that some researchers presented similar results coincident with our work. Chen and Gupta in [CG96] list equations of the transition coverage for the LFSR and LHCA generators with different formats from the results presented in [ZBMM94b. ZBMM95] (Also *see* Chapter 3).

Lo and Fujiwara present results on the probability of achieving the TSC goal in [LF96]. where they improve their results presented in [LF93] and provide a means of dynamic error handling performance evaluation of self-checking circuits using a different approach from the one presented in [ZM95] (*see* Chapter 7).

8.2 Further Work

Application of the theory derived in this dissertation to further explore the world of BIST is a general research direction. The following areas may be a good starting point for further research.

(1) Work in determining the proper connection between the circuit under test and the test vector generator may be done by considering ways to characterize circuit topology.

(2) From our experimental results, a different initial seed for the generator determines a segment of the whole test sequence generated which may have different fault coverage. Moreover, a test generator with an appropriate initial seed may detect the faults with a shorter sequence. Research work is still required to find the appropriate initial seed.

(3) We need to find a good algorithm for determining the untestable faults in LUT-based circuits. Moreover, when mapping a circuit to the LUT-based circuit, we should minimize the untestable faults.

Bibliography

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

[AKS93] V.D. Agrawal, C.R. Kime, and K.K. Saluja. A tutorial on built-in self-test (Part 1: Principles). *IEEE Design and Test of Computers,* pages 73–82, March 1993.

[AM73] D.A. Anderson and G. Metze. Design of totally self-checking circuits for m-out-of-n codes. *IEEE Transactions on Computers,* 22(3):263–269, March 1973.

[And71] D.A. Anderson. Design of self-checking digital networks using coding techniques. *Coordinated Sci. Lab., Univ. Illinois, Urbana, Report R-527,* September 1971.

[Avr94] L. Avra. Synthesis techniques for built-in self-testable designs. *Ph.D. Dissertation, Department of Electrical Engineering and Computer Science, Stanford University,* June 1994.

[Bar92a] P.H. Bardell. Discrete logarithms: a parallel pseudorandom pattern generator analysis method. *Journal of Electronic Testing: Theory and Applications,* 3(1):17–31, 1992.

[Bar92b] P.H. Bardell. Primitive polynomials of degree 301 through 500. *Journal of Electronic Testing: Theory and Applications,* 3(2):175–176, 1992.

[BF85] F. Brglez and H. Fujiwara. A neutral netlist of 10 combinational benchmark circuits and a target translator in FORTRAN. In *Proceedings of IEEE International Symposium on Circuits and Systems,* pages 663–698, 1985.

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

[Bry86] R.E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, 35(8):677-691, August 1986.

[Cat95] K. Cattell. Characteristic polynomials of one-dimensional linear hybrid cellular automata. *Ph.D. Dissertation, Department of Computer Science, University of Victoria, Victoria, BC, Canada*, May 1995.

[CD94] J. Cong and Y. Ding. On nominal delay minimization in LUT-based FPGA technology mapping. *INTEGRATION, the VLSI Journal*, 18:73–94, 1994.

[CG96] C.-A. Chen and S.K. Gupta. BIST test pattern generators for two-pattern testing – theory and design algorithms. *IEEE Transactions on Computers*, 45(3):257–269, March 1996.

[CLR90] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms*. The MIT Press, 1990.

[CM91] K. Cattell and J.C. Muzio. A linear cellular automata algorithm: algorithm summary. In *Proceedings of the 5th Technical Workshop on New Directions for IC Testing*, August 1991.

[CM96] K. Cattell and J.C. Muzio. Synthesis of one-dimensional linear hybrid cellular automata. *IEEE Transactions on Computer-Aided Design*, 1996.

[Coc75] J. Cocking. RAM test patterns and test strategy. In *Proceedings of Semicond. Test Symposium*, pages 1–8, October 1975.

[CZ95] K. Cattell and S. Zhang. Minimal cost one-dimensional linear hybrid cellular automata of degree through 500. *Journal of Electronic Testing: Theory and Applications*, 6(2):255–258, April 1995.

[DGD+90] A.K. Das, A. Ganguly, A. Dasgupta, S. Bhawmik, and P.P. Chaudhuri. Efficient characterisation of cellular automata. *IEE Proceedings - Computers and Digital Techniques*, 137(1):81–87, January 1990.

[FM87] E. Fujiwara and K. Matsuoka. A self-checking generalized prediction checker and its use for built-in testing. *IEEE Transactions on Computers*. 36(1):86–93. January 1987.

[FM91] K. Furuya and E.J. McCluskey. Two-pattern test capabilities of autonomous TPG circuits. In *Proceedings of IEEE International Test Conference*. pages 704–711. 1991.

[FMM84] E. Fujiwara. N. Mutoh. and K. Matsuoka. A self-testing group-parity prediction checker and its use for built-in testing. *IEEE Transactions on Computers*. 33(6):146–153. June 1984.

[FS94] A. H. Farrahi and M. Sarrafzadeh. Complexity of the lookup-table minimization problem for FPGA technology mapping. *IEEE Transactions on Computer-Aided Design*. 13(11):1319–1332. November 1994.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability - A Guide to the Theory of NP-Completeness*. Freeman. 1979.

[GLT87] A. Goyal. S.S. Lanvenberg, and K.S. Trivedi. Probabilistic modeling of computer system availability. *Annals of Operations Research*. 8:285–306. March 1987.

[HK93] J. Hartmann and G. Kemnitz. How to do weighted random testing for BIST. In *Proceedings of IEEE International Conference on Computer-Aided Design*. pages 568–571. November 1993.

[HMC89] P.D. Hortensius. R.D. McLeod. and H.C. Card. Parallel random number generation for VLSI systems using cellular automata. *IEEE Transactions on Computers*. 38(10):1466–1472. October 1989.

[HMP+89] P.D. Hortensius. R.D. McLeod. W. Pries. D.M. Miller. and H.C. Card. Cellular automata-based pseudorandom number generators for built-in self-test. *IEEE Transactions on Computer-Aided Design*. 8(8):842–859. August 1989.

[HRT+95] S. Hellebrand, J. Rajski. S. Tarnick, S. Venkataraman. and B. Courtois. Built-in test for circuits with scan based on reseeding of multiple-polynomial linear feedback shift registers. *IEEE Transactions on Computers*, 42(2):223–233, February 1995.

[Hur98] S.L. Hurst. *VLSI Testing - digital and mixed analogue/digital techniques.* The Institution of Electrical Engineers. 1998.

[IBM82] R.K. Iyer. S.E. Butner. and E.J. McCluskey. A statistical failure/load relationship: results of a multicomputer study. *IEEE Transactions on Computers.* 31(7):697–705. July 1982.

[Joh89] B.W. Johnson. *Design and Analysis of Fault Tolerant Digital Systems.* Addison-Wesley. 1989.

[JW93] N.K. Jha and S.J. Wang. Design and synthesis of self-checking VLSI circuits. *IEEE Transactions on Computer-Aided Design.* 12(6):878–887. June 1993.

[Kad93] F. Kadri. Enhancing transition fault coverage in built-in self-test. *M.Sc. Thesis. Department of Computer Science. University of Victoria. Victoria. BC. Canada.* May 1993.

[KT94] D. Kagaris and S. Tragoudas. A design for testability technique for test pattern generation with LFSRs. In *Proceedings of IEEE VLSI Test Symposium.* pages 68–73. April 1994.

[LF93] J.C. Lo and E. Fujiwara. A probabilistic measurement for totally self-checking circuits. In *Proceedings of International Workshop on Defect and Fault Tolerance in VLSI Systems.* pages 262–270. October 1993.

[LF96] J.C. Lo and E. Fujiwara. Probability to achieve tsc goal. *IEEE Transactions on Computers.* 45(4):450–460. April 1996.

[LGB94] M. Lempel. S.K. Gupta. and M.A. Breuer. Test embedding with discrete logarithms. In *Proceedings of IEEE VLSI Test Symposium.* pages 74–80. April 1994.

[LM84] D.J. Lu and E.J. McCluskey. Quantitative evaluation of self-checking circuits. *IEEE Transactions on Computer-Aided Design.* 3(4):150–155. April 1984.

[Man91] M.M. Mano. *Digital Logic.* Second Edition. Prentice-Hall. Inc.. 1991.

[Mil93] D.M. Miller. Multiple-valued logic design tools. In *Proceedings of IEEE International Symposium on Multiple-Valued Logic*, pages 1–10, May 1993.

[MMR94] S.K. Mukund, E.J. McCluskey, and T.R.N. Rao. An apparatus for pseudo-deterministic testing. Technical Report CRC No. 94-12, Center for Reliable Computing, Computer Systems Laboratory, Stanford University, October 1994.

[MST79] S.R. McConnel, D.P. Siewiorek, and M.M. Tsao. The measurement and analysis of transient errors in digital computer systems. In *Proceedings of IEEE International Symposium on Fault-Tolerant Computing*, pages 67–70, 1979.

[MV95] A. Majumdar and S.B.K. Vrudhula. Fault coverage and test length estimation for random pattern testing. *IEEE Transactions on Computers*, 44(2):234–247, February 1995.

[NVCC94] S. Nandi, B. Vamsi, S. Chakraborty, and P.P. Chaudhuri. Cellular automata as a BIST structure for testing CMOS circuits. *IEE Proceedings – Computers and Digital Techniques*, 141(1):41–47, January 1994.

[PNK+94] J. Park, M. Naivar, R. Kapur, M.R. Mercer, and T.W. Williams. Limitations in predicting defect level based on stuck-at fault coverage. In *Proceedings of IEEE VLSI Test Symposium*, pages 186–191, April 1994.

[PR94] I. Pomeranz and S. M. Reddy. On testing delay faults in macro-based combinational circuits. In *Proceedings of IEEE International Conference on Computer-Aided Design*, pages 332–339, 1994.

[Sav92] J. Savir. Developments in delay testing. In *Proceedings of IEEE VLSI Test Symposium*, pages 247–253, 1992.

[Sav95] J. Savir. Generator choices for delay test. In *Proceedings of the 4th Asian Test Symposium*, pages 214–221, 1995.

[Sav97] J. Savir. Delay test generation: a hardware perspective. *Journal of Electronic Testing: Theory and Applications*, pages 245–254, 1997.

[SB92] J. Savir and R. Berry. AC strength of a pattern generator. *Journal of Electronic Testing: Theory and Applications*, 3(2):119–125, May 1992.

[SKM+78] D.P. Siewiorek, V. Kini, H. Mashburn, S.R McConnel, and M. Tsao. A case study of C.mmp, Cm*, and C.vmp: Part I - experiences with fault tolerance in multiprocessor systems. *Proceedings of the IEEE*, 66(10):1178–1199, October 1978.

[SM78] J.E. Smith and G. Metze. Strongly fault secure logic networks. *IEEE Transactions on Computers*, 27(6):491–499, June 1978.

[Smi85] G.L. Smith. Model for delay faults based upon paths. In *Proceedings of IEEE International Test Conference*, pages 342–349, 1985.

[SS90] T. Slater and M. Serra. Table of linear hybrid 90/150 cellular automata. Technical Report DCS-105-IR, Department of Computer Science, University of Victoria, Victoria, BC, Canada, 1990.

[SSMM90] M. Serra, T. Slater, J.C. Muzio, and D.M. Miller. The analysis of one-dimensional linear cellular automata and their aliasing properties. *IEEE Transactions on Computer-Aided Design*, 9(7):767–778, July 1990.

[Sto73] H.S. Stone. *Discrete Mathematics Structures and Their Applications*. Science Research Associates Inc., 1973.

[TM94] N.A. Touba and E.J. McCluskey. Transformed pseudo-random patterns for BIST. Technical Report CSC No. 94-10, Center for Reliable Computing, Computer Systems Laboratory, Stanford University, October 1994.

[TMWH92] K.S. Trivedi, J.K. Muppala, S.P. Woolet, and B.R. Haverkort. Composite performance and dependability analysis. *Performance Evaluation*, 14:197–212, February 1992.

[Tri92] S. Trimberger. Field-programmable gate arrays. *IEEE Design and Test of Computers*, pages 3–5, September 1992.

[TvdG91] G. Tromp and A.J. van de Goor. Logic synthesis of 100-percent testable logic networks. In *Proceedings of IEEE International Conference on Computer Design*, pages 428–431, 1991.

[vdG91] A. J. van de Goor. *Testing Semiconductor Memories - Theory and Practice*. John Wiley and Sons. 1991.

[WLRI87] J.A. Waicukauski. E. Lindbloom. B.K. Rosen. and V.S. Iyengar. Transition fault simulation. *IEEE Design and Test of Computers*. pages 32–38. April 1987.

[ZBM92] S. Zhang. R. Byrne. and D.M. Miller. BIST generators for sequential faults. In *Proceedings of IEEE International Conference on Computer Design*. pages 260–263. October 1992.

[ZBMM94a] S. Zhang. R. Byrne. J.C. Muzio. and D.M. Miller. The evaluation of linear finite state machine as generators for Built-In Self-Test. Technical Report DCS-227-IR. Department of Computer Science. University of Victoria. Victoria. BC. Canada. January 1994.

[ZBMM94b] S. Zhang, R. Byrne, J.C. Muzio. and D.M. Miller. Why cellular automata are better than LFSRs as Built-In Self-Test generators for sequential-type faults. In *Proceedings of IEEE International Symposium on Circuits and Systems*. pages 69–72. May 1994.

[ZBMM95] S. Zhang. R. Byrne. J.C. Muzio. and D.M. Miller. Quantitative analysis for linear hybrid cellular automata and LFSR as Built-In Self-Test generators for sequential faults. *Journal of Electronic Testing: Theory and Applications*. 7(3):209–221. December 1995.

[Zha93] S. Zhang. BIST generators for faults with sequential behavior. *M.Sc. Thesis. Department of Computer Science. University of Victoria. Victoria. B.C.. Canada*. May 1993.

[ZM95] S. Zhang and J.C. Muzio. Evaluating the safety of self-checking circuits. *Journal of Electronic Testing: Theory and Applications*. 6(2):243–253. April 1995.

[ZMM91] S. Zhang. D.M. Miller. and J.C. Muzio. Determination of minimal cost one-dimensional linear hybrid cellular automata. *IEE Electronics Letters*. 27(18):1625–1627. August 1991.

[ZMM95a] S. Zhang, D. M. Miller, and J. C. Muzio. Upper bounds on the size of reduced ordered binary decision diagrams for multiple-output functions. *Submitted to Discrete Applied Mathematics*, 1995.

[ZMM95b] S. Zhang, D.M. Miller, and J.C. Muzio. Quantitative measures of pseudorandom BIST generators and the improvement of delay fault coverage. In *Proceedings of the 1st IEEE International On-Line Testing Workshop*, July 1995.

[ZMM96] S. Zhang, D.M. Miller, and J.C. Muzio. Notes on complexity of the lookup-table minimization problem for FPGA technology mapping. *IEEE Transactions on Computer-Aided Design*, 15(12):1588–1590, December 1996.

Appendix A

Minimum Cost Maximum Length LHCA for $n \leq 500$

The following table gives the maximum length LHCA with the minimal cost. one of each degree up to 500. The entries in the table indicate which cells use rule 150. For example. the entry

(10) 2 7

represents a rule vector for an LHCA of degree 10

$$[0, 1, 0, 0, 0, 0, 1, 0, 0, 0].$$

where '0' denotes rule 150 and '1' denotes rule 90. That is. the LHCA has 10 cells. numbering the cells 1 to 10 from left to right. where cells 2 and 7 use rule 150 and the rest use rule 90.

LHCA of Degree from 1 *to* 100											
Degree	*LHCA*		*Degree*	*LHCA*		*Degree*	*LHCA*		*Degree*	*LHCA*	
(1)		1	(26)		1	(51)		1	(76)	2	22
(2)		1	(27)	1	20	(52)	2	29	(77)	3	44
(3)		1	(28)		3	(53)		1	(78)	1	41
(4)	1	3	(29)		1	(54)		9	(79)		9
(5)		1	(30)		1	(55)		17	(80)	1	71
(6)		1	(31)		11	(56)	4	14	(81)		1
(7)		3	(32)	1	15	(57)		9	(82)	1	69
(8)	2	3	(33)		1	(58)		17	(83)		1
(9)		1	(34)	1	19	(59)	4	15	(84)		36
(10)	2	7	(35)		1	(60)	2	38	(85)	1	46
(11)		1	(36)		6	(61)	1	10	(86)		1
(12)	3	7	(37)		9	(62)		5	(87)		13
(13)		5	(38)		7	(63)		31	(88)		5
(14)		1	(39)		1	(64)	3	5	(89)		1
(15)		3	(40)		8	(65)		1	(90)		1
(16)	1	15	(41)		1	(66)	1	19	(91)		15
(17)		5	(42)		19	(67)		15	(92)	3	71
(18)	1	17	(43)		3	(68)		8	(93)		33
(19)		3	(44)	4	26	(69)		1	(94)		42
(20)	2	3	(45)		9	(70)	1	37	(95)		1
(21)	1	10	(46)	2	10	(71)		17	(96)		6
(22)		5	(47)		13	(72)	6	55	(97)	1	82
(23)		1	(48)		15	(73)		9	(98)		8
(24)	8	12	(49)	1	10	(74)		1	(99)		13
(25)		9	(50)		11	(75)		7	(100)	1	67

LHCA of Degree from 101 *to* 200							
Degree	*LHCA*	*Degree*	*LHCA*	*Degree*	*LHCA*	*Degree*	*LHCA*
(101)	1 20	(126)	40	(151)	27	(176)	1 45
(102)	33	(127)	15	(152)	40	(177)	1 22
(103)	15	(128)	1 29	(153)	13	(178)	1 79
(104)	2 40	(129)	49	(154)	66	(179)	1
(105)	1	(130)	1 27	(155)	1	(180)	19
(106)	30	(131)	1	(156)	2 24	(181)	81
(107)	19	(132)	18	(157)	33	(182)	34
(108)	1 35	(133)	1 4	(158)	1	(183)	1
(109)	1 4	(134)	26	(159)	61	(184)	2 163
(110)	13	(135)	1	(160)	1 151	(185)	61
(111)	27	(136)	1 97	(161)	1 116	(186)	1 125
(112)	2 5	(137)	1 132	(162)	73	(187)	45
(113)	1	(138)	28	(163)	1 24	(188)	2 10
(114)	22	(139)	11	(164)	26	(189)	1
(115)	41	(140)	8	(165)	3 130	(190)	2 8
(116)	16	(141)	25	(166)	72	(191)	1
(117)	33	(142)	5	(167)	1 60	(192)	3 111
(118)	30	(143)	35	(168)	1 113	(193)	9
(119)	1	(144)	13	(169)	81	(194)	14
(120)	3 73	(145)	1 46	(170)	22	(195)	19
(121)	45	(146)	1	(171)	37	(196)	32
(122)	14	(147)	1 136	(172)	3	(197)	65
(123)	51	(148)	8	(173)	1	(198)	1 145
(124)	21	(149)	1 108	(174)	16	(199)	41
(125)	13	(150)	2 102	(175)	2 71	(200)	11

LHCA of Degree from 201 *to* 300											
Degree	*LHCA*		*Degree*	*LHCA*		*Degree*	*LHCA*		*Degree*	*LHCA*	
(201)	1	26	(226)	1	33	(251)		1	(276)		81
(202)		24	(227)		109	(252)	4	31	(277)	1	72
(203)		7	(228)		6	(253)		93	(278)		44
(204)	1	103	(229)		21	(254)		1	(279)		69
(205)		57	(230)		1	(255)		55	(280)	2	47
(206)	1	71	(231)		1	(256)	1	127	(281)		1
(207)	1	130	(232)		93	(257)		49	(282)	3	19
(208)		74	(233)		1	(258)	1	181	(283)		35
(209)		1	(234)	1	145	(259)		21	(284)		49
(210)		1	(235)		69	(260)	4	89	(285)	1	208
(211)		17	(236)		61	(261)		1	(286)	2	167
(212)	2	169	(237)		33	(262)	3	189	(287)		109
(213)	1	56	(238)		35	(263)		95	(288)	1	61
(214)		59	(239)		1	(264)	2	173	(289)		101
(215)		29	(240)	3	7	(265)		69	(290)		112
(216)	4	26	(241)	1	162	(266)		46	(291)		97
(217)		53	(242)		4	(267)		103	(292)		5
(218)		50	(243)		1	(268)	1	73	(293)		1
(219)		49	(244)	2	49	(269)	1	210	(294)	1	79
(220)		47	(245)		1	(270)		81	(295)		57
(221)		1	(246)		40	(271)		5	(296)		2
(222)	2	12	(247)	1	102	(272)	3	109	(297)	3	84
(223)		9	(248)	1	185	(273)		1	(298)		111
(224)		64	(249)		81	(274)	2	29	(299)		1
(225)		33	(250)	1	217	(275)		103	(300)	1	251

LHCA of Degree from 301 *to* 400											
Degree	*LHCA*		*Degree*	*LHCA*		*Degree*	*LHCA*		*Degree*	*LHCA*	
(301)		81	(326)		1	(351)	1	64	(376)		62
(302)	2	105	(327)		25	(352)	1	145	(377)		149
(303)		1	(328)		80	(353)		97	(378)	2	12
(304)	4	130	(329)		1	(354)		24	(379)		33
(305)		73	(330)		1	(355)		69	(380)	4	102
(306)		1	(331)		21	(356)		40	(381)	1	44
(307)		123	(332)	1	65	(357)		73	(382)		138
(308)		14	(333)	1	52	(358)		9	(383)		77
(309)		1	(334)		104	(359)		1	(384)	1	215
(310)	1	43	(335)	1	12	(360)	1	97	(385)		161
(311)		7	(336)	4	35	(361)	1	282	(386)		1
(312)	1	295	(337)	1	100	(362)	2	154	(387)	1	176
(313)		57	(338)	1	41	(363)		51	(388)	1	339
(314)	3	41	(339)	2	9	(364)		27	(389)		89
(315)		123	(340)	2	37	(365)		85	(390)	2	56
(316)	1	25	(341)		145	(366)	1	65	(391)	1	252
(317)	2	127	(342)		69	(367)		93	(392)		173
(318)		16	(343)		99	(368)		139	(393)		33
(319)		21	(344)		13	(369)	1	74	(394)		59
(320)	3	79	(345)		49	(370)		75	(395)	1	86
(321)		97	(346)		32	(371)		1	(396)	1	113
(322)	2	74	(347)		13	(372)	1	257	(397)		113
(323)		1	(348)	1	25	(373)		5	(398)		1
(324)		19	(349)		117	(374)	2	313	(399)		61
(325)		33	(350)	2	63	(375)		1	(400)		198

LHCA of Degree from 401 *to* 500							
Degree	*LHCA*	*Degree*	*LHCA*	*Degree*	*LHCA*	*Degree*	*LHCA*
(401)	185	(426)	1	(451)	11	(476)	25
(402)	154	(427)	2 91	(452)	29	(477)	1 266
(403)	179	(428)	184	(453)	1	(478)	86
(404)	68	(429)	1	(454)	18	(479)	1 140
(405)	117	(430)	3 93	(455)	31	(480)	8 239
(406)	116	(431)	1	(456)	2 209	(481)	225
(407)	95	(432)	1 49	(457)	1 220	(482)	1 285
(408)	2 239	(433)	45	(458)	91	(483)	1
(409)	4 97	(434)	86	(459)	19	(484)	3 237
(410)	3 181	(435)	2 281	(460)	32	(485)	1 272
(411)	1	(436)	17	(461)	1 90	(486)	1 181
(412)	185	(437)	5	(462)	138	(487)	159
(413)	1	(438)	187	(463)	33	(488)	188
(414)	4 137	(439)	171	(464)	82	(489)	109
(415)	1 276	(440)	4 135	(465)	1 202	(490)	2 287
(416)	1 219	(441)	1	(466)	90	(491)	1
(417)	129	(442)	138	(467)	95	(492)	15
(418)	36	(443)	1	(468)	2 378	(493)	1 184
(419)	1	(444)	184	(469)	113	(494)	2 3
(420)	1 323	(445)	197	(470)	1	(495)	1
(421)	1 108	(446)	1 443	(471)	2 185	(496)	3 69
(422)	1 21	(447)	223	(472)	2 212	(497)	1 200
(423)	2 221	(448)	1 153	(473)	1	(498)	63
(424)	2 80	(449)	209	(474)	61	(499)	1 174
(425)	37	(450)	2 137	(475)	45	(500)	2 78