

**EVALUATION OF FIBONACCI TEST PATTERN GENERATOR FOR COST
EFFECTIVE IC TESTING**

by

MD. TANVEER AHMED

MASTER OF ENGINEERING IN INFORMATION AND COMMUNICATION
TECHNOLOGY

Institute of Information and Communication Technology
BANGLADESH UNIVERSITY OF ENGINEERING AND TECHNOLOGY

January, 2013

The project titled “EVALUATION OF FIBONACCI TEST PATTERN GENERATOR FOR COST EFFECTIVE IC TESTING” submitted by Md. Tanveer Ahmed, Roll No. M04083119P, Session April-2008 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of Master of Engineering (ICT) held on 19th January, 2013.

BOARD OF EXAMINERS

-
- | | | |
|----|---|----------|
| 1. | Dr. Md. Liakot Ali
Professor
Institute of Information and Communication Technology
BUET, Dhaka-1000, Bangladesh. | Chairman |
|----|---|----------|
-
-
- | | | |
|----|---|--------|
| 2. | Dr. Md. Saiful Islam
Professor
Institute of Information and Communication Technology
BUET, Dhaka-1000, Bangladesh. | Member |
|----|---|--------|
-
-
- | | | |
|----|---|--------|
| 3. | Dr. Mohammad Shah Alam
Assistant Professor
Institute of Information and Communication Technology
BUET, Dhaka-1000, Bangladesh. | Member |
|----|---|--------|

CANDIDATE'S DECLARATION

It is hereby declared that this report or any part of it has not been submitted elsewhere for the award of any degree or diploma.

Md. Tanveer Ahmed

**Dedicated to
My Parents**

CONTENTS

Title	Page No
BOARD OF EXAMINERS	II
CANDIDATE’S DECLARATION	III
CONTENTS	V
LIST OF TABLES	X
LIST OF FIGURES	XI
LIST OF ABBREVIATION	XIV
ACKNOWLEDGEMENT	XVI
ABSTRACT	XVII
CHAPTER 1: INTRODUCTION	1
1.1 Introduction	1
1.2 Motivation	2
1.3 Objectives with Specific Aims and Possible Outcome	3
1.4 Outline of the Project	3
 CHAPTER 2: FUNDAMENTALS OF IC TESTING	 5
2.1 Introduction	5
2.2 Preliminaries	5
2.3 Different Approaches of IC Testing	7
2.3.1 Exhaustive Approach	7
2.3.2 Pseudo-Exhaustive Approach	8
2.3.3 Deterministic Approach	8
2.3.4 Pseudo-random Approach	9
2.3.5 Weighted Random Approach	10
2.3.6 Mixed-mode Approach	11

2.4	Faults in IC	12
	2.4.1 Fault Modeling	13
	2.4.2 Fault Coverage	15
	2.4.3 Fault Simulation	15
2.5	Linear Feedback Shift Register (LFSR)	17
	2.5.1 LFSR Terminology	19
2.6	LFSR Design	20
	2.6.1 Galois Linear Feedback Shift Register (GLFSR)	21
	2.6.2 Fibonacci Linear Feedback Shift Register (FLFSR)	22
2.7	Primitive Polynomial	23
2.8	Coefficient of Variation (CV)	25
2.9	Summary	26
CHAPTER 3:	EVALUATION TECHNIQUE FOR TEST PATTERN	
	GENERATOR	28
3.1	Introduction	28
3.2	Basic Test Arrangement	28
3.3	Benchmark Circuits	29
3.4	Design of 64-bit FLFSR	31
	3.4.1 64-bit Fibonacci Linear Feedback Shift Register	31
	3.4.2 Feedback	31
	3.4.3 Conventions for Feedback Tap Specification	32
	3.4.4 Taps of Linear Feedback Shift Register	33
3.5	Illustration of FLFSR Working	36

3.6	Test Pattern Generation Using FLFSR	37
3.7	Techniques for Determining Best Seed	41
3.8	Software Requirements for Proposed Work	42
	3.8.1 Fault Simulator (FSIM)	42
	3.8.2 MATLAB	44
3.9	Summary	44
CHAPTER 4:	RESULTS AND DISCUSSIONS	45
4.1	Introduction	45
4.2	Fault Simulation	45
4.3	Fault simulation results of the ISCAS Benchmark Circuits	46
	4.3.1 Best Seed Determination	61
	4.3.2 Comparison	63
4.4	Summary	65
CHAPTER 5:	CONCLUSION AND FUTURE WORK	66
5.1	Conclusion	66
5.2	Future Work	66
	REFERENCES	67
	APPENDIX	70
A1	Computer Program for generating random numbers	70
A2	Command for Fault Simulation using FSIM	72
A3	Random Test Vectors for ISCAS85 Benchmark Circuit c432.bench	74
A4	Fault simulation result of circuit c432.bench (For feedback polynomial $1+x^{60}+x^{61}+x^{63}+x^{64}$)	79

A5	Fault simulation result of circuit c432.bench (For feedback polynomial $1+x+x^3+x^4+x^{64}$)	80
A6	Fault simulation result of circuit c499.bench (For feedback polynomial $1+x^{60}+x^{61}+x^{63}+x^{64}$)	81
A7	Fault simulation result of circuit c499.bench (For feedback polynomial $1+x+x^3+x^4+x^{64}$)	82
A8	Fault simulation result of circuit c880.bench (For feedback polynomial $1+x^{60}+x^{61}+x^{63}+x^{64}$)	83
A9	Fault simulation result of circuit c880.bench (For feedback polynomial $1+x+x^3+x^4+x^{64}$)	84
A10	Fault simulation result of circuit c1355.bench (For feedback polynomial $1+x^{60}+x^{61}+x^{63}+x^{64}$)	85
A11	Fault simulation result of circuit c1355.bench (For feedback polynomial $1+x+x^3+x^4+x^{64}$)	86
A12	Fault simulation result of circuit c1908.bench (For feedback polynomial $1+x^{60}+x^{61}+x^{63}+x^{64}$)	87
A13	Fault simulation result of circuit c1908.bench (For feedback polynomial $1+x+x^3+x^4+x^{64}$)	88
A14	Fault simulation result of circuit c2670.bench (For feedback polynomial $1+x^{60}+x^{61}+x^{63}+x^{64}$)	89
A15	Fault simulation result of circuit c2670.bench (For feedback polynomial $1+x+x^3+x^4+x^{64}$)	90
A16	Fault simulation result of circuit c3540.bench (For feedback polynomial $1+x^{60}+x^{61}+x^{63}+x^{64}$)	91
A17	Fault simulation result of circuit c3540.bench (For feedback polynomial $1+x+x^3+x^4+x^{64}$)	92
A18	Fault simulation result of circuit c5315.bench (For feedback polynomial $1+x^{60}+x^{61}+x^{63}+x^{64}$)	93
A19	Fault simulation result of circuit c5315.bench (For feedback polynomial $1+x+x^3+x^4+x^{64}$)	94

A20	Fault simulation result of circuit c6288.bench (For feedback polynomial $1+x^{60}+x^{61}+x^{63}+x^{64}$)	95
A21	Fault simulation result of circuit c6288.bench (For feedback polynomial $1+x+x^3+x^4+x^{64}$)	96
A22	Screenshot of Fault Simulation Result of Circuit c432.bench (For feedback polynomial $1+x^{60}+x^{61}+x^{63}+x^{64}$)	97
A23	Screenshot of Fault Simulation Result of Circuit c499.bench (For feedback polynomial $1+x^{60}+x^{61}+x^{63}+x^{64}$)	98
A24	Screenshot of Fault Simulation Result of Circuit c1908.bench (For feedback polynomial $1+x^{60}+x^{61}+x^{63}+x^{64}$)	99
A25	Matlab Program for Measuring CV of PRV Sequences	100
A26	Screenshot of the Output of Matlab Program for Measuring CV	101
OUTCOME OF THIS PROJECT		102

LIST OF TABLES

Title		Page No
Table 2.1	States of the LFSR with feedback polynomial	19
Table 2.2	Primitive polynomials (mod 2) of orders 1 through 5	25
Table 3.1	ISCAS Benchmark Circuits	30
Table 3.2	Taps for maximum-length LFSR counters	35
Table 4.1	Fault coverage for different PRVs for benchmark circuit c432.bench	49
Table 4.2	FC Comparison for two Different Primitive Polynomials	50
Table 4.3	CV of PRV sequences for different seeds for different benchmark circuits	62
Table 4.4	Summary of fault simulation results of the ISCAS85 benchmark circuits with using proposed technique	64
Table 4.5	Comparison of fault simulation results of the ISCAS85 benchmark circuits with that of other researchers	64

LIST OF FIGURES

Title	Page No
Figure 2.1	Fault coverage versus random test vectors (Stanley 1998) 9
Figure 2.2	An eight-input AND gate 10
Figure 2.3	Generalized scheme of the mixed-mode technique 11
Figure 2.4	General structure of an n-bit LFSR 17
Figure 2.5	A four stage LFSR 18
Figure 2.6	Structure of Galois LFSR 21
Figure 2.7	3-bit GLFSR with feedback ploynomial $1+x+x^3$ 21
Figure 2.8	Structure of Fibonacci LFSR 22
Figure 2.9	3-bit FLFSR with feedback ploynomial $1+x^2+x^3$ 22
Figure 2.10	8-bit FLFSR with feedback ploynomial $1+x^4+x^5+x^6+x^8$ 22
Figure 3.1	General IC testing process 29
Figure 3.2	Feedback Logic using XOR gates 32
Figure 3.3	64-bit FLFSR with feedback logic 32
Figure 3.4	Galois implementation of LFSR 33
Figure 3.5.	(a) Block Diagram, 36
	(b) truth table, 36
	(c) state diagram of 4-bit FLFSR with characteristics 37
	ploynomial $1+X^3+X^4$
Figure 3.6	Flow chart of test pattern generation 39
Figure 3.7	Test vector file generation using FLFSR 40
Figure 3.8	Flow Chart for measuring CV 42
Figure 4.1	Fault detection profile of PRV for the benchmark circuit c432.bench 47
Figure 4.2	Fault simulation result of circuit c432.bench 51
	(For feedback ploynomial $1+x^{60}+x^{61}+x^{63}+x^{64}$)

Figure 4.3	Fault simulation result of circuit c432.bench (For feedback polynomial $1+x+x^3+x^4+x^{64}$)	52
Figure 4.4	Fault simulation result of circuit c499.bench (For feedback polynomial $1+x^{60}+x^{61}+x^{63}+x^{64}$)	52
Figure 4.5	Fault simulation result of circuit c499.bench (For feedback polynomial $1+x+x^3+x^4+x^{64}$)	53
Figure 4.6	Fault simulation result of circuit c880.bench (For feedback polynomial $1+x^{60}+x^{61}+x^{63}+x^{64}$)	53
Figure 4.7	Fault simulation result of circuit c880.bench (For feedback polynomial $1+x+x^3+x^4+x^{64}$)	54
Figure 4.8	Fault simulation result of circuit c1355.bench (For feedback polynomial $1+x^{60}+x^{61}+x^{63}+x^{64}$)	54
Figure 4.9	Fault simulation result of circuit c1355.bench (For feedback polynomial $1+x+x^3+x^4+x^{64}$)	55
Figure 4.10	Fault simulation result of circuit c1908.bench (For feedback polynomial $1+x^{60}+x^{61}+x^{63}+x^{64}$)	55
Figure 4.11	Fault simulation result of circuit c1908.bench (For feedback polynomial $1+x+x^3+x^4+x^{64}$)	56
Figure 4.12	Fault simulation result of circuit c2670.bench (For feedback polynomial $1+x^{60}+x^{61}+x^{63}+x^{64}$)	56
Figure 4.13	Fault simulation result of circuit c2670.bench (For feedback polynomial $1+x+x^3+x^4+x^{64}$)	57
Figure 4.14	Fault simulation result of circuit c3540.bench (For feedback polynomial $1+x^{60}+x^{61}+x^{63}+x^{64}$)	57
Figure 4.15	Fault simulation result of circuit c3540.bench (For feedback polynomial $1+x+x^3+x^4+x^{64}$)	58
Figure 4.16	Fault simulation result of circuit c5315.bench (For feedback polynomial $1+x^{60}+x^{61}+x^{63}+x^{64}$)	58
Figure 4.17	Fault simulation result of circuit c5315.bench (For feedback polynomial $1+x+x^3+x^4+x^{64}$)	59

Figure 4.18	Fault simulation result of circuit c6288.bench (For feedback polynomial $1+x^{60}+x^{61}+x^{63}+x^{64}$)	59
Figure 4.19	Fault simulation result of circuit c6288.bench (For feedback polynomial $1+x+x^3+x^4+x^{64}$)	60
Figure 4.20	CV for different seed positions	62
Figure 4.21	Image representation of PRV sequences	63

LIST OF ABBREVIATION

LFSR	Linear Feedback Shift Register
ASIC	Application Specific Integrated Circuit
ATE	Automatic Test Equipment
GLFSR	Galois Linear Feedback Shift Register
FLFSR	Fibonacci Linear Feedback Shift Register
CUT	Circuit Under Test
BIST	Built-In Self-Test
SOC	System-on-a-chip
PRV	Pseudo Random Vector
TV	Test Vector
NTV	Number of Test Vector
FC	Fault Coverage
XOR	Exclusive-OR
XNOR	Exclusive-NOR
ISCAS	International Symposium on Circuits and Systems
IC	Integrated Circuit
CAR	Cellular Automata Register
DFT	Design For Testability
FSIM	Fault Simulator
ETD	Easy-To-Detect
HTD	Hard-To-Detect
VLSI	Very Large Scale Integration

NG	Number of Gates
NL	Number of Lines
NPI	Number of Primary Inputs
NPO	Number of Primary Outputs
NS	Number of Stems
CV	Coefficient of Variation
IDDQ	Integrated Circuit Quiescent Current

ACKNOWLEDGEMENT

(All praise be to Allah, The Beneficent, The Merciful)

First of all I would like to thank the almighty Allah for giving me the opportunity to conduct this project. I would like to express my sincere thanks to my research project supervisor, Dr. Md. Liakot Ali, for giving me the opportunity to conduct this project. Without his ever helping personalities, this project would not have got the success.

I would like to convey my thanks to Professor Dr. Md. Saiful Islam, Director, IICT, BUET and Assistant Professor Dr. Mohammad Shah Alam, IICT, BUET. Their motivation and inspiration gave me the courage to do this work.

I gratefully acknowledge the restless support and advice of my fellow classmates and friends during the design and implementation phase of this project. My special thanks to all other teachers, students and staffs of IICT, BUET.

I would like to thank all of my friends and family members for their continuous support and inspiration throughout the whole period of this undertaking.

ABSTRACT

With the increase of the complexities of VLSI circuit, testing problem has become more acute. Testing at low cost with reliable performance is now a burning issue in the semiconductor world. Test pattern generator is very important in VLSI Testing. Researchers have proposed different testing approaches where test pattern generation plays an important role on performance of the testing. Finding proper seed for a test pattern generator, finding optimum switching point from pseudo-random test technique to deterministic test etc. are challenges in VLSI testing. Fault simulation experiments have been conducted on a number of benchmark circuits to find the best seed and optimum switching points. Recently Fibonacci pseudo-random test pattern generator has been proved efficient in many cryptographic applications. Then we have evaluated the effectiveness of a 64-bit Fibonacci test pattern generator in VLSI circuit testing. The project focuses on design and simulation of a 64-bit Fibonacci test pattern generator capable of generating sufficient long test pattern. By changing the seed and feedback connection, a set of test vectors are generated for different benchmark circuits. Then we have conducted fault simulation experiments on ISCAS (International Symposium on Circuits and Systems) benchmark circuits for its evaluation in cost effective IC Testing. The result has been compared with that of other researchers. It is found better as compared to all other results.

CHAPTER 1

INTRODUCTION

1.1 Overview

In this modern era, electronic equipment and products have become a part and parcel of our daily life. Zero failure, reliability and longevity are now major business issues as well as customer's expectation for the electronic goods. Accuracy and high reliability are essential and even life critical for many applications, for example, in medical and aerospace. The key components of an electronic product are integrated circuits (ICs). In IC manufacturing, various physical defects may occur during numerous physical, chemical and thermal processes [1]. It is very unwise to sell components to customers without being absolutely sure that the devices are functioning as per specification. During the early stage of semiconductor technology, testing of IC was simple and the designer did not need to be much concerned about that. With the dramatic improvement of semiconductor technology, the design complexities and packing densities of IC have exceedingly increased. Now millions of transistors are being integrated on a single chip. System-on-a-chip (SOC) is a vision of this era. With the increase of the complexities of VLSI circuit, testing problem has become more complex. Testing at low cost with reliable performance is now a burning issue in the semiconductor world. If a fault can be detected at an earlier stage, it is possible to avoid a larger cost of fault detection. Efficient testing method plays a vital role in the economic success of VLSI circuits. Researchers have proposed different testing approaches. They are exhaustive test technique, pseudo random test technique, weighted random test technique, mixed mode test technique etc. Usually in all the testing approaches a number of test patterns are applied to the circuit under test (CUT) and faults are detected. In a literature, it has been shown that number of faults detected depends on the randomness of the test pattern. So test

pattern generation technique is very important in VLSI circuit testing. Usually linear feedback shift register (LFSR) and cellular automata register (CAR) are widely used in test pattern generation for testing ICs. Due to the limitations of integration and fabrication technology previous researchers used 32-bit LFSR or CAR in designing IC tester or test processor chip. However now integration technology has tremendously improved. In this project a 64-bit Fibonacci test pattern generator which is a modified version of LFSR has been evaluated in VLSI testing. Fibonacci test pattern generator has already been used in many cryptographic applications and proved very much efficient. So, there are scopes of research to evaluate its effectiveness in VLSI testing.

1.2 Motivation

Modern IC production facilities use computer controlled Automatic Test Equipment (ATE) for testing ICs. With the increase of complexities of ICs, ATE suffers from the following drawbacks, (i) high equipment cost ,(ii) slow test speed ,(iii) huge memory requirements ,(iv) yield less. To ease the burden of IC testing using ATE, different test techniques have been proposed in the literatures for the purpose of reducing the test cost using 32-bit test pattern generator [2-3]. Based on the proposed algorithms different researchers have shown their fault simulation results to prove the effectiveness of their proposed techniques [2-8]. But the number of test patterns to achieve full fault coverage is still high enough. In all the test techniques test pattern generation plays a vital role because if the generated pattern has better randomness then performance of the testing increases. Recently Fibonacci pseudo-random test pattern generator has been proved efficient in many cryptographic applications. As a result in this project work we proposed a new 64-bit Fibonacci test pattern generator which is a modified version of LFSR .So there are scopes

of research to evaluate the Fibonacci pseudo-random test pattern generator to be used in VLSI testing.

1.3 Objectives with Specific Aims and Possible Outcome

The proposed project has the following objectives:

- To design a Fibonacci test pattern generator capable of generating sufficient long test pattern without repeating the sequence.
- To conduct fault simulation experiments on benchmark circuits.
- To evaluate the test pattern and find the fault coverage for different seeds and different tap positions of the proposed test pattern generator.
- To determine the best seed and the optimum switching point for a mixed mode test.
- To compare the test results with that of other researchers.

1.4 Outline of the Project

The project is arranged in five chapters. Chapter 1 describes complexities and difficulties of testing ICs of present days. Scope and motivation of the research work is presented in this chapter.

Chapter 2 of this project describes different testing techniques of digital IC. It also covers the theory of fault modeling and fault simulation, the details of LFSR, types of LFSR, test pattern generation and primitive polynomial.

Chapter 3 of this project describes about the details design of the proposed FLFSR.

Chapter 4 of this project describes the simulation result of the ISCAS benchmark circuits using Fibonacci test pattern generator by FSIM and comparison of fault simulation results with that of

other researchers. Extensive simulations are performed to study the effect of reseeding and programmability of feedback polynomial in improving fault coverage.

Chapter 5 presents the conclusion and future work of the proposed system.

CHAPTER 2

FUNDAMENTALS OF IC TESTING

2.1 Introduction

This chapter includes the fundamental topics related to VLSI circuit testing. It has been mentioned in the previous chapter that the aim of the research work presented in this project is to design a Fibonacci test pattern generator and evaluates its effectiveness in VLSI circuit testing. In order to give a better understanding for the reader of this project, different topics related to digital IC testing such as fault modeling and fault simulation, different types of LFSR and different types of testing techniques and their relative advantages and disadvantages have been discussed in this chapter.

2.2 Preliminaries

Test pattern (test vector): To carry out the testing of a digital circuit, a set of signals are applied to its input and then compare the corresponding output responses with that of a good circuit. The input signals are called input test vectors and output signals are called response vectors. In a sequence of test vectors, if the combination of binary 0 and 1 are at random and are not predictable then it is called random test vectors. True random test vectors are not possible to generate. Generally, after a certain period, the sequence repeats and so they are known as pseudo-random test vectors.

Fault coverage: The percentage of detected faults over all possible detectable faults is known as fault coverage. Typically, fault coverage refers to the percentage of single faults detected by the test. As a rule, test engineers attempt to provide as close as possible to 100% fault coverage. Due

to the difficulty in developing such tests, however, in practice, a fault coverage that is too high is difficult to achieve.

Fault simulation: An empirical method used to determine how faults affect the operation of a circuit and/or also how much testing is required to obtain desired fault coverage. Fault simulation plays a significant role in the testing of digital circuits. It determines which faults in the circuit are detected by a given set of test patterns.

LFSR: A shift register formed by flip-flops and XOR gates, chained together, with a synchronous clock, used either as input pattern generator or as signature analyzer. An LFSR with a well-chosen feedback function can produce a sequence of bits which appears random and which has a very long cycle.

Maximal Length Sequences (L): A maximal length sequence for a shift register of length N is referred to as an m -sequence, and is defined as: $L = 2^N - 1$. An eight-stage LFSR, for example, will have a set of m -sequences of length 255.

Pseudo-random pattern generator: It generates a binary sequence of patterns where the patterns appear to be random in the local sense, but they are deterministically repeatable.

Random testing: The process of testing using a set of pseudo-randomly generated patterns.

MISR: Multiple-input LFSR.

Aliasing: It occurs if the faulty output produces the same signature as a fault-free output.

Built-in self-test (BIST): Built-in self-test is the capability of a digital circuit to carry out self test using the built in hardware facilities. It reduces the costs of external test pattern generation and fault simulation, the testing time and simplifies the external test equipment.

Off-line testing: A testing process carried out while the tested circuit is not in use.

On-line testing: Concurrent testing to detect errors while circuit is in operation.

Signature analysis/data compaction: A test where the output responses of a device over time are compacted into a characteristic value called a *signature*, which is then compared to a known good one.

Stuck-at fault: A fault model represented by a signal stuck at a fixed logic value (0 or 1).

Redundant Fault: If there is no test vector that can detect a fault in a circuit then the fault is called redundant [9].

2.3 Different Approaches of IC Testing

With the evolution of test technology, various approaches have been developed for IC testing. The choice of an approach depends on the factors such as fault coverage, test length, test application time and simplicity of hardware. Major approaches for IC testing are as follows:

- Exhaustive approach
- Pseudo-exhaustive approach
- Deterministic approach
- Pseudo-random approach
- Weighted random approach
- Mixed-mode approach

2.3.1 Exhaustive approach

According to exhaustive approach all the possible combination of input test vectors is generated and applied to the circuit-under-test (CUT) [1]. For a circuit with n number of inputs, the possible combination of input vectors will be 2^n . Binary counter, Gray counter, linear feedback shift register (LFSR) are generally used as exhaustive test pattern generator. This approach offers

100% fault coverage with a low computational cost. However with inputs roughly greater than twenty test lengths become very long. If the number of inputs is high, then test length becomes very long and takes much longer time for test generation. For example, the exhaustive testing of the 8080 microprocessor would take over 10^{20} years, at one million tests a second.

2.3.2 Pseudo-exhaustive approach

In pseudo-exhaustive approach, circuits are logically partitioned into smaller parts and then each part is tested exhaustively by much fewer number of test vectors than that of exhaustive approach. To reduce hardware overhead and testing time, different algorithms have been proposed for circuit-partitioning. Although pseudo-exhaustive method achieves the benefits of exhaustive testing by using far fewer test patterns but the disadvantage of this approach is the large test sets with the increase of complexities in circuits. Very often physical segmentation of the circuit is necessary in this approach. Moreover, hardware implementation of pseudo-exhaustive pattern generator is difficult and most of the design does not lead to minimal test set.

2.3.3 Deterministic approach

Deterministic approach allows CUT to be examined at first of the test and test vectors are generated after that using any suitable algorithm for deterministic test pattern generation. For deterministic test pattern generation different algorithms have been proposed such as Exclusive-OR, D-algorithm, PODEM (path-oriented decision making) and FAN (Fan oriented test pattern) [10]. This approach enables error signals to be generated due to presence of faults and propagate

them to some observable output form. This approach guarantees full fault coverage but large test data volumes and computational complexities are the drawbacks of this approach.

2.3.4 Pseudo-random approach

Pseudo-random approach is now an established technique for low cost IC testing [11]. In this approach, a set of test vectors is generated randomly from 2^n possible input patterns where n is the number of inputs. Linear feedback shift registers (LFSRs) are commonly used for test pattern generation for its simple structure and can also be used as output response analyzer and thereby serves dual purposes. The main advantage of this approach is that random pattern generation circuitry is simple and a large number of tests can be generated using smaller data storage. The disadvantage of this approach is that the length of the test set that detects a set of faults is much larger (usually 10 times or more) than deterministically generated test set for the same faults. Figure 2.1 shows relationship between fault coverage and pseudo-random test vectors applied to a typical complex circuit.

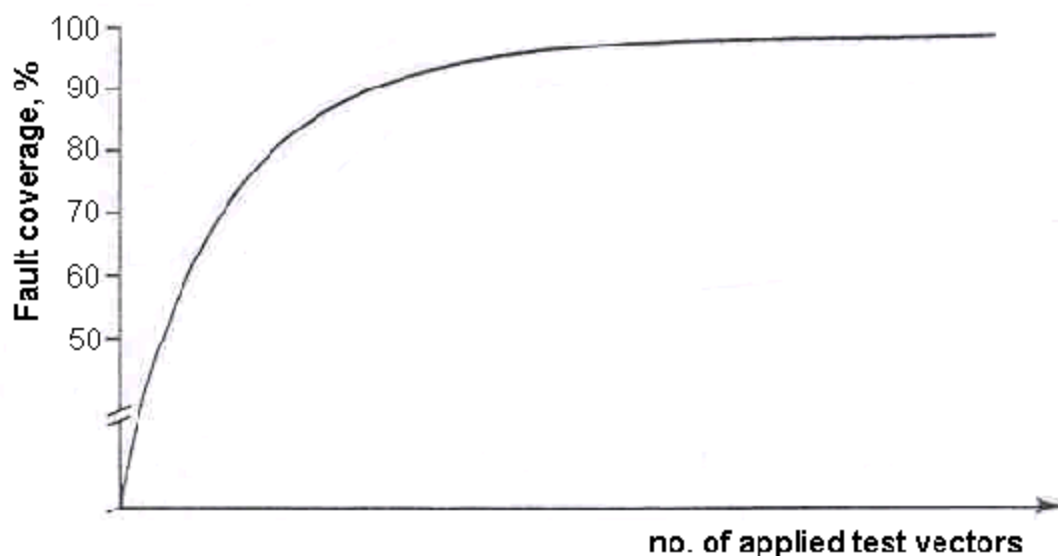


Figure 2.1: Fault coverage versus random test vectors

Figure 2.1 shows that in pseudo-random approach, almost 90% fault coverage can be achieved using fewer number of test vectors. A large number of test vectors are needed to detect the remaining faults. Another implication of this approach is that there are faults in some circuits known as random pattern resistant faults where acceptable fault coverage cannot be achieved even after applying a large number of test patterns. To illustrate this, an eight-input AND gate is shown in Figure 2.2.

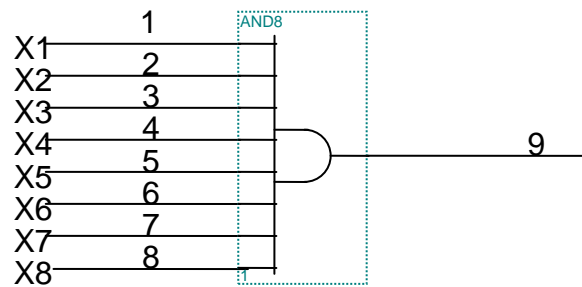


Figure 2.2: An eight-input AND gate

A stuck-at-1 fault at line 1 in Figure 2.2 can be detected by only one test vector which is $X1.....X8=\{01111111\}$. Hence the probability that this fault may be detected by a random pattern is $1/256$. The probability decreases with the increase of number of inputs.

2.3.5 Weighted random approach

Weighted random approach has been proposed to overcome the drawbacks of low fault coverage due to hard-to-detect (HTD) and random pattern resistant faults in IC testing. It has been shown that the biased or the weighted pseudo random vector (PRV) can test PRV resistant faults more efficiently using a lower number of test vectors than the unbiased PRV. The disadvantage of this approach is preprocessing is necessary to calculate the signal probabilities for every input and to generate necessary weight set. Complex and additional hardware circuitry is necessary to design

weighted random pattern generator. Moreover, for complex circuits, multiple sets of weight are necessary to achieve acceptable fault coverage. This leads to a large volume of test data to be stored and manipulated.

2.3.6 Mixed-mode approach

Mixed-mode approach was first proposed by Koenemann [12]. It is a hybrid test technique where deterministic test technique is followed by pseudo-random test technique. This approach exploits advantages of both the pseudo-random test technique and the deterministic test technique. A generalized scheme of mixed-mode technique is presented in Figure 2.3.

Generally most of the faults in a typical circuit are easy-to-detect (ETD), which can be easily detected using the first few PRVs and the remaining faults are HTD, which need long PRV sequences to detect [1]. In the mixed-mode approach, PRV is generated using LFSR and is applied to a CUT to detect all the ETD faults and then deterministic test sets are generated using the same LFSR to target the remaining HTD faults using compacted test data named as seed. Thereby all the faults of a circuit are detected in this approach.

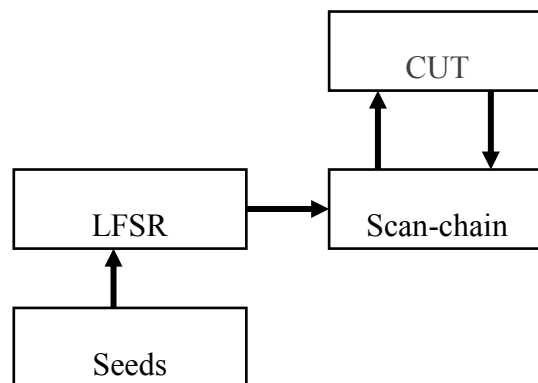


Figure 2.3: Generalized scheme of the mixed-mode technique

A number of mixed-mode approaches have been proposed and it has been claimed that this approach ensures complete fault coverage while offers reduced storage requirements, shorter test application time, and smaller area overhead compared to weighted random approach [12-16].

2.4 Faults in IC

Faults in an IC can be defined in terms of its operation. The IC is faulty if it does not perform the job it is supposed to do. Error in the operation of an IC is the manifestation of a fault while fault is the manifestation of physical defect [17]. Physical defects in the IC occur during the numerous physical, chemical and thermal processes in IC manufacturing. Some common defects are particles (small bits of materials that bridge two lines), incorrect spacing (wide or narrow variations in line spacing that may short a circuit), incorrect implant value (due to machine error or blockages), misalignment (misplacing one layer with respect to the previous layer), holes (exposed area that is unexpectedly etched), weak oxides and contamination (unwanted foreign material).

Faults can be categorized in different ways. If the presence of faults changes the logic value of a signal line in a circuit from zero to one or vice versa, they are referred to as logical faults while if the fault causes some parameters of the circuit to change, such as current drawn by the circuit, it is known as parametric fault [9]. It also can be classified as (a) transient (b) intermittent and (c) permanent depending on the basis of duration of faults in a circuit. The presence of transient faults is only for a short duration. The transient faults are generally caused by α -particle radiation or power supply fluctuation. Intermittent faults appear regularly but not present continuously. They are generally due to loose connections, bad designs or environmental effects like

temperature and humidity variations. Permanent faults remain present continuously. They are predominantly caused by shorts and opens in the IC.

2.4.1 Fault Modeling

An IC may have different types of physical defects and the number of defects increases with the increase of its complexities. If each of the physical defects of an IC needs to be considered for its test, it would soon become unmanageable. In order to successfully tackle this problem, the physical defects in a chip are represented at a higher level with the help of fault model. Any fault from a fault model may represent many physical defects. Thus the use of fault models speeds up the testing process of an IC. Most popular fault models are (a) stuck-at (b) stuck-open (c) stuck-on (d) bridging (e) parametric and (f) delay fault. Brief discussion of different types of fault models is given below:

Stuck-at fault model: Most commonly used fault model is the stuck-at fault model and it has been proven reliable in practical applications [18]. The fault in this model makes a signal line in a circuit permanently at logic 0 or vice versa. If the line is permanently at logic 0, it is said to be stuck-at 0 (s-a-0) while if the line is permanently at logic 1, it is said to be stuck-at 1 (s-a-1).

Stuck-open fault model: The fault which makes a transistor in a circuit non-conducting is known as stuck-open fault [9]. In order to detect a stuck-open fault, exact sequence of vectors is required to be fed to the circuit. It usually requires a sequence consisting of two vectors to detect stuck-open fault. The first vector is called the initialization vector and the second vector is called the test vector. The sequence of these two vectors is referred to as two-pattern test.

Stuck-on fault model: If a fault causes a transistor in a circuit to conduct continuously, the transistor is said to be stuck-on. Logic testing does not guarantee to detect this type of fault. Integrated circuit quiescent current (IDDQ) testing (monitoring the quiescent current of a device) is a very effective technique for detecting stuck-on faults.

Bridging fault model: Bridging fault indicates short between two or more signal lines in a circuit. This may occur due to defective masking or etching, aluminum migration or breakdown of insulators [9]. Bridging fault results in complex situation in a circuit. IDDQ testing is usually used to detect these faults [17].

Parametric fault model: There may occur some faults in an IC, which do not affect the functional behavior, but affect the performance and reliability of the circuit. They are known as parametric faults, which include the substrate leakage current, gate-oxide leakage current, variation in threshold voltage and capacitive coupling or cross talk. Parametric faults may create functional error at any future instant. Usually, these faults are detected by accelerated stress testing based on voltage, current, temperature, shock and vibration. IDDQ testing is also an example of parametric test.

Delay fault model: This type of fault occurs due to the propagation delay of any signal in a circuit [9]. The voltage on a faulty line could either be slow-to-rise or slow-to-fall. This fault usually occurs due to stuck-on fault or structural impurities. This fault-model is classified two types: (a) gate delay model and (b) path delay model. Gate delay model is based on the delay at

the inputs or output of a logic gate while path delay model is based on the cumulative propagation delays along a circuit path.

2.4.2 Fault Coverage

Faults in an IC are detected by applying a set of binary input sequences (known as test patterns) to the inputs of the IC. The test patterns make the fault effects observable at the outputs of the IC. Fault detection ability of a set of test patterns are measured by fault coverage which is defined as the percentage of ratio of the number of faults detected to the number of possible faults [18]. Therefore,

$$FC \text{ (fault coverage)} = \frac{\text{Number of detected faults}}{\text{Number of possible faults}} \times 100\%$$

Some faults in the circuits are undetectable. Therefore, the actual fault coverage:

$$AFC = \frac{\text{Number of detected faults}}{\text{Number of possible faults} - \text{number of undetectable faults}} \times 100\%$$

2.4.3 Fault Simulation

It consists of few processes such as (a) simulation of a circuit in the presence of faults (b) comparison of the simulation results with those of the fault free simulation of the same circuit and (c) determination of faults detected. Fault simulation technique is used for the followings:

- To evaluate (grade) a test set. The grade of a test set is expressed in terms of fault coverage. Fault coverage of a test set for a circuit can be found out using fault simulation technique.
- To generate a test set that can produce satisfactory fault coverage.

- To analyze the operation of a circuit in the presence of faults. This is especially important in high-reliability systems, since some faults may drastically affect the operation of the circuit.

The most common types of fault simulation techniques are parallel, deductive and concurrent fault simulations. In parallel fault simulation, effort is made to reduce computation time by simulating more than one fault in one pass for a given set of input vectors. In this fault simulation approach, the circuit being simulated must be expressed in Boolean terms due to which memory and large sequential circuits are impractical or impossible to handle.

Deductive fault simulation relies upon the fault list data (input/output relationship of logic gates under fault-free and chosen faulty condition). In this technique, fault lists are serially propagated through the circuit to the primary output and all the detectable faults are counted after each pass for each test vector. This method is not suitable for multiple logic values due to increasing complexities.

At present, concurrent fault simulation is the preferred method of fault simulation. It consists of simulating the fault-free circuit and concurrently simulating the faulty circuit only if the faulty circuit's activity actually differs from that of the fault-free circuit [18]. This method is capable of handling multiple logic values, which makes it suitable for increasingly complex circuits. However, this approach requires more memory space than that of deductive fault simulation approach.

2.5 Linear Feedback Shift Register (LFSR)

A linear feedback shift register (LFSR) is a shift register whose input bit is a linear function of its previous state. It is widely used for test pattern generation as it is simple and most efficient pseudo-random test pattern generator [19]. The initial value of the LFSR is called the seed. However, a LFSR with a well-chosen feedback function can produce a sequence of bits which appears random and which has a very long cycle. Figure 2.4 shows the basic structure of a standard LFSR. It consists of a set of storage elements (D-Flip-Flops) and modulo-2 adder (X-OR gate). The connection is in such a way that the state of each element is shifted to the next element with the application of clock signal.

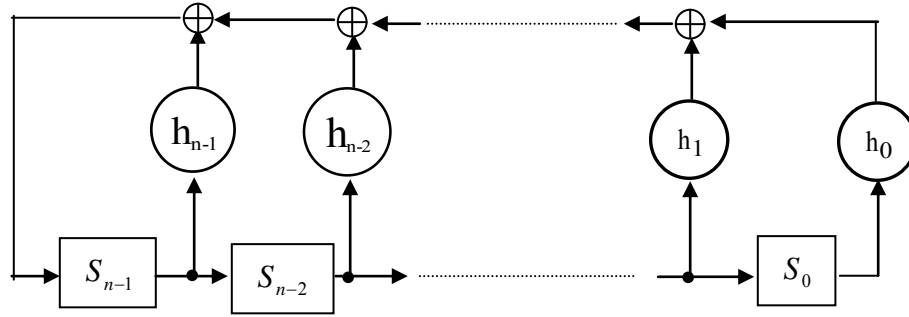


Figure 2.4: General structure of an n-bit LFSR

In Figure 2.4, All the operations are in Galois Field GF(2). $S = (S_0, S_1, \dots, S_{n-1})$, the binary n-tuples, represents the state of the LFSR. It can be represented in the polynomial form as follows:

$$S(x) = S_0 + S_1x + \dots + S_{n-2}x^{n-2} + S_{n-1}x^{n-1}$$

where x^i denotes the i^{th} stage of the LFSR. For example, x^0 represents stage 0, x^1 represents stage 1 and so on. Feedback function of the LFSR is called the feedback polynomial or the generator polynomial and can be represented as follows:

$$h(x) = h_0 + h_1x^1 + + h_{n-1}x^{n-1} + x^n$$

where $h_i \in \{1,0\}$ denotes the feedback tap in the i^{th} stage of the LFSR. $h_i = 0$ means there exists no feedback link in the i^{th} stage whereas $h_i = 1$ means there exists feedback link in that stage.

With the application of clock signal, the LFSR goes into autonomous mode. The past state of the LFSR ($S(x)$) changes to a new state and generates pseudo-random patterns. If the period of the LFSR is u then the LFSR returns to the initial state after u number of shifts. The period (u) of the LFSR depends on the feedback polynomial and initial state. If the initial state is all zero then u will be 1 meaning that the state remains unchanged. Again if the initial state of the LFSR is non-zero and the feedback polynomial is primitive then we become near exhaustive [20]. For an n -stage LFSR with feedback connection based on the primitive polynomial, it goes through all the states except all zero state i.e. $u = 2^n - 1$. The primitive polynomial is discussed in section 2.7.

Figure 2.5 shows a 4-stage LFSR. Its feedback polynomial is $h(x) = 1 + x + x^4$ which is primitive [20]. Assuming that the initial state of the LFSR is 1000, Table 2.1 shows all the states of the LFSR when it is subjected to clock signal.

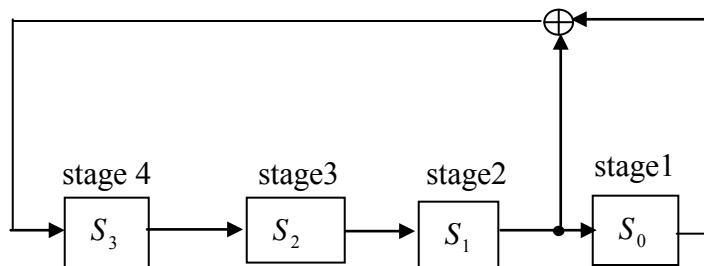


Figure 2.5: A four stage LFSR

Table 2.1: States of the LFSR with feedback polynomial $h(x) = 1 + x + x^4$

states	stage 4	stage 3	stage 2	stage 1
1	1	0	0	0
2	0	1	0	0
3	0	0	1	0
4	1	0	0	1
5	1	1	0	0
6	0	1	1	0
7	1	0	1	1
8	0	1	0	1
9	1	0	1	0
10	1	1	0	1
11	1	1	1	0
12	1	1	1	1
13	0	1	1	1
14	0	0	1	1
15	0	0	0	1
16	1	0	0	0

From the Table 2.1, it is seen that the state of the LFSR repeats after $2^4 - 1 = 15$ clock cycles.

2.5.1 LFSR Terminology

LFSRs sequence through $2N - 1$ states, where N is the number of registers in the LFSR. The contents of the registers are shifted right by one position at each clock cycle. The feedbacks from predefined registers or taps to the leftmost register are XORed together.

LFSRs have several variables:

- The number of stages in the shift registers
- The number of taps in the feedback path
- The position of each tap in the shift registers stage
- The initial starting condition of the shift register, often referred to as the FILL state

In the case of LFSRs with an XOR feedback, the FILL value must be non-zero to avoid the LFSR locking up in the next state.

2.5.1.1 Shift Register Length

The shift register length is often referred to as the degree, and the longer the shift register, the longer the duration of the pseudo-random sequence before it repeats. For a shift register of fixed length N , the number and duration of the sequences it can generate are determined by the number and position of taps used to generate the parity feedback bit.

2.5.1.2 Shift Register Taps

The combination of taps and their location is often referred to as a polynomial, and expressed as $P(x) = X^7 + X^3 + 1$. Various conventions are used to map the polynomial terms to register stages in the shift register implementation. In the polynomial $P(x) = X^7 + X^3 + 1$, the trailing "1" represents X^0 , which is the output of the last stage of the shift register. X^3 is the output of register stage 3 and X^7 the output of the XOR. A few points to note about LFSRs and the polynomial used to describe them:

- The last tap of the shift register is the leading "1" and is always used in the shift register feedback path.
- The length of the shift register can be deduced from the exponent of the highest order term in the polynomial.
- The highest order term of the polynomial is the signal connecting the final XOR output to the shift register input. It does not feed back into the parity calculation along with the other taps identified in the polynomial.

2.6 LFSR Design

There are two major design of LFSR, namely the Galois LFSR (GLFSR) and Fibonacci LFSR (FLFSR). Figure 2.6 and 2.8 shows these two types of LFSR each with characteristics

polynomial $P(x) = 1 + c_1x + c_2x^2 + \dots + c_nx^n$. If a connection exists then $c_i=1$, otherwise $c_i=0$.

The Fibonacci implementation has logic in the feedback path, whereas the Galois implementation has an output that is fed back to selected points in the feed forward path.

2.6.1 Galois Linear Feedback Shift Register (GLFSR)

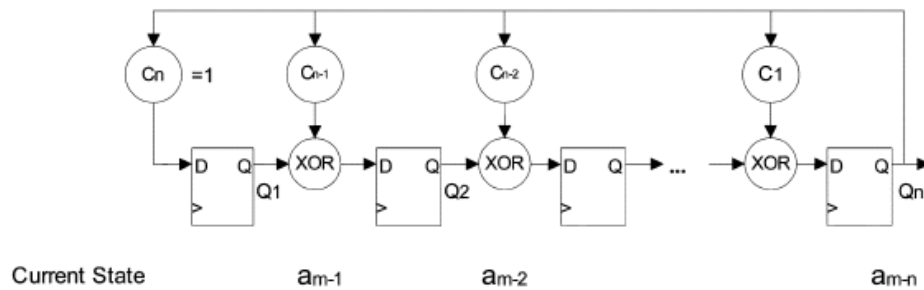


Figure 2.6. Structure of Galois LFSR

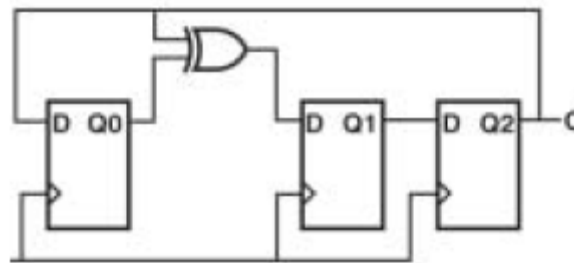


Figure 2.7. 3-bit GLFSR with feedback polynomial $1+x+x^3$

As shown in Figure 2.6, the data flow is from left to right and the feedback path is from right to left. The polynomial increments from left to right with x^0 term (the "1" in the polynomial) as the first term. This is referred to as a Tap polynomial, as it indicates which taps (appropriate taps for maximum-length LFSR is listed in section 3.4.4) are to be fed back from the shift register. Since the XOR gate is in the shift register path, the Galois implementation is also known as an in-line or modular type (M-type) or one-to-many LFSR.

2.6.2 Fibonacci Linear Feedback Shift Register (FLFSR)

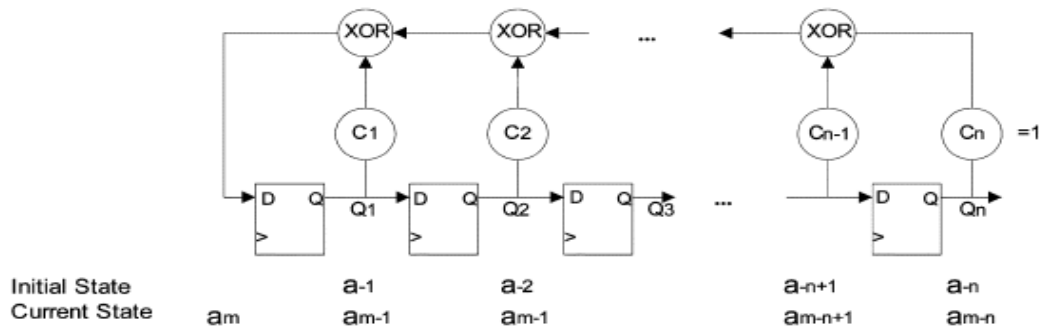


Figure 2.8. Structure of Fibonacci LFSR

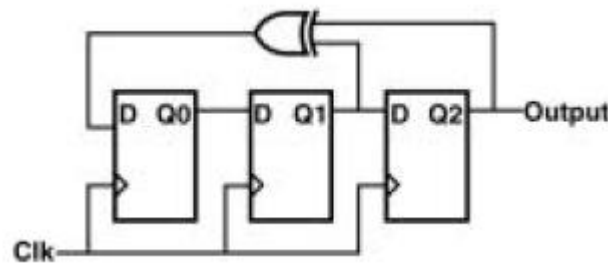


Figure 2.9. 3-bit FLFSR with feedback polynomial $1+x^2+x^3$

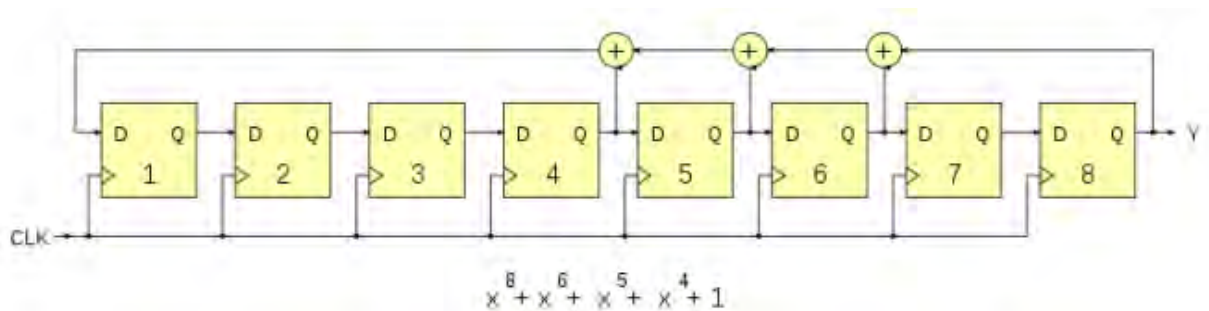


Figure 2.10. 8-bit FLFSR with feedback polynomial $1+x^4+x^5+x^6+x^8$

In Figure 2.8, the data flow is from left to right and the feedback path is from right to left, similar to the Galois implementation. However, in Fibonacci implementation polynomial decrements from left to right with x^0 as the last term in the polynomial. This polynomial is referred to as a Reciprocal Tap polynomial and the feedback taps (appropriate taps for maximum-length LFSR is listed in section 3.4.4) are incrementally annotated from right to left along the shift register. Since the XOR gate is in the feedback path, the Fibonacci implementation is also known as an

out-of-line or simple type (S-type) or many-to-one LFSR. In this project we are focusing on this type of LFSR for our experiments.

The two 7-bit LFSRs with feedback polynomial $1+x+x^3+x^6+x^7$ cycle through the following states when the registers are initially loaded with 0000001:

	Galois	Fibonacci
t	g7...g1	f7...f1
0	0000001	0000001
1	0000010	1000000
2	0000100	1100000
3	0001000	1110000
4	0010000	1111000
5	0100000	0111100
6	1000000	1011110
7	1001011	1101111
8	1011101	0110111
9	1110001	0011011
10	0101001	1001101
11	1010010	1100110
12	1101111	0110011
13	0010101	0011001
14	0101010	0001100
15	1010100	1000110
16	1100011	0100011
17	0001101	0010001
18	0011010	1001000
19	0110100	0100100
20	1101000	0010010

2.7 Primitive Polynomial

In an n-stage LFSR, it is possible to have many possible combination of feedback taps and thereby many corresponding feedback polynomials. It has been mentioned that the period of the LFSR depends on the feedback polynomials. If the sequence of pseudo-random numbers generated by an n-stage LFSR has period (2^n-1) , then it is called maximum length LFSR (m-

sequence LFSR) and the corresponding feedback polynomial is called primitive polynomial. A primitive polynomial is a polynomial that generates all elements of an extension field from a base field. Primitive polynomials are also irreducible polynomials. For any prime or prime power q and any positive integer n , there exists a primitive polynomial of degree n over $GF(q)$ There are

$$a_q(n) = \frac{\phi(q^n - 1)}{n} \quad (2.5)$$

primitive polynomials over $GF(q)$, where $\phi(n)$ is the totient function.

A polynomial of degree n over the finite field $GF(2)$ (i.e., with coefficients either 0 or 1) is primitive if it has polynomial order $2^n - 1$. For example, $x^2 + x + 1$ has order 3 since

$$\frac{x+1}{x^2+x+1} = \frac{x+1}{x^2+x+1} \pmod{2} \quad (2.6)$$

$$\frac{x^2+1}{x^2+x+1} = 1 + \frac{x}{x^2+x+1} \pmod{2} \quad (2.7)$$

$$\frac{x^3+1}{x^2+x+1} = x+1 \pmod{2}. \quad (2.8)$$

Plugging in $q=2$ to equation (1), the numbers of primitive polynomials over $GF(2)$ are

$$a_2(n) = \frac{\phi(2^n - 1)}{n}, \quad (2.9)$$

giving 1, 1, 2, 2, 6, 6, 18, 16, 48, ... (Sloane's [A011260](#)) for $n=1, 2, \dots$. The Table 2.2 lists the primitive polynomials (mod 2) of orders 1 through 5.

For an n -stage LFSR, the maximum possible number of n^{th} degree feedback polynomials is 2^n but the number of n^{th} degree primitive polynomials are much less than 2^n .

Table 2.2: Primitive polynomials (mod 2) of orders 1 through 5

n	primitive polynomials
1	$1 + x$
2	$1 + x + x^2$
3	$1 + x + x^3, 1 + x^2 + x^3$
4	$1 + x + x^4, 1 + x^3 + x^4$
5	$1 + x^2 + x^5, 1 + x + x^2 + x^3 + x^5, 1 + x^3 + x^5,$ $1 + x + x^3 + x^4 + x^5, 1 + x^2 + x^3 + x^4 + x^5,$ $1 + x + x^2 + x^4 + x^5$

2.8 Coefficient of Variation (CV)

Coefficient of variation denoted as CV is relative measure in statistics. This measure developed by Karl Pearson is the most commonly used measure of relative variation. It is used in such problems where we want to compare the variability of two or more than two series. The series for which the coefficient of variation is greater is said to be more variable or conversely less consistent, less uniform, less stable, less homogeneous. On the other hand, the series for which the coefficient of variation is less is said to be less variable or more consistent, more uniform, more stable, more homogeneous [21]. CV is obtained as follows:

$$CV = \frac{\sigma}{\bar{X}} \times 100$$

where σ = Standard Deviation and

\bar{X} = Arithmetic Mean

Here we will describe details about arithmetic mean and standard deviation. The most popular and widely used measure for representing the entire data by one value is what the arithmetic mean, often referred to as mean in statistics. Its value is obtained by adding together all the observations and by dividing this total by the number of observations. Thus if X_1, X_2, \dots, X_N represent the values of N items or observations, the arithmetic mean denoted by \bar{X} (read as Xbar) is defined as :

$$\bar{X} = \frac{X_1 + X_2 + \dots + X_N}{N} = \frac{\sum X}{N}$$

The standard deviation is by far the most important and widely used measure of studying variation. It is a measure of how much “spread” or “variability” is present in data. If all the numbers in the data series are very close to each other, the standard deviation is close to zero. If the numbers are well dispersed, the standard deviation will tend to be large. Standard deviation is denoted by the small Greek letter σ (read as sigma) and is defined as :

$$\sigma = \sqrt{\frac{\sum (X - \bar{X})^2}{N}}$$

If we square standard deviation, we get what is called Variance.

Hence Variance = σ^2 or $\sigma = \sqrt{\text{Variance}}$

The standard deviation measures the absolute variation of a distribution; the greater amount of variation, the greater the standard deviation. A small standard deviation means a high degree of uniformity of the observations as well as homogeneity of a series, a large standard deviation means just the opposite.

2.9 Summary

Different techniques for testing digital IC have been elaborated in this chapter. It can be concluded from the review of different test techniques that the mixed mode approach

outperforms all other existing test technologies in terms of simplicity in hardware implementation and control complexity, encoding efficiency, test application time and data storage requirements. Pseudo-random vector generation using the LFSR has also been discussed and shown that an LFSR having feedback connection based on primitive polynomial generates maximal length sequence.

CHAPTER 3

EVALUATION TECHNIQUE FOR TEST PATTERN GENERATOR

3.1 Introduction

This chapter presents the fault simulation technique for the evaluation of the proposed Fibonacci test pattern generator. It also describes all the required tools for the above purpose.

3.2 Basic Test Arrangement

Now a days mixed-mode technique outperforms all other test techniques in VLSI testing. This technique used in this project is a hybrid test technique of deterministic test approach followed by pseudo-random test approach. This approach takes advantages of both the pseudo-random test approach and deterministic test approach. In mixed mode approach, Circuit Under Test (CUT) is first subjected under pseudo-random testing mode and then at an optimum point of fault coverage it is switched to deterministic test mode. In this approach, pseudo-random test pattern generation and proper switching point from pseudo-random test mode to deterministic test mode are very important to make the testing process cost effective. This approach ensures complete fault coverage while offers reduced storage requirements, shorter test application time, and smaller area overhead compared to weighted random approach.

The basic arrangement for testing a digital IC is shown in Figure 3.1. A test pattern generator generates a vector of binary inputs that are applied to the circuit-under-test (CUT). For each input vector, the response is measured and compared with the expected output. More than a

single input vector is needed to adequately test the CUT, so a test vector set is created to determine the functionality of the chip.

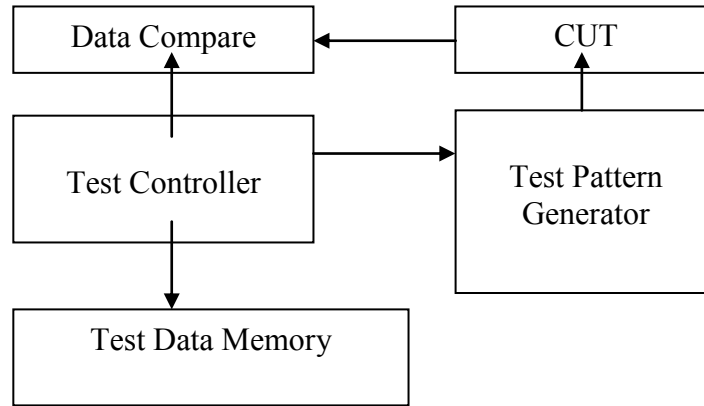


Figure 3.1: General IC testing process

Since each measurement takes time, it is needed a minimal test vector set to reduce the total time to determine if the chip is functional or not. Test vector generation is one of the more challenging aspects of testing.

3.3 Benchmark Circuits

To evaluate the performance of mixed-mode technique in IC testing, fault simulation experiments have been conducted on ISCAS benchmark circuits. ISCAS benchmark circuits are a set of combinational circuits focusing on different level of IC complexities. They are a group of well-defined, gate level netlist and functions based on common building blocks. The circuits were presented in the international symposium on circuits and systems held on 1985[22, 23]. These circuits are proposed with the objective of evaluating performance of test pattern

generation algorithm's, fault simulation, testability analysis, formal verification, logic synthesis, design verification, test generation, clock distribution, power consumption, timing analysis, technology mapping and layout synthesis. Its main advantage is that researchers all over the world can work on a common set of problems, compare their results and cooperate with each other. The HDL code of the ISCAS benchmark circuits is available on the website [24]. General characteristics of ISCAS benchmark circuits are presented in Table 3.1.

Table 3.1: ISCAS Benchmark Circuits

Circuits	Circuit Function	*NG	*NL	*NS	*NPI	*NPO	*NF
c432	Priority Decoder	160	432	89	36	7	802
c499	Error Correcting Circuit	221	499	59	41	32	1306
c880	ALU and Control	383	880	125	60	26	1428
c1355	Error Correcting Circuit	546	1355	259	41	32	1970
c1908	Error Correcting Circuit	880	1908	385	33	25	1282
c2670	ALU and Control	1193	2670	454	233	140	2588
c3540	ALU and Control	1669	3540	579	50	22	2988
c5315	ALU and Selector	2307	5315	806	178	123	5640
c6288	Multiplier	2517	6288	1456	32	32	9804

*NPI= Number of Primary Inputs, *NG= Number of Gates, *NS=Number of Stems, *NPO= Number of Primary Outputs, *NL= Number of Lines, *NF= Total Number of Faults

3.4 Design of 64-bit FLFSR

A 64-bit FLFSR has been proposed in this project. Table 3.2 lists the appropriate taps for maximum-length LFSR counters of up to 168 bits taken from Xilinx Data Books. An LFSR with a well-chosen feedback function is capable of generating sufficient long test pattern.

3.4.1 64-bit Fibonacci Linear Feedback Shift Register

In order for an LFSR to iterate through its largest possible sequence of values, it must use a polynomial which will produce such a sequence. The tap positions shown in Figure 3.2 will produce maximum sequence lengths for the proposed 64-bit FLFSR. The design uses the Fibonacci approach to implement test pattern generator.

3.4.2 Feedback

The LFSR feedback network performs modulo-2 summation as was discussed in Chapter 2. These summations can be performed with either XOR or XNOR gates in the logic. However XOR gate is used in this project. The design uses Fibonacci configuration of test pattern generator. The output FB_Out is fed back to the first stage of the FLFSR. Figure 3.3 shows the feedback logic using XOR gates for an LFSR with 4 taps. The tap numbers in Figure 3.2 are taken from Table 3.2 for a 64-stage LFSR. Figure 3.3 shows the complete FLFSR arrangement with feedback tap positions Q60, Q61, Q63, Q64, hence the characteristics polynomial is $1+X^{60}+X^{61}+X^{63}+X^{64}$.

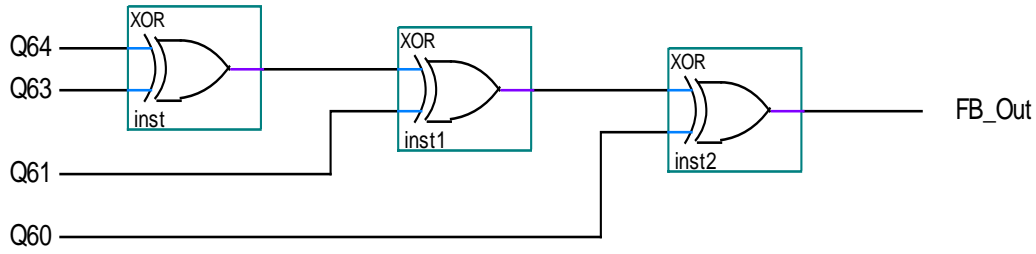


Figure 3.2. Feedback Logic using XOR gates

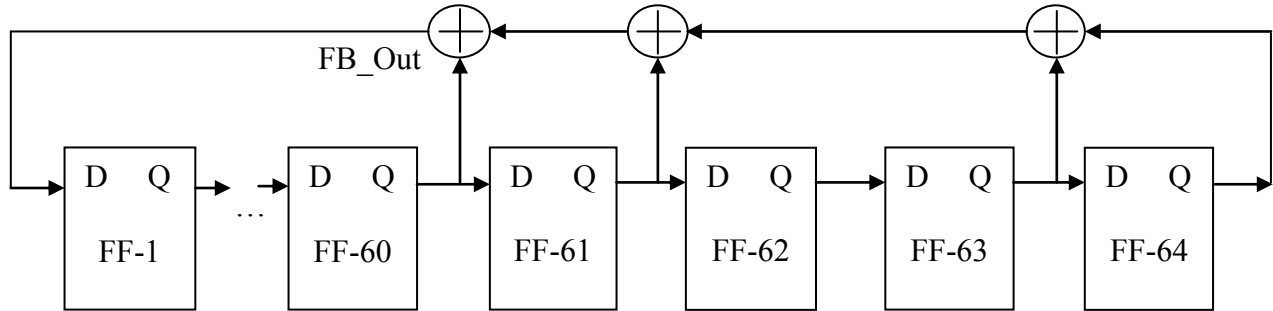


Figure 3.3. 64-bit FLFSR with feedback logic

From Figure 3.3 we see that the outputs of the 60th, 61st, 63rd, 64th stages are XORed and fed back to first stage of the proposed 64-bit FLFSR.

3.4.3 Conventions for Feedback Tap Specification

A given set of feedback connections can be expressed in a convenient and easy-to-use shorthand form, with the connection numbers being listed within a pair of square brackets. In doing so, connection g_0 (defined in Figure 3.4) is implied, and not listed, since it is always connected. Although g_m is also always connected, it is listed in order to convey the shift register size (i.e. the number of registers).

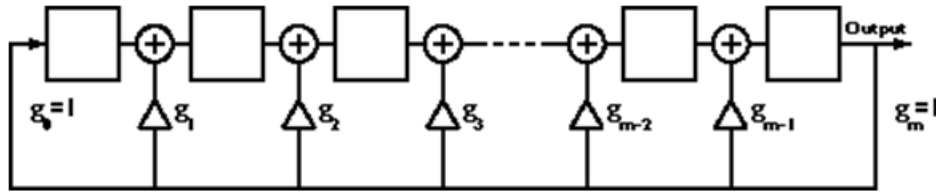


Figure 3.4. Galois implementation of LFSR

Specifically, a set of feedback connections, or taps, is denoted as

$$[f_1, f_2, f_3, \dots, f_J]$$

where subscript J is the total number of feedback taps (not including g_0), $f_1 = m$ is the highest-order feedback tap (and the size of the LFSR), and f_j represent the remaining feedback taps. The value of each f_j is equal to the subscript of the corresponding connection g . Note that the tap numbers f_j are customarily arranged in descending order from left to right.

A set of feedback taps specified in this format is called a *feedback tap set*, *feedback set*, or *feedback pattern*. As an example, the [8, 4, 3, 2] feedback set would signify an eight-stage shift register with feedback connections at taps g_8 , g_4 , g_3 , g_2 , and, as always, at g_0 .

A related convention is that an LFSR with m shift register stages is said to be an R_m LFSR. For example, an R_8 generator is one with eight stages. An alternative to this convention is PN_m , or PN_8 in this example. (PN is an acronym for pseudonoise, which is a term used in some industries for maximal length pseudorandom sequences.)

3.4.4 Taps of Linear Feedback Shift Register

Table 3.2 lists the appropriate taps for maximum-length LFSR counters of up to 168 bits. The basic description and the table for the first 40 bits was originally published in XCELL and

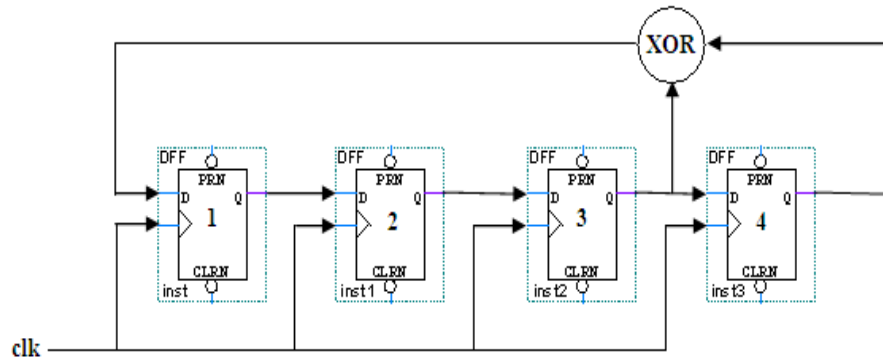
reprinted on page 9-24 of the 1993 and 1994 Xilinx Data Books. Responding to repeated requests, the list is here extended to 168 bits. This information is based on unpublished research done by Wayne Stahnke while he was at Fairchild Semiconductor in 1970.

Table 3.2: Taps for maximum-length LFSR counters

n	XOR from	n	XOR from	n	XOR from	n	XOR from
3	3,2	45	45,44,42,41	87	87,74	129	129,124
4	4,3	46	46,45,26,25	88	88,87,17,16	130	130,127
5	5,3	47	47,42	89	89,51	131	131,130,84,83
6	6,5	48	48,47,21,20	90	90,89,72,71	132	132,103
7	7,6	49	49,40	91	91,90,8,7	133	133,132,82,81
8	8,6,5,4	50	50,49,24,23	92	92,91,80,79	134	134,77
9	9,5	51	51,50,36,35	93	93,91	135	135,124
10	10,7	52	52,49	94	94,73	136	136,135,11,10
11	11,9	53	53,52,38,37	95	95,84	137	137,116
12	12,6,4,1	54	54,53,18,17	96	96,94,49,47	138	138,137,131,130
13	13,4,3,1	55	55,31	97	97,91	139	139,136,134,131
14	14,5,3,1	56	56,55,35,34	98	98,87	140	140,111
15	15,14	57	57,50	99	99,97,54,52	141	141,140,110,109
16	16,15,13,4	58	58,39	100	100,63	142	142,121
17	17,14	59	59,58,38,37	101	101,100,95,94	143	143,142,123,122
18	18,11	60	60,59	102	102,101,36,35	144	144,143,75,74
19	19,6,2,1	61	61,60,46,45	103	103,94	145	145,93
20	20,17	62	62,61,6,5	104	104,103,94,93	146	146,145,87,86
21	21,19	63	63,62	105	105,89	147	147,146,110,109
22	22,21	64	64,63,61,60	106	106,91	148	148,121
23	23,18	65	65,47	107	107,105,44,42	149	149,148,40,39
24	24,23,22,17	66	66,65,57,56	108	108,77	150	150,97
25	25,22	67	67,66,58,57	109	109,108,103,102	151	151,148
26	26,6,2,1	68	68,59	110	110,109,98,97	152	152,151,87,86
27	27,5,2,1	69	69,67,42,40	111	111,101	153	153,152
28	28,25	70	70,69,55,54	112	112,110,69,67	154	154,152,27,25
29	29,27	71	71,65	113	113,104	155	155,154,124,123
30	30,6,4,1	72	72,66,25,19	114	114,113,33,32	156	156,155,41,40
31	31,28	73	73,48	115	115,114,101,100	157	157,156,131,130
32	32,22,2,1	74	74,73,59,58	116	116,115,46,45	158	158,157,132,131
33	33,20	75	75,74,65,64	117	117,115,99,97	159	159,128
34	34,27,2,1	76	76,75,41,40	118	118,85	160	160,159,142,141
35	35,33	77	77,76,47,46	119	119,111	161	161,143
36	36,25	78	78,77,59,58	120	120,113,9,2	162	162,161,75,74
37	37,5,4,3,2,1	79	79,70	121	121,103	163	163,162,104,103
38	38,6,5,1	80	80,79,43,42	122	122,121,63,62	164	164,163,151,150
39	39,35	81	81,77	123	123,121	165	165,164,135,134
40	40,38,21,19	82	82,79,47,44	124	124,87	166	166,165,128,127
41	41,38	83	83,82,38,37	125	125,124,18,17	167	167,161
42	42,41,20,19	84	84,71	126	126,125,90,89	168	168,166,153,151
43	43,42,38,37	85	85,84,58,57	127	127,126		
44	44,43,18,17	86	86,85,74,73	128	128,126,101,99		

3.5 Illustration of FLFSR Working

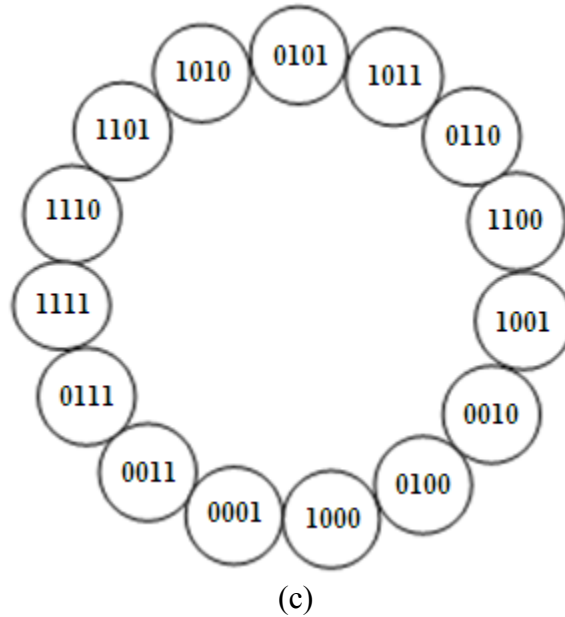
Consider a 4-bit Fibonacci LFSR with characteristics polynomial $1+X^3+X^4$. The XOR gate provides feedback to the register that shifts bits from left to right. The maximal sequence consists of every possible state except the "0000" state.



(a)

clock	Q1	Q2	Q3	Q4
1	1	0	0	0
2	0	1	0	0
3	0	0	1	0
4	1	0	0	1
5	1	1	0	0
6	0	1	1	0
7	1	0	1	1
8	0	1	0	1
9	1	0	1	0
10	1	1	0	1
11	1	1	1	0
12	1	1	1	1
13	0	1	1	1
14	0	0	1	1
15	0	0	0	1
16	Repeat			

(b)



(c)
Figure 3.5. (a) Block Diagram, (b) truth table, (c) state diagram of 4-bit FLFSR with characteristics polynomial $1+X^3+X^4$

Note that in Figure 3.5(a) the outputs of 3rd and fourth stage of the LFSR are XORed and are fed back to the first stage. Since the LFSR has four stages, the truth table in Figure 3.5(b) shows that it has 15 different states. After 15th clock cycle the LFSR repeats its states. The 15 distinct states of the LFSR are also depicted with the state diagram in Figure 3.5(c).

3.6 Test Pattern Generation using FLFSR

Algorithm of the program for generating random numbers is given below.

Algorithm:

```

Read the number of stages (m), number of feedback taps, and their position in the FLFSR;
Read reseeding position;
Initialize the stage with the entered position to 1 and all other stages to 0;
Read the number of test vectors and number of primary inputs of CUT;
Loop1
{

```

```

Store the values of the tap positions of the FLFSR into temporary variables;
Loop2
{
    Store the values of the tap positions of the FLFSR into temporary variables;
    Loop3
    {
        Assign the value of (m-i-2)th stage of the FLFSR to (m-i-1)th stage;
        Repeat the process equal to the number of stages of the FLFSR;
    }
    Calculate the XORED output of the temporary variables and assign it to first stage
    of the FLFSR;
    Write the value of the (m-1)th stage of the FLFSR in the test vector file;
    Repeat the process equal to the number of primary inputs of the FLFSR;
}
}
Move to the next line of the test vector file and repeat Loop1 equal to the number of test
vectors;
End;

```

Figure 3.5 shows the flowchart of test vector file generation using the proposed 64-bit FLFSR in this project. This will give the reader a better understanding of how test patterns are generated using the proposed technique.

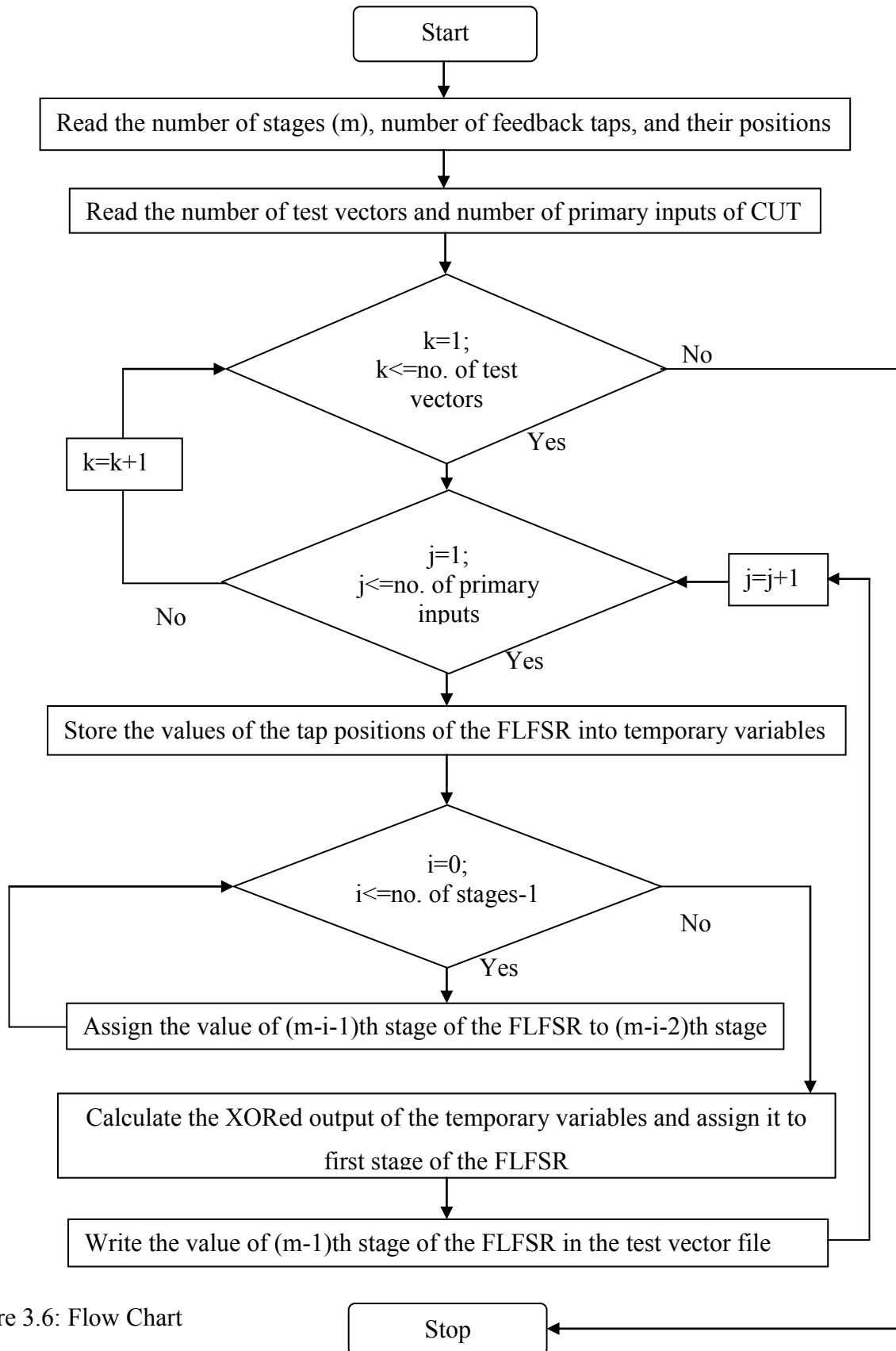
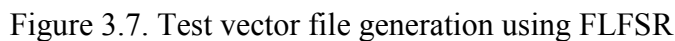


Figure 3.6: Flow Chart



3.7 Techniques for Determining Best Seed

An LFSR generates a PRV sequence for a particular seed and achieves reasonable fault coverage. To achieve highest percentage of fault coverage, PRV sequences are generated for different seeds. It is observed that for a particular seed the LFSR achieves the highest fault coverage as compared to other seeds. This particular seed is called the best seed or proper seed in the context of IC testing. One would simply ask that why the fault coverage is high for that particular seed. The answer can be easily given in terms of coefficient of variation (CV) discussed in chapter 2. Theoretically, the seed for which CV is maximum will be the best seed because the fault coverage is the highest for that seed.

Algorithm of the MATLAB program for measuring CV is given below.

Algorithm:

Read the random sequence with 1 and 0 for a particular seed.
Compute the Standard Deviation (σ) of the random sequence.
Compute the Mean (\bar{X}) of the random sequence.
Divide σ by \bar{X} and multiply the result by 100.

Figure 3.8 shows the Flowchart for the measurement of CV.

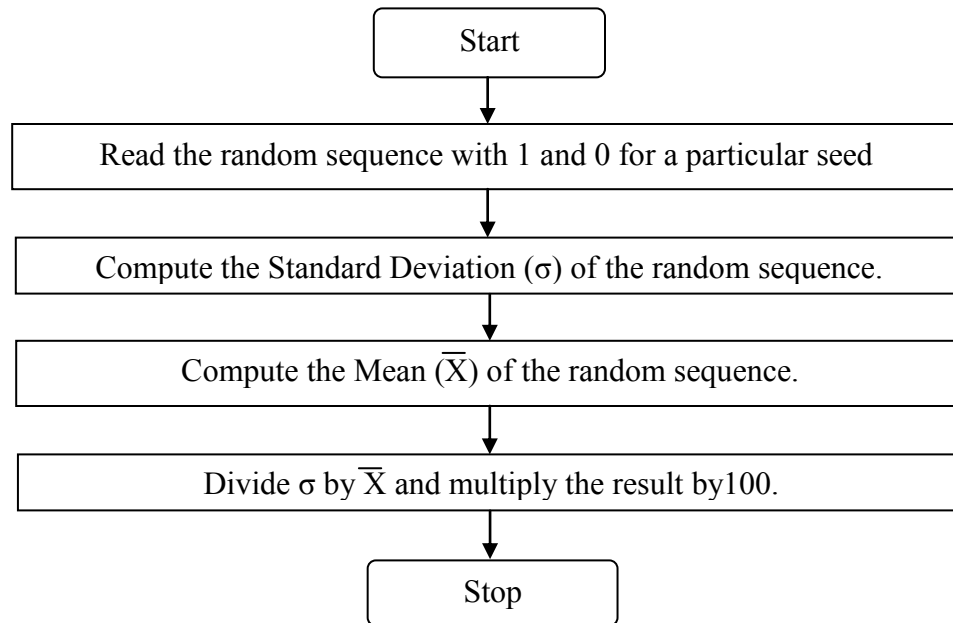


Figure 3.8: Flowchart for measuring CV

3.8 Software Requirements for Proposed Work

In this section we will discuss about different software used in this project to perform the simulation works.

3.8.1 Fault Simulator (FSIM)

FSIM is a fault simulator for combinational circuits. It employs the parallel pattern single fault propagation technique. FSIM has been developed in the Bradley Department of Electrical Engineering, Virginia Polytechnic Institute & State University (VPI&SU) and the copy right belongs to VPI&SU. A FSIM emulates the behavior of a circuit in the presence of the given set of faults. Because of its inherent low-complexity, the fault simulator serves many purposes in VLSI testing. In fact, as the circuit simulator in circuit design, the fault simulator is regarded as the most fundamental tool in testing.

The FSIM is the most fundamental tool in VLSI testing. It serves many applications. We will summarize the major applications in the following.

- (1) The most prominent application is to evaluate the quality of a given test set for a circuit under test (CUT). The fault simulator simulates the circuit under different inserted faults and determines the number of faults detected by the test set, usually in terms of fault coverage which is the percentage of detected faults over all considered faults. The test engineer then decides whether the obtained fault coverage is acceptable or not.
- (2) If the fault coverage is lower than expectation, more test patterns must be included into the test set. These extra patterns are preferably generated by an automatic test pattern generator (ATPG). Here the fault simulator also plays an important role. The simulator is used to find out the detected faults for each newly generated test pattern from ATPG, thereby significantly reduces the CPU time of high-complexity ATPG.
- (3) In VLSI testing, sometimes we need to investigate why the tested IC fails. The first step of such diagnosis is to locate the defect. Although the fault is only the behavioral model of defects, the location of fault generally gives a good indicator of defect location. To find out the fault location for a failed IC, we first need to construct a fault dictionary, with the help of a fault simulator, which lists the complete faulty behavior of all considered faults.
- (4) In addition, a fault simulator serves the application of reliability analysis of ICs, in which the faulty behavior from considered faults is simulated to see whether or not the faults adversely affect the function or performance of a system. In summary, fault simulation plays an essential role in VLSI testing. Its efficiency determines the performance of the above applications. As the size of circuits and test sets grow increasingly, the demand of efficient fault simulation algorithm becomes even more acute.

3.8.2 MATLAB

MATLAB (**matrix laboratory**) is a numerical computing environment and fourth-generation programming language. Developed by MathWorks, MATLAB allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages. It is a high-level language and interactive environment for numerical computation, visualization, and programming. Using MATLAB, one can analyze data, develop algorithms, and create models and applications. The language, tools, and built-in math functions enable one to explore multiple approaches and reach a solution faster than with spreadsheets or traditional programming languages, such as C/C++ or Java. MATLAB can be used for a range of applications, including signal processing and communications, image and video processing, control systems, test and measurement, computational finance, and computational biology. More than a million engineers and scientists in industry and academia use MATLAB, the language of technical computing.

3.9 Summary

This chapter mainly discusses the design and functional operation of the proposed system. Necessary software and hardware to implement the design are also described. The test pattern generator is capable of generating PRVs of sufficient length since it is a 64-bit LFSR having feedback connection based on the primitive polynomial.

CHAPTER 4

RESULTS AND DISCUSSIONS

4.1 Introduction

This chapter presents the fault simulation results of different ISCAS benchmark circuits using the pseudo random test vectors generated by the proposed 64-bit FLFSR. FSIM is used to carry out the simulation process.

4.2 Fault Simulation

„FSIM“ is a digital fault simulator [25] which is used for conducting the fault simulation experiments on the ISCAS benchmark circuits. It is a process by which fault coverage of a circuit is determined for a set of test pattern. It shows that if an LFSR is initialized with proper seed, it generates PRV sequences, which detect the maximum faults of a CUT using lower number of PRVs than that of using other seeds of the same LFSR. In this project, similar experiments have been carried out on the ISCAS benchmark circuits using fault simulator to find the appropriate seed of the FLFSR for generating PRV sequences with better fault detection capability.

A computer program has been developed using C programming language to generate PRV for fault simulation experiments. The program represents the FLFSR that is used in the proposed system. It can generate PRV of any predefined number of test length by user and can store in a file. The source code of the program has been presented in Appendix A1. The commands by which fault simulation experiments have been conducted using FSIM is specified in Appendix A2. When the program is executed, it generates a test vector file. Appendix A3 shows a sample of test vectors for benchmark circuit c432.bench.

In IC testing, maximum fault coverage using minimum test application time is the most desirable. Researchers have shown that better randomness and less correlation in the PRV sequences result in better fault detection capability [26-28]. Fault simulation experiments on ISCAS benchmark circuits has been carried out using FSIM to find the best seed of the LFSR for generating PRV sequences with better fault detection capability.

Since in the mixed mode approach, deterministic test is followed by pseudo-random test, the performance of IC testing in this approach largely depends on the optimum switching from the pseudo-random test mode to the deterministic test mode. The fewer the number of deterministic test vectors, the less the data storage requirements. Fault detection profile of the PRV sequences for a circuit helps to determine the appropriate switching moment from the pseudo-random test mode to the deterministic test mode. In this project, PRV sequences have been generated using best seed of the FLFSR for the benchmark circuits and then fault simulation experiments have been conducted with the PRV sequences to determine its fault detection profile for the benchmark circuits using the FSIM.

4.3 Fault Simulation Results of the ISCAS Benchmark Circuits

Pseudo-random testing is a cost-effective means of testing VLSI circuits. Using Fibonacci pseudo-random test patterns it is possible to achieve a maximum percentage of fault coverage only applying fewer number of test vectors. This fact has been verified in this project. Fault simulation experiments using FLFSR have been conducted out on the ISCAS Benchmark circuits. The fault simulation results have been compared with that of other researchers [2-8].

The technique of maximization of fault detection using PRV sequences is presented in the literature [29] where „FSIM“ digital fault simulator [25] is used in conducting the fault simulation experiments on the ISCAS benchmark circuits.

Forty different seeds have been used to generate PRV sequences. The PRV sequences are applied to the benchmark circuits and fault coverage (%) versus number of PRVs are measured with respect to every seed. Figure 4.1 shows the fault detection profile of the PRV sequences for the benchmark circuit c432.bench.

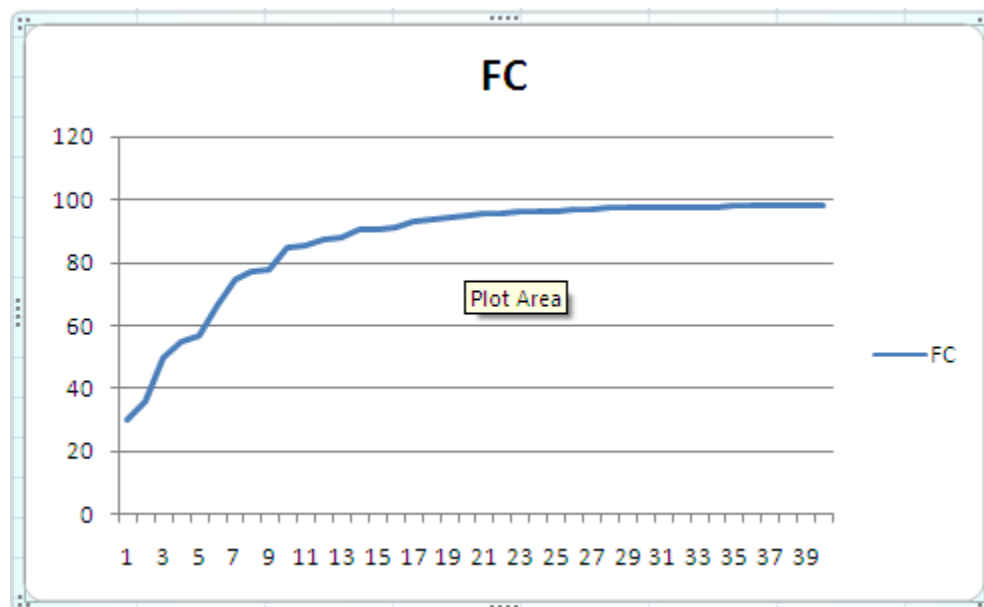


Figure 4.1: Fault detection profile of PRV for the benchmark circuit c432.bench

Figure 2.1 in section 2.3.4 shows the relationship between fault coverage and pseudo-random test vectors applied to a typical circuit. Here Figure 4.1 shows that the first few PRVs detect the maximum faults of the circuit c432.bench. Then the slope of the fault detection profile of the PRV rapidly decreases with the increase of number of test vectors. More than 80 % fault coverage is achieved using only 50 test vectors. These faults are ETD faults. After the detection of the ETD faults, much higher number of test vectors is needed to detect the remaining faults.

These remaining faults are HTD faults and random resistant faults. These faults cause potential difficulties in achieving acceptable fault coverage in the pseudo-random testing of IC. Fault detection profile of the PRV sequences for the rest of the benchmark circuits is similar to that as shown in Figure 4.1. It clearly indicates that with the increase of number of test vectors, increase of fault coverage sharply decreases and goes towards zero. When the increment of fault coverage is very low or almost zeros then the mode of test is switched from the pseudo-random test to the deterministic test. For example, in the simulation result as shown in Appendix A4, when number of PRV is 200 for circuit c432.bench then it is appropriate to switch from the pseudo-random test mode to the deterministic mode. Note that the best seed is represented using the bold letter in the table in Appendix A4. Fault simulation results for the rest of the benchmark circuits follow the similar profile.

Another experiment has been carried out on ISCAS Benchmark Circuit c432.bench. Three different seeds have been chosen arbitrarily and with respect to every seed, a set of test vectors have been generated. The fault simulation results have been presented in Table 4.1. From this table it can be proved that maximum percentage of faults in the circuit can be detected by using small number of test vectors and the rate of fault coverage decreases with the increase of number of test vectors. Almost 80-90% fault coverage can be achieved applying only 60-70 test vectors. The main reason of this is that the most of the faults in the circuit are easy to detect but there are some other faults which are hard to detect. These undetectable faults are called PRV resistant faults. Table 4.1 also compares the fault detection profile of the test vectors generated with respect to three different seeds of the FLFSR. This shows that a significant improvement in the fault coverage can be gained by changing the seed of the FLFSR.

Table 4.1: Fault coverage for different PRVs for benchmark circuit c432.bench

PRV	FC1 (%)	FC2 (%)	FC3 (%)
10	49.05	33.21	36.26
20	62.21	46.18	55.15
30	71.37	67.18	66.22
40	75.38	75.95	77.29
50	84.54	82.06	84.73
60	86.83	84.54	87.77
70	88.74	87.41	90.65
80	91.03	90.08	91.6
90	91.99	90.65	93.89
100	93.51	93.89	95.23

Abbreviations used in the following Table 4.1.

PRV= Pseudo Random test Vector

FC1= Fault coverage for seed1

FV2= Fault coverage for seed2

FC3= Fault coverage for seed3

Experiments are successful to use two sample of 64 degree feedback polynomial such as $1+x^{60}+x^{61}+x^{63}+x^{64}$ and $1+x+x^3+x^4+x^{64}$ on achieving full fault coverage. For any of these feedback polynomials, a test vector file is generated. FC Comparison is shown in Table 4.2.

Table 4.2: FC (%) Comparison for two different feedback polynomials

ISCAS Benchmark Circuits	NTV	Polynomial1 ($1+x+x^3+x^4+x^{64}$)	Polynomial2 ($1+x^{60}+x^{61}+x^{63}+x^{64}$)
c432	200	98.09	98.28
c499	190	96.17	96.70
c880	120	90.13	93.52
c1355	180	91.49	92.00
c1908	880	94.41	96.33
c2670	250	82.71	83.55
c3540	540	91.80	91.57
c5315	560	98.30	98.24
c6288	60	99.47	99.11

NTV=No. of TV

To analyze the effect of reseeding and polynomial programmability on achieving full fault coverage, experiments have also been performed on different ISCAS benchmark circuits. Two sample of 64 degree feedback polynomial such as $1+x^{60}+x^{61}+x^{63}+x^{64}$ and $1+x+x^3+x^4+x^{64}$ have been chosen. The seed of an FLFSR is defined as the initial value of the stages of the FLFSR before starting to generate the test vectors. Forty different seeds have been used to generate PRV sequences. The PRV sequences are applied to the benchmark circuits and fault coverage versus number of PRV are measured with respect to every seed. For seed of the FLFSR in the experiment, one of the stages of the FLFSR has set to „1“and others to „0“and in this project „1“ is mentioned as seed for simplicity. For any of the feedback polynomials, a test vector file is generated for different seeds. Figure 4.2 and 4.3 represent fault simulation results of the circuit c432.bench for the two selected feedback polynomials. Figure 4.2 shows that fault detection capability of the PRV sequences for the benchmark circuits varies with the seed of the FLFSR. It is possible to determine the best seed of the FLFSR for the benchmark circuits out of the given

seeds. The best seed of the FLFSR produces the highest fault coverage using lowest number of PRV sequences. For example, seed number „24“ in Figure 4.2 can be considered the best seed of the FLFSR for the benchmark circuit c432.bench. The best seed of the FLFSR for the benchmark circuit c432.bench is highlighted using bold letter in the Appendix A4. Similarly, Figure 4.4, Figure 4.6, Figure 4.8, Figure 4.10, Figure 4.12, Figure 4.14, Figure 4.16 and Figure 4.18 represent fault simulation results highlighting the best seed of the circuits c499.bench, c880.bench, c1355.bench, c1908.bench, c2670.bench, c3540.bench, c5315.bench, c6288.bench respectively. The tabulated results are presented in Appendices A4 to A21.

In Figure 4.2, best seed of circuit c432.bench for feedback polynomial $1+x^{60}+x^{61}+x^{63}+x^{64}$ is indicated by the arrow sign where the fault coverage is 98.28% and the required number of test vectors is 200. At this point of testing, pseudo-random test mode can be switched to deterministic test mode.

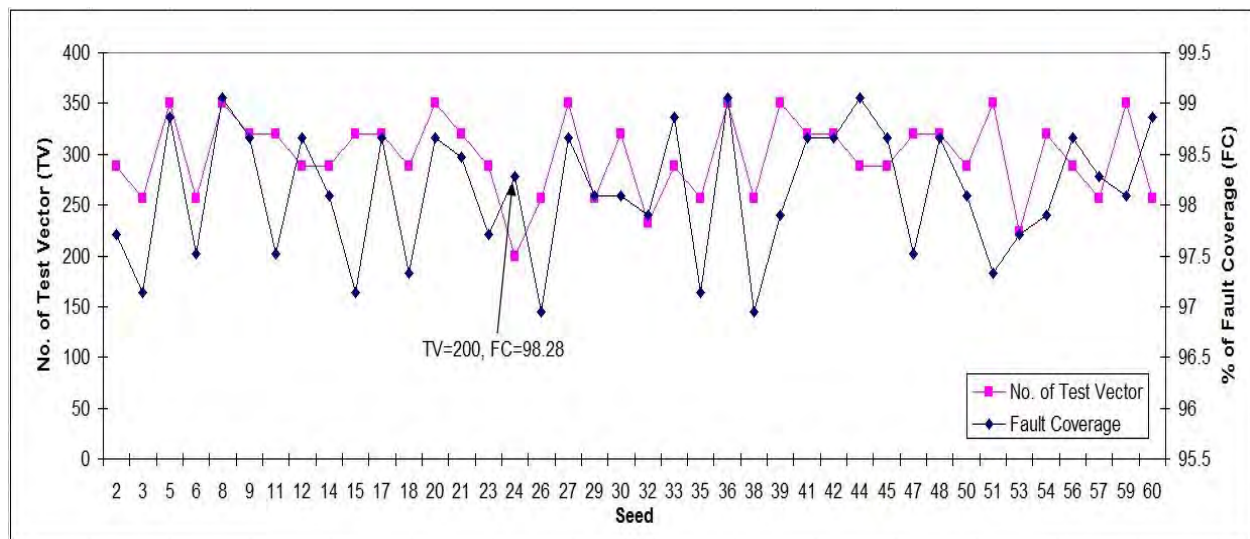


Figure 4.2. Fault simulation result of circuit c432.bench (for feedback polynomial $1+x^{60}+x^{61}+x^{63}+x^{64}$)

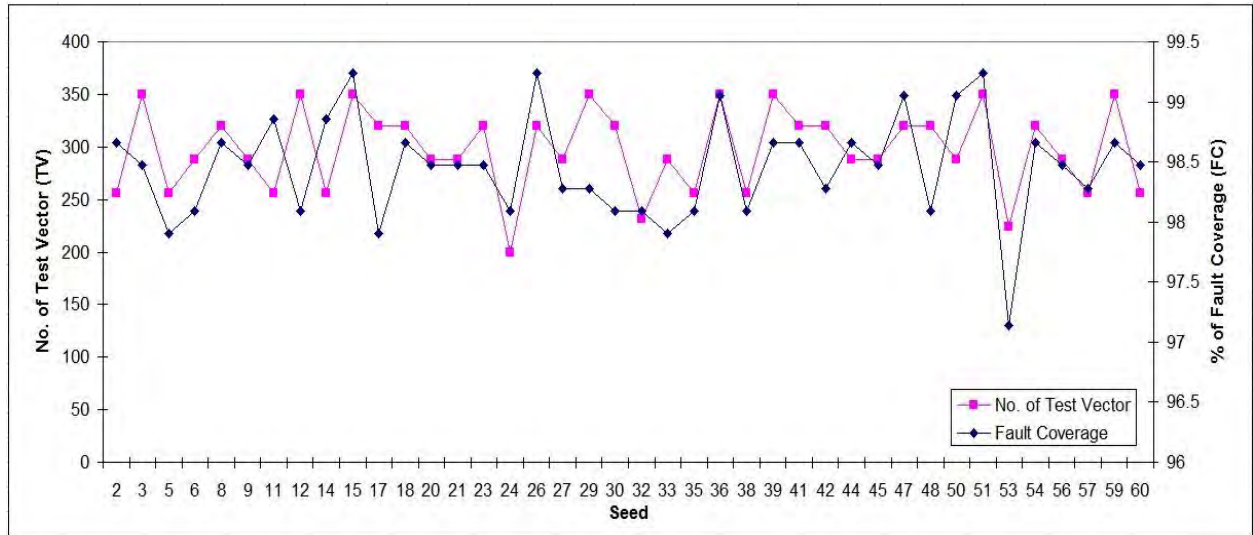


Figure 4.3. Fault simulation result of circuit c432.bench (for feedback polynomial $1+x+x^3+x^4+x^{64}$)

In Figure 4.4, seed „7“ of circuit c499.bench for feedback polynomial $1+x^{60}+x^{61}+x^{63}+x^{64}$ can be considered as the best seed where the fault coverage is 96.70% and the required number of test vectors is 190. The best seed is indicated by the arrow sign.

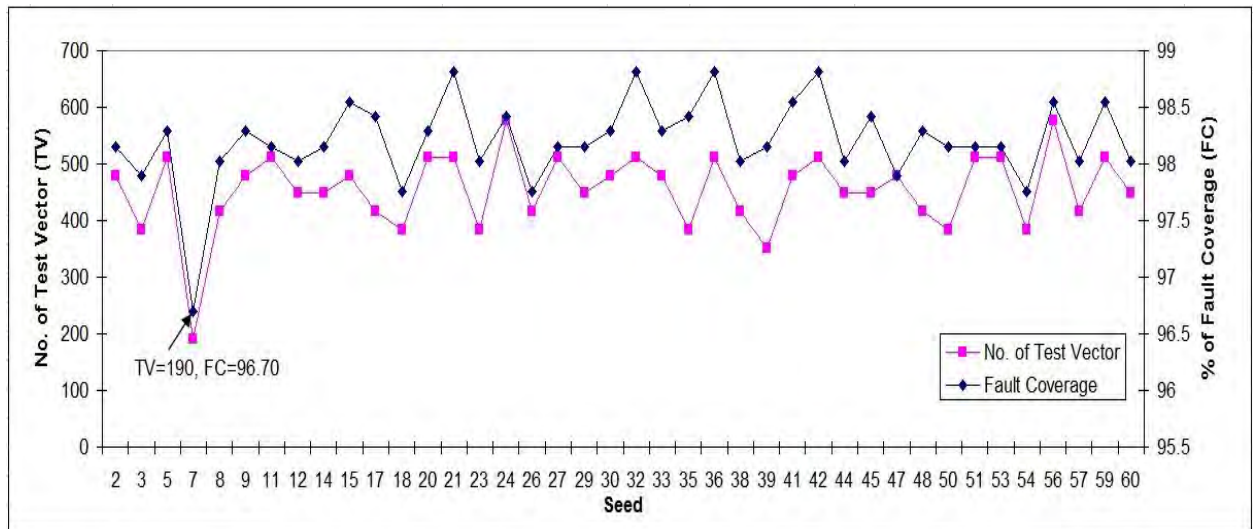


Figure 4.4. Fault simulation result of circuit c499.bench (for feedback polynomial $1+x^{60}+x^{61}+x^{63}+x^{64}$)

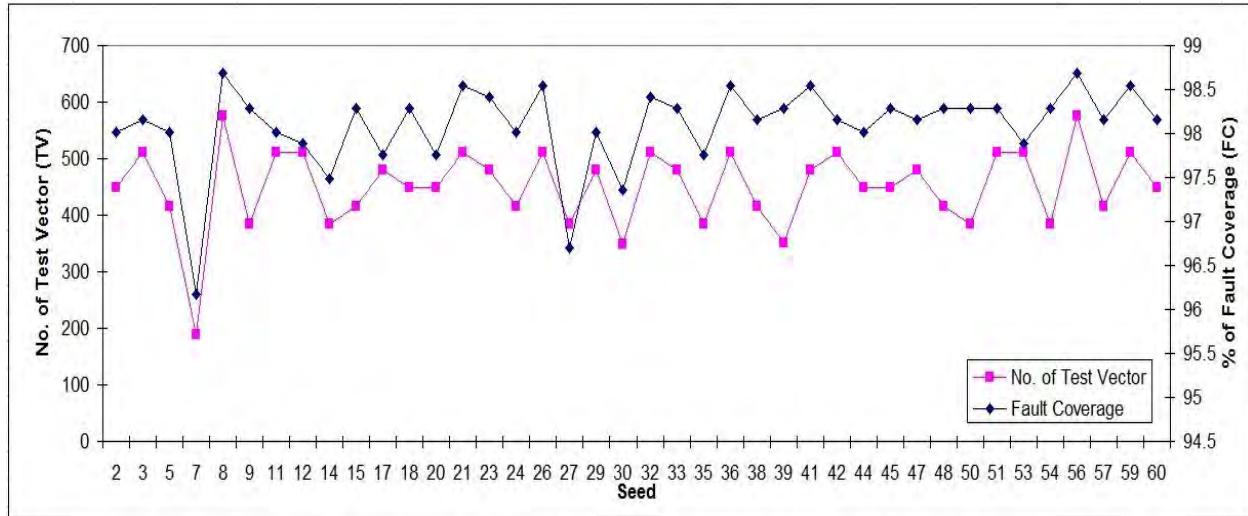


Figure 4.5. Fault simulation result of circuit c499.bench (for feedback polynomial $1+x+x^3+x^4+x^{64}$)

In Figure 4.6, seed „27“ of circuit c880.bench for feedback polynomial $1+x^{60}+x^{61}+x^{63}+x^{64}$ can be considered as the best seed where the fault coverage is 93.52% and the required number of test vectors is 120. The best seed is indicated by the arrow sign.

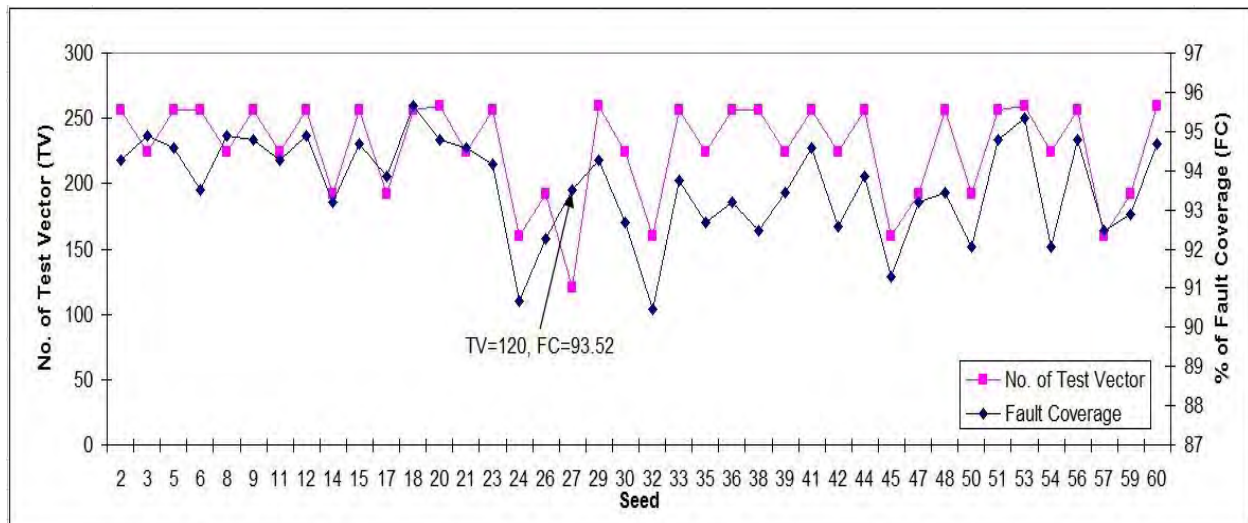


Figure 4.6. Fault simulation result of circuit c880.bench (for feedback polynomial $1+x^{60}+x^{61}+x^{63}+x^{64}$)

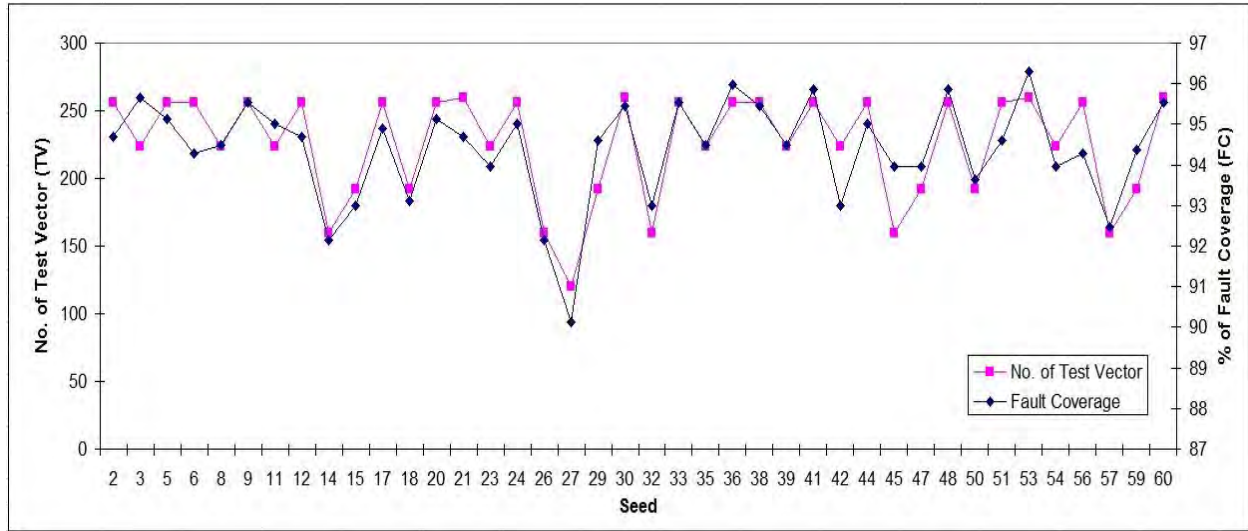


Figure 4.7. Fault simulation result of circuit c880.bench (for feedback polynomial $1+x+x^3+x^4+x^{64}$)

In Figure 4.8, seed „21“ of circuit c1355.bench for feedback polynomial $1+x^{60}+x^{61}+x^{63}+x^{64}$ can be considered as the best seed where the fault coverage is 92.00% and the required number of test vectors is 180. The best seed is indicated by the arrow sign.

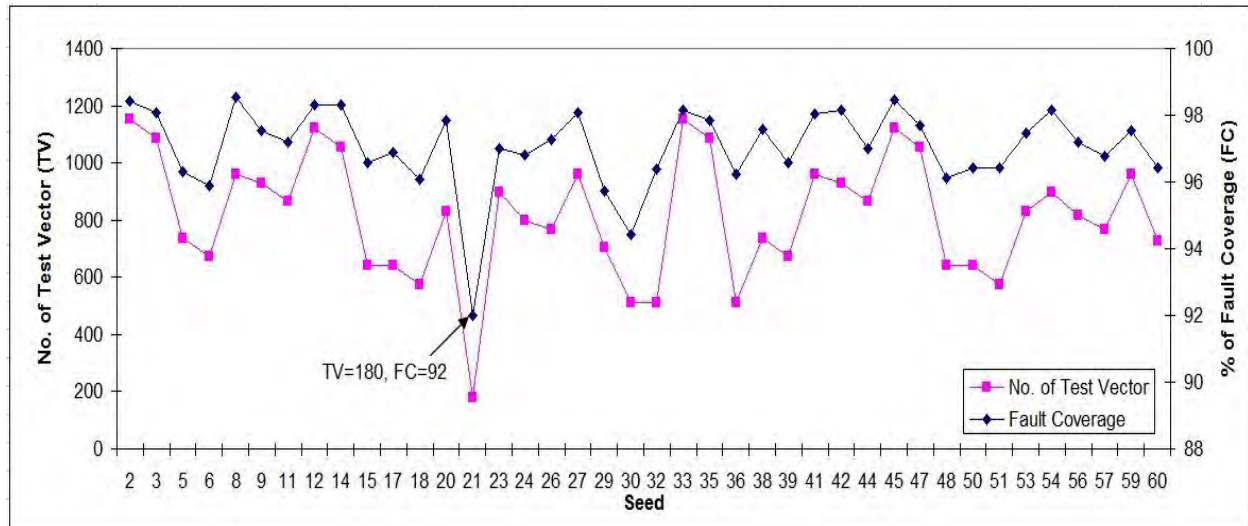


Figure 4.8. Fault simulation result of circuit c1355.bench (for feedback polynomial $1+x^{60}+x^{61}+x^{63}+x^{64}$)

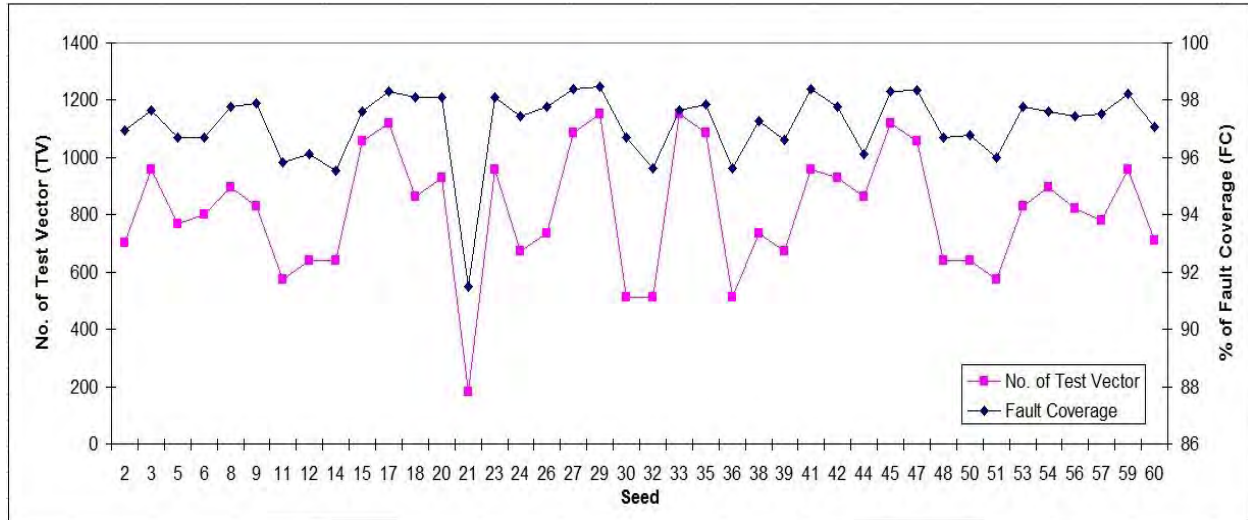


Figure 4.9. Fault simulation result of circuit c1355.bench (for feedback polynomial $1+x+x^3+x^4+x^{64}$)

In Figure 4.10, seed „24“ of circuit c1908.bench for feedback polynomial $1+x^{60}+x^{61}+x^{63}+x^{64}$ can be considered as the best seed where the fault coverage is 96.33% and the required number of test vectors is 880. The best seed is indicated by the arrow sign.

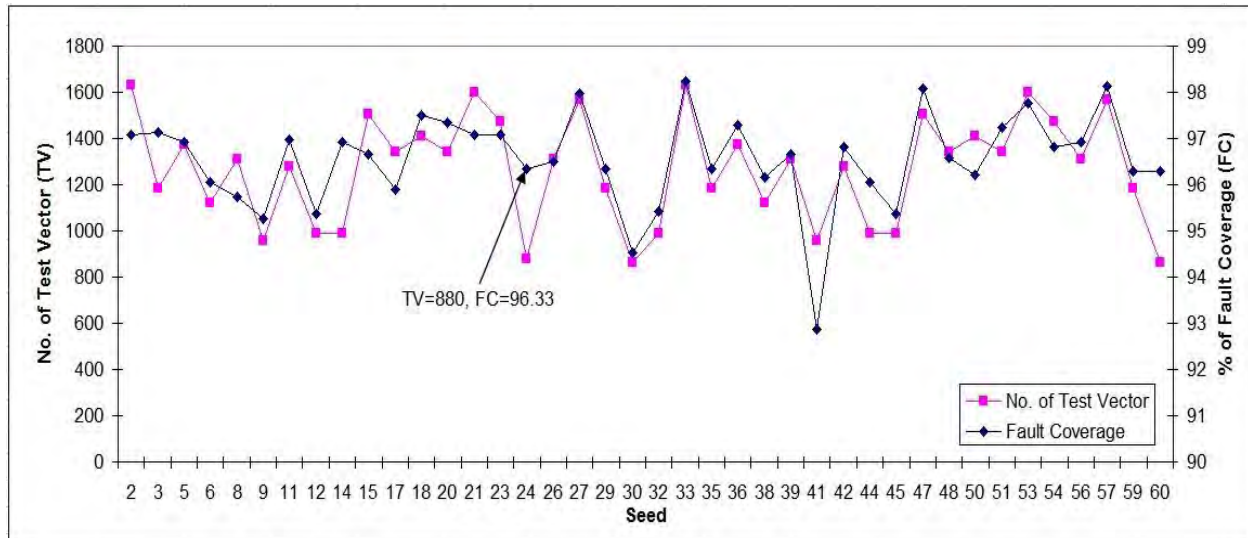


Figure 4.10. Fault simulation result of circuit c1908.bench (for feedback polynomial $1+x^{60}+x^{61}+x^{63}+x^{64}$)

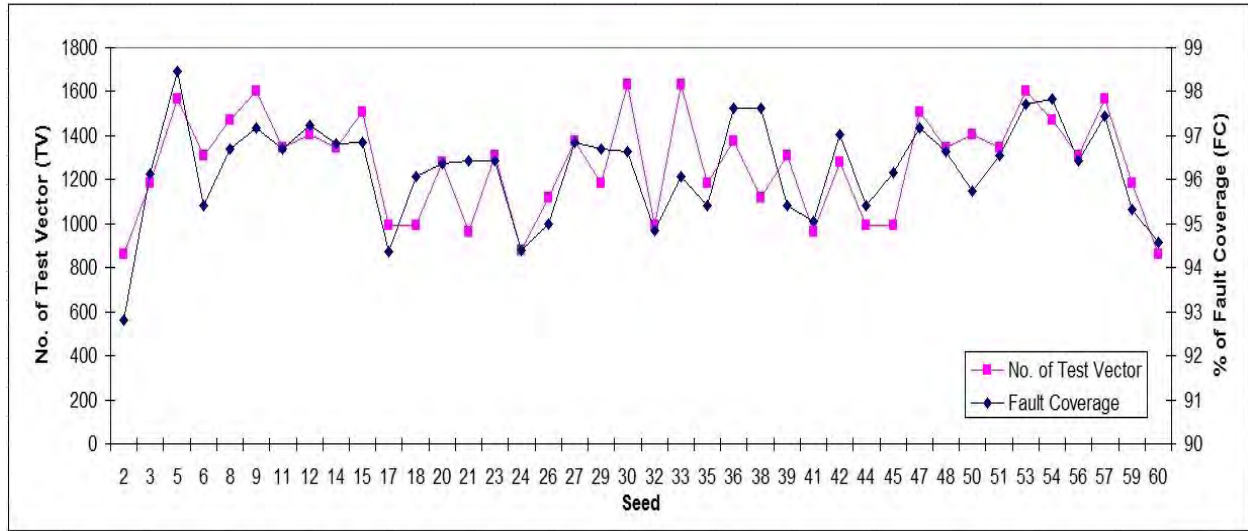


Figure 4.11. Fault simulation result of circuit c1908.bench (for feedback polynomial $1+x+x^3+x^4+x^{64}$)

In Figure 4.12, seed „36“ of circuit c2670.bench for feedback polynomial $1+x^{60}+x^{61}+x^{63}+x^{64}$ can be considered as the best seed where the fault coverage is 83.55% and the required number of test vectors is 250. The best seed is indicated by the arrow sign.

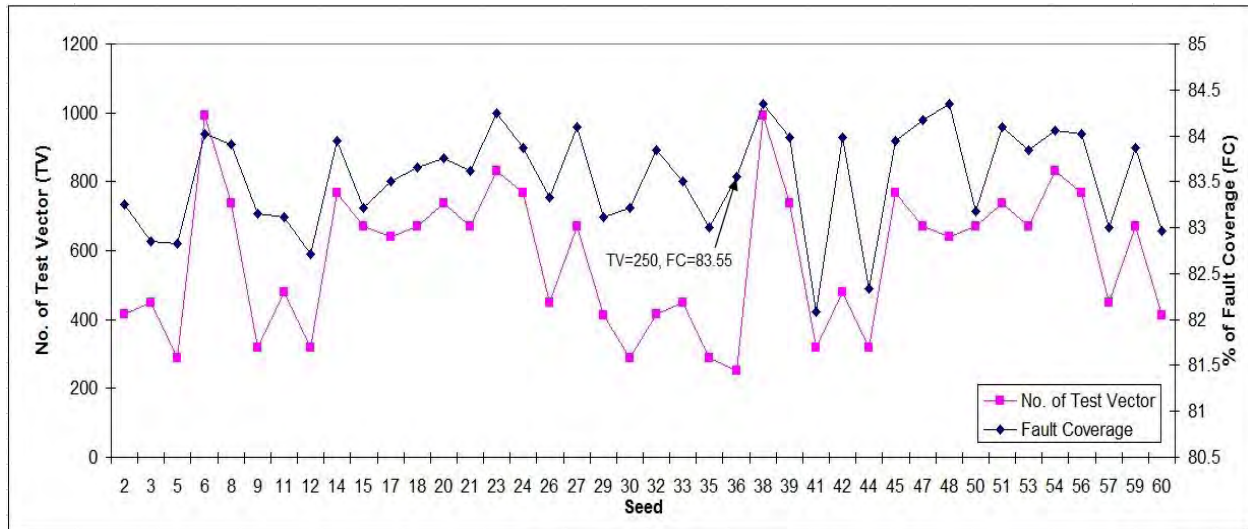


Figure 4.12. Fault simulation result of circuit c2670.bench (for feedback polynomial $1+x^{60}+x^{61}+x^{63}+x^{64}$)

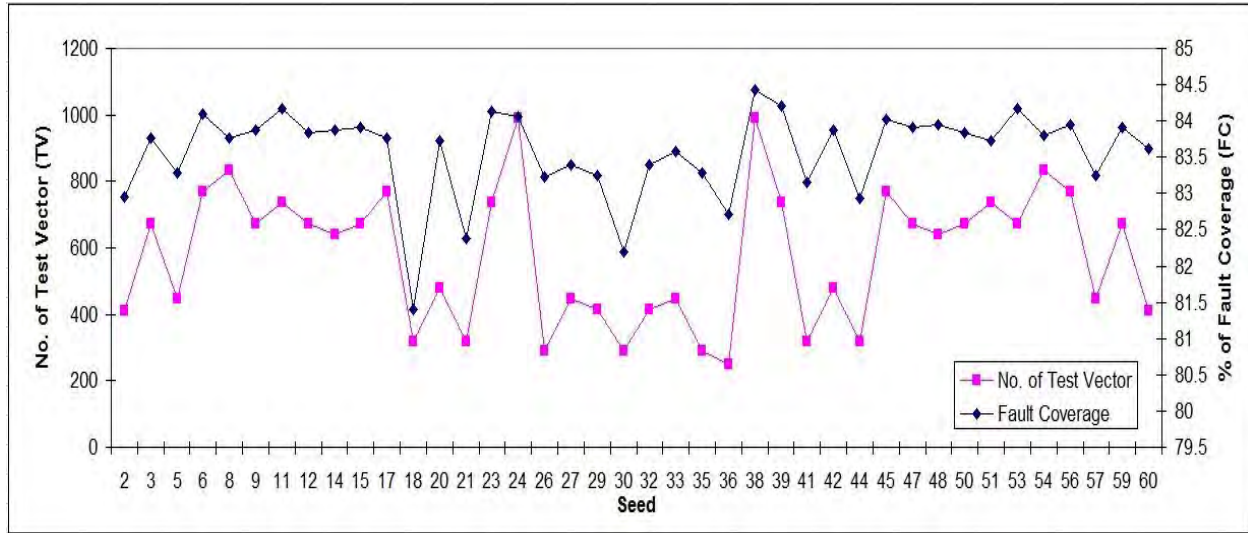


Figure 4.13. Fault simulation result of circuit c2670.bench (for feedback polynomial $1+x+x^3+x^4+x^{64}$)

In Figure 4.14, seed „17“ of circuit c3540.bench for feedback polynomial $1+x+x^3+x^4+x^{64}$ can be considered as the best seed where the fault coverage is 91.80% and the required number of test vectors is 540. The best seed is indicated by the arrow sign.

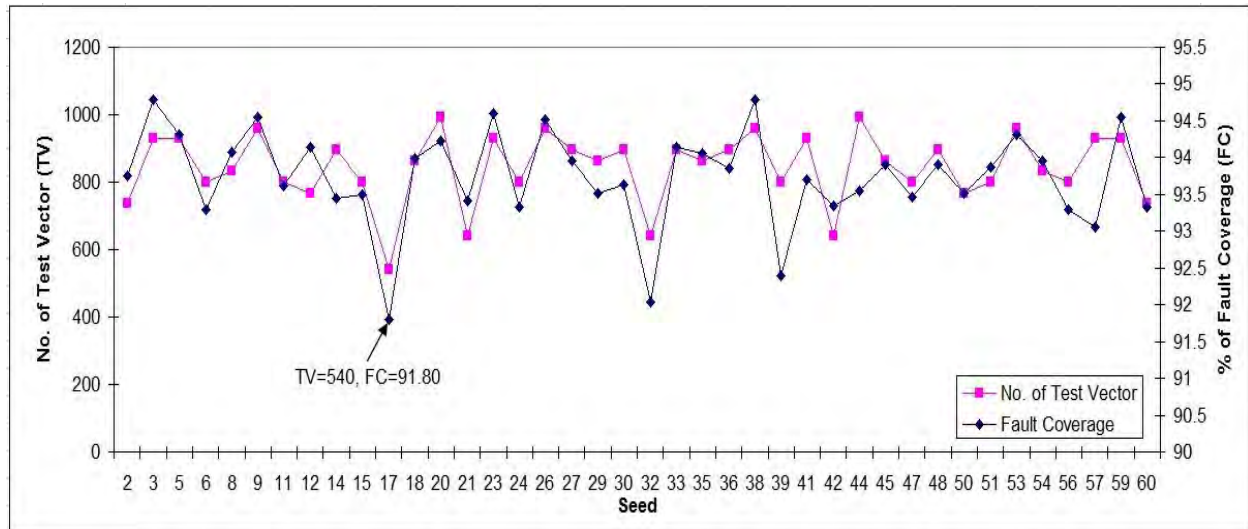


Figure 4.14. Fault simulation result of circuit c3540.bench (for feedback polynomial $1+x+x^3+x^4+x^{64}$)

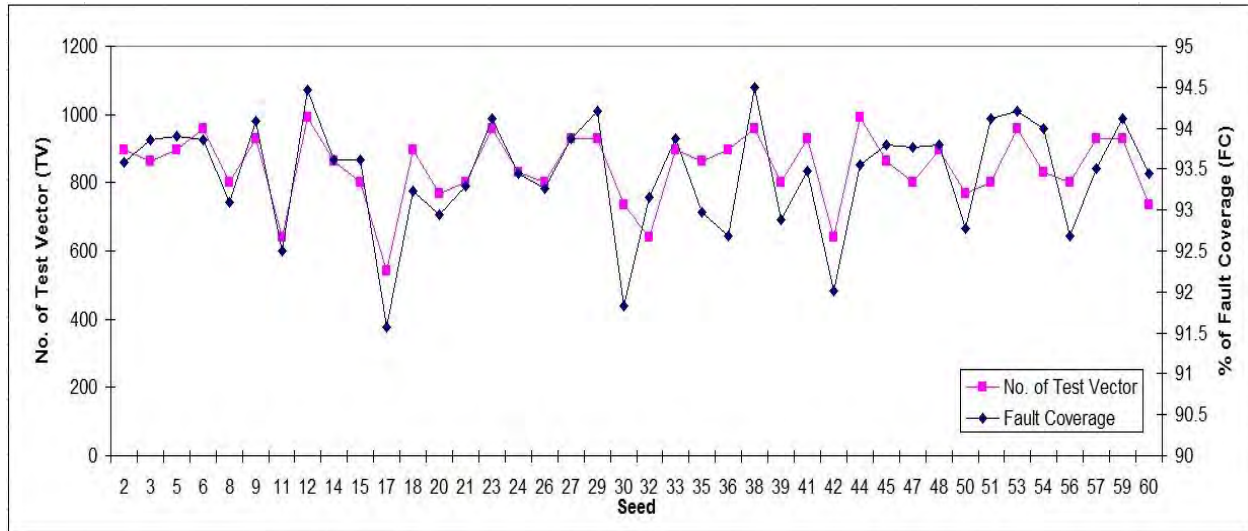


Figure 4.15. Fault simulation result of circuit c3540.bench (for feedback polynomial $1+x^{60}+x^{61}+x^{63}+x^{64}$)

In Figure 4.16, seed „22“ of circuit c5315.bench for feedback polynomial $1+x+x^3+x^4+x^{64}$ can be considered as the best seed where the fault coverage is 98.30% and the required number of test vectors is 560. The best seed is indicated by the arrow sign.

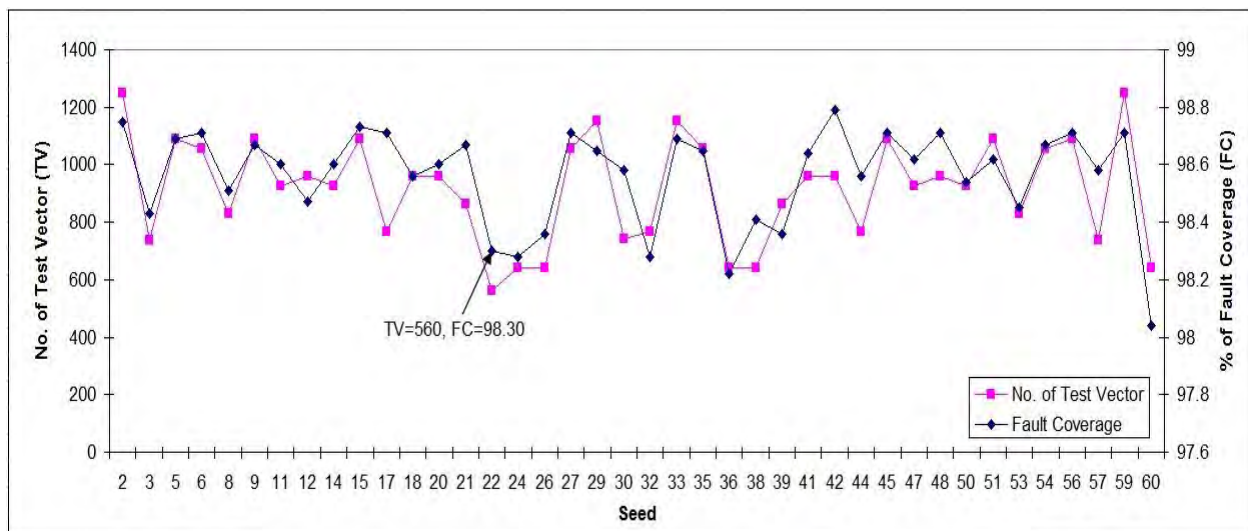


Figure 4.16. Fault simulation result of circuit c5315.bench (for feedback polynomial $1+x+x^3+x^4+x^{64}$)

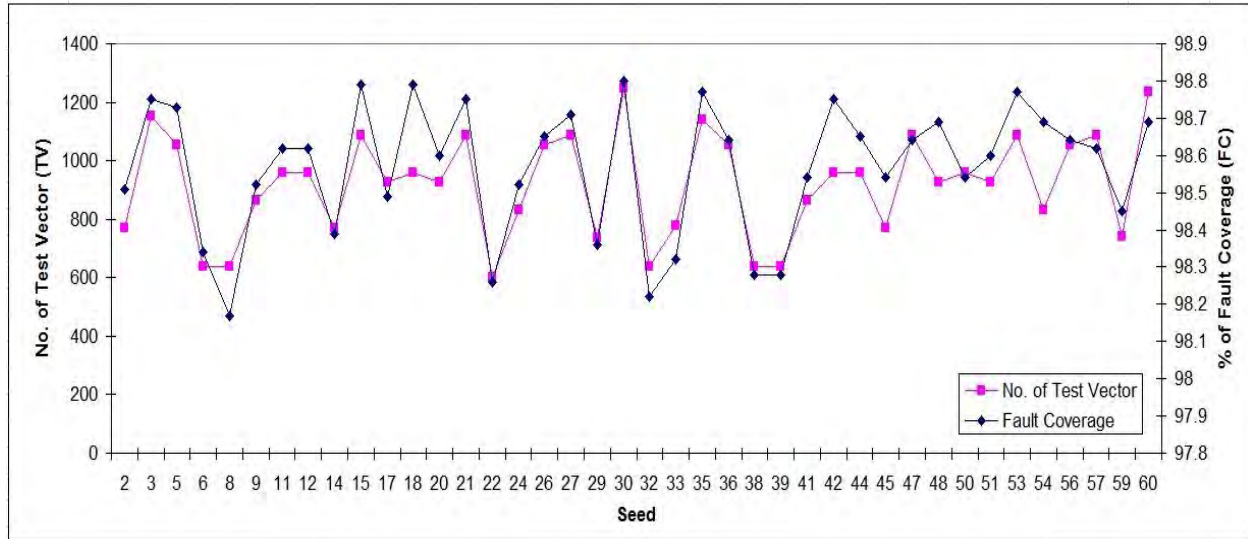


Figure 4.17. Fault simulation result of circuit c5315.bench (for feedback polynomial $1+x^{60}+x^{61}+x^{63}+x^{64}$)

In Figure 4.18, seed „11“ of circuit c6288.bench for feedback polynomial $1+x+x^3+x^4+x^{64}$ can be considered as the best seed where the fault coverage is 99.47% and the required number of test vectors is 60. The best seed is indicated by the arrow sign.

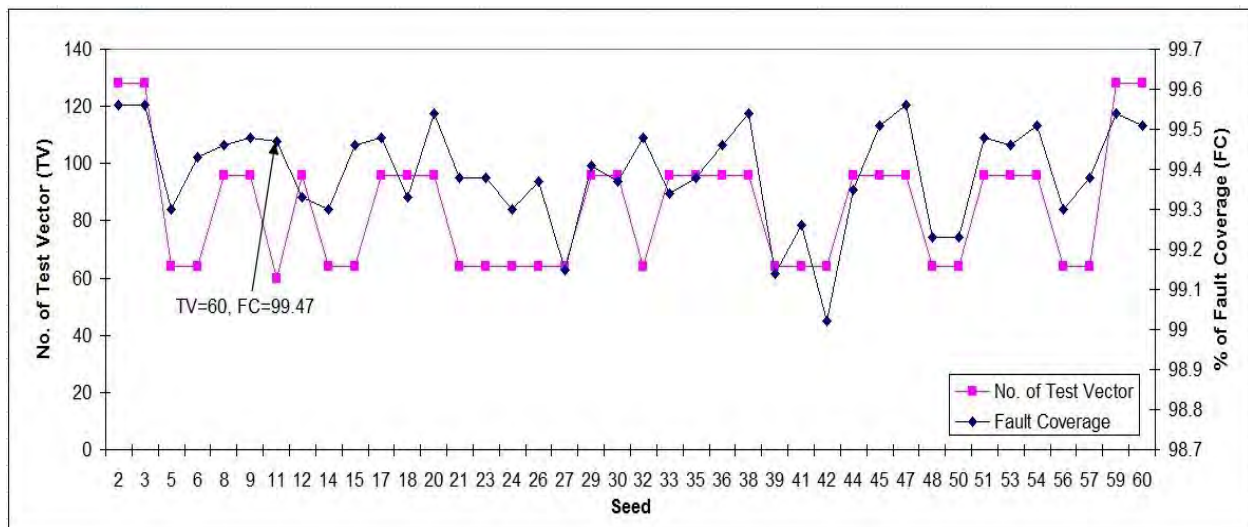


Figure 4.18. Fault simulation result of circuit c6288.bench (for feedback polynomial $1+x+x^3+x^4+x^{64}$)

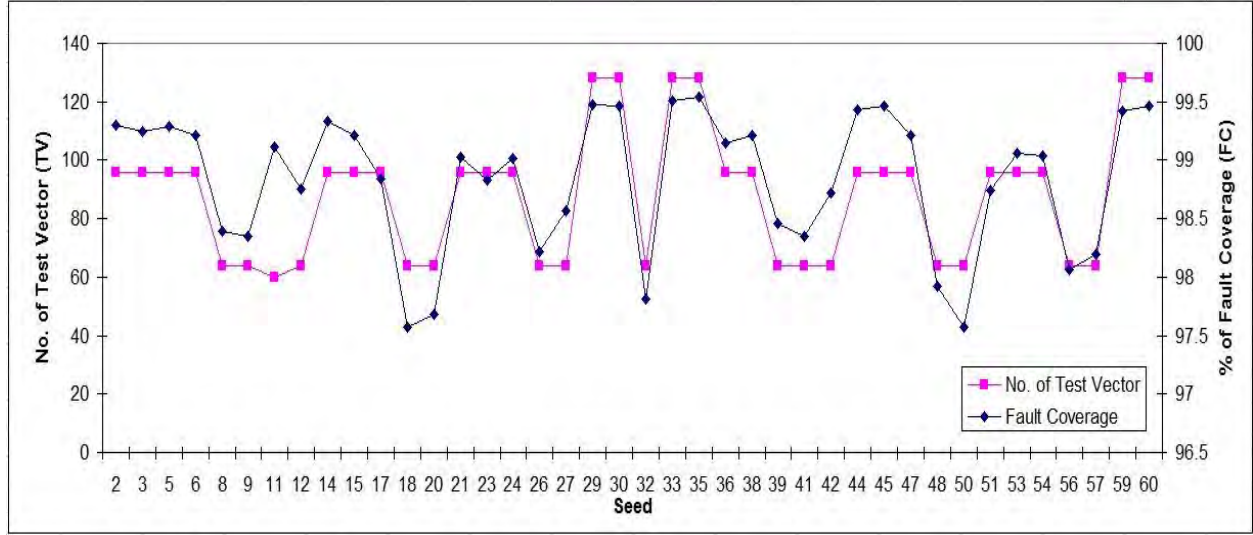


Figure 4.19. Fault simulation result of circuit c6288.bench (for feedback polynomial $1+x^{60}+x^{61}+x^{63}+x^{64}$)

It is also seen that fault detection profile of the PRV sequences for a benchmark circuit is not exactly same as that of other circuit. For example, number of PRVs to achieve 80 to 90 percent fault coverage for the circuit c880.bench, c1355.bench, c1908.bench and c5315.bench are much more than that for the circuits c432.bench, c499.bench and c3540.bench. It is due to the different complexity levels of the circuits as mentioned in Table 3.2. The benchmark circuits are having different number of gate density, primary inputs and outputs, number of lines and nodes and with different number of total faults.

The screenshots of fault simulation result on the ISCAS benchmark circuit c432.bench using the PRV sequences generated by the proposed FLFSR is presented in Appendix A22.

First it shows the number of inputs, outputs, gates and the level of the circuit. The name of the test vector file is output1.test is also shown. The figure shows that the percentage of fault coverage is 98.282 using 200 number of test vectors. Among the 524 faults 515 faults are

detected and the remaining 9 faults were undetected. It also shows the amount of memory used and the total CPU time required. Similarly Appendix A23 and Appendix A24 shows the screenshots of fault simulation result of circuit c499.bench and c1908.bench circuits respectively with the same meanings.

To determine the optimum switching point from the pseudo-random test mode to the deterministic test mode, fault simulation experiments have been carried out on the ISCAS benchmark circuits using the PRV sequences generated based on the best seed of the LFSR as presented in Appendix A4 to A21.

Once the optimal switching moment from the pseudo-random test to the deterministic test mode is determined and the number of PRVs (PRV generated using the best seed) detecting the maximum ETD faults are also determined using the fault simulation technique, the remaining faults (HTD) are targeted using the deterministic test vectors. The deterministic test vectors are generated using the FSIM.

4.3.1 Best Seed Determination

For measuring CV of a PRV sequence for a particular seed for a benchmark circuit, MATLAB R2012b is used. To compare the value of CV of PRV sequences generated for different seeds, we have selected a number of different seeds randomly. We have measured the CV of PRV sequences for different seeds and reported accordingly. The source code of MATLAB program for finding CV is given in Appendix A25. The result is given in Table 4.3.

Table 4.3. CV of PRV sequences for different seeds for different benchmark circuits.

c432	Seed 6	Seed 15	Seed 23	Seed 24	Seed 26	Seed 35	Seed 38	Seed 42	Seed 51	Seed 59
	118.54	115.58	116.51	119.01	118.41	118.30	118.30	115.31	113.86	113.77
c499	Seed 5	Seed 7	Seed 12	Seed 18	Seed 24	Seed 33	Seed 42	Seed 47	Seed 54	Seed 60
	109.51	112.39	111.25	111.70	108.25	110.23	109.35	110.13	111.47	110.88

From Table 4.3 it is observed clearly that the value of CV of the PRV sequence is maximum for the seed 24 for benchmark circuit c432.bench. So, we can say that the PRV sequence generated for seed 24 is more random as compared to other seeds. As a result, seed 24 for circuit c432.bench is determined as the best seed. In similar way, we can say that seed 7 for circuit c499.bench is the best seed. Results of CV for the rest of the benchmark circuits follow the similar profile. The screenshot of the output of the MATLAB program for finding CV is given in Appendix A26.

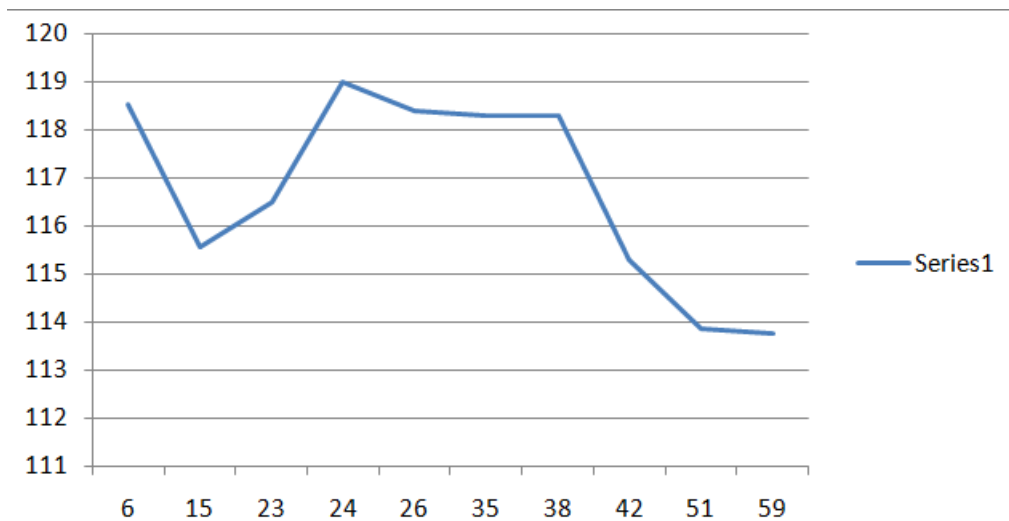


Figure 4.20. CV for different seeds for circuit c432.bench

Figures 4.20 graphically shows the CV of different seeds as indicated in the X-axis. It is clear from the figure that seed 24 has the peak value.

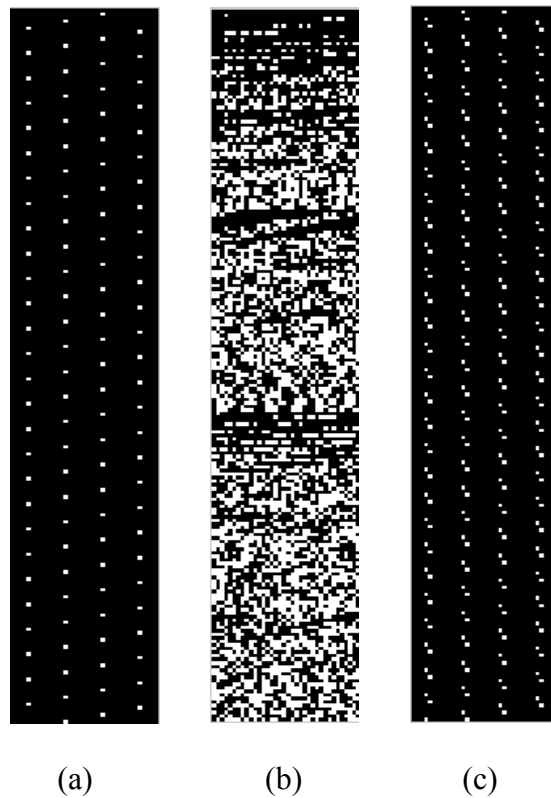


Figure 4.21. Image representation of PRV sequences

This section demonstrates the effectiveness of the proposed FLFSR in terms of fault coverage. Figure 4.21 (a) and (c) shows the image representation of the PRV sequence generated from a general LFSR and 4.21(b) shows the same generated from our proposed 64-bit FLFSR. The images are created from a test vector file of 7200 bits using MATLAB. It's quite clear that image (b) is more random than image (a) or (c). This may provide one possible explanation as to why the proposed FLFSR admit significantly higher fault coverage.

4.3.2 Comparison

Summary of the fault simulation results of the ISCAS benchmark circuits using the proposed 64-bit FLFSR is presented in Table 4.4.

Table 4.4: Summary of fault simulation results of the ISCAS benchmark circuits with using proposed technique

ISCAS Benchmark Circuits	Total Number of Faults Inserted	Number of Test Vectors		Total Number of Test Vectors	% Fault Coverage
		Random	Deterministic		
c432	802	200	9	209	100
c499	1306	190	25	215	100
c880	1428	120	61	181	100
c1355	1970	180	126	306	100
c1908	1282	880	69	949	100
c2670	2588	250	452	702	100
c3540	2988	540	281	821	100
c5315	5640	560	91	651	100
c6288	9804	60	41	101	100

Table 4.4 shows the total number of test vectors required to achieve the complete fault coverage for the ISCAS benchmark circuits using the FLFSR. It shows that 100% fault coverage has been achieved for all the benchmark circuits. The results presented in Table 4.5 can be compared with that of other researchers [2-8]. Comparison of the fault simulation results is presented in Table 4.5.

Table 4.5: Comparison of fault simulation results of the ISCAS benchmark circuits with that of other researchers

ISCAS Benchmark Circuits	*NTV1	*NTV2	*NTV3	*NTV4	*NTV5	*NTV6	*NTV7
c432	209	214	224	320	512	1024	320
c499	215	225	512	-	-	-	-
c880	181	248	160	416	260	1280	160
c1355	306	314	512	1664	2244	2098	2784
c1908	949	969	992	2496	2308	5376	3916
c2670	702	724	288	6240	10766	5888	6400
c3540	821	271	640	9504	12220	3840	4352
c5315	651	388	640	1950	1316	2048	1024
c6288	101	234	64	-	-	-	-

*NTV1 = Number of test vectors required using FLFSR based mixed-mode technique in the present work

*NTV2 = Number of test vectors required using LFSR based mixed-mode technique [2]

*NTV3 = Number of test vectors required using LFSR based mixed-mode technique [3]

*NTV4 = Number of test vectors using weighted random technique [5]

*NTV5 = Number of test vectors using weighted random technique [6]

*NTV6 = Number of test vectors using weighted random technique [7]

*NTV7 = Number of test vectors using weighted random technique [8]

The fault simulation results of the benchmark circuits „c499“ and „c6288“ from other researchers are not available. The „–“ sign in Table 4.5 is to indicate the unavailability of the actual data. It shows that the proposed 64-bit Fibonacci test pattern generator in mixed mode approach is capable of producing 100% fault coverage using lowest number of test vectors than that of all other researchers.

4.4 Summary

Fault Simulation results of the ISCAS benchmark circuits presented in this chapter verify the effectiveness of the proposed approach in IC testing. The results obtained from the fault simulation experiments on the different ISCAS benchmark circuits show that the proposed 64-bit FLFSR produces 100% fault coverage for the benchmark circuits using much lower number of test vectors than that of other researchers.

CHAPTER 5

CONCLUSION AND FUTURE WORK

5.1 Conclusion

Test pattern generator is an important module for any IC tester. Performance of the tester in terms of testing time, test economics etc largely depends on it. This project evaluates a 64-bit FLFSR for its effectiveness in VLSI testing. Fault simulation experiments have been conducted on a number of ISACS benchmark circuits for this purpose. The fault simulation results show that complete fault coverage can be achieved using lower number of test vectors than that of other researchers. Moreover best seed and optimum switching point have also been examined by conducting fault simulation experiments on ISCAS benchmark circuits. Determination of best seed has again been verified by calculating the coefficient of variation of the random sequences that have applied to the benchmark circuits. Based on the result in the project initiative can be taken for designing low cost IC Tester.

5.2 Future Work

For this project we recommend the following future works.

- i) Fault simulation experiments can be conducted on the other types of benchmark circuits and can be tested the effectiveness of the proposed 64-bit FLFSR.
- ii) The test results in this project can be used in designing low cost IC tester.

REFERENCES

- [1] Stanely, L. H., "VLSI testing (digital and mixed analogue/digital techniques) ", IEEE, London, U. K., 1998.
- [2] Kabir, A., Ali, L., "Design of GLFSR based test processor chip" Proceedings of 2009 Student Conference on Research and Development (SCORED 2009), UPM Serdang, Malaysia, 16-18 Nov. 2009.
- [3] Ali, L., Roslina, S., Ishak, A., Alauddin, M. A, Bambang, S. S., "Challenge and directions for IC testing", Integration, the VLSI Journal, pp. 17-28, vol 37(1), Elsevier Science, Netherland, Feb. 2004.
- [4] Ali, L., "Design of a test processor chip using multiple polynomial, multiple seed linear feedback shift register", M.Sc Thesis, Universiti Kebangsaan Malaysia, 1998.
- [5] Iftekhhar, A., "VLSI circuit testing using probabilistic approach", Ph.D Thesis, Universiti Kebangsaan Malaysia, 1995.
- [6] Wunderlich, H. J., "**Multiple distributions for biased random test patterns**", IEEE Trans. on Comp.-Aided Design, vol. 9 (6), pp. 584-593, 1990.
- [7] Waicukauski, J. A., Lindbloom, E., Eicheblberger, E. B., and Forlenza, O. P., "A method for generating weighted random test patterns", IBM Journal of research and development, vol. 33(2): 149-161, 1989.
- [8] Lisanke, R., Braglez, F., Degeus, A. J., and Gregory, D., "Testability-driven random test-pattern generation", IEEE Trans. Comp.-Aided Design, vol. 6(6), pp. 1082-1087, 1987.
- [9] Niraj, K. J., and Sandip, K., "Testing and reliable design of CMOS circuits", Kluwer academic publishers, USA, 1990.

- [10] Hamzaoglu, I., and Patel, J. H., “New techniques for deterministic test pattern generation”, 6th IEEE Proc. of VLSI Test Symposium, pp. 446-452, 1998.
- [11] David, R., “Random testing of digital circuits: Theory and Applications”, Marcel Dekker Inc., New York, 1998.
- [12] Koenemann, B., “LFSR-coded test patterns for scan designs”, Proc. of European Test Conf., Germany, pp. 237-242, 1991.
- [13] Venkataraman, S., Rajski, J., Hellebrand, S., and Tarnick, S., “An efficient bist scheme based on reseeding of multiple polynomial linear feedback shift registers”, IEEE/ACM Int. Conf. on Comp.-Aided Design, pp. 572-577, 1993.
- [14] Hellebrand, S., Tarnick, S., Rajski, J., and Courtois, B., “Generation of vector patterns through reseeding of multiple-polynomial linear feedback shift registers”, IEEE Proc. of Int. Test Conf., pp. 120-129, 1992.
- [15] Rajski, J., Tyszer, J., and Zacharia, N., “Test data decompression for multiple scan designs with boundary scan”, IEEE Trans. on Comp., vol. 47(11), pp. 1188-1200, 1998.
- [16] Krishna, C. V., Jas, A., and Touba, N. A., “Test vector encoding using partial LFSR reseeding”, Proc. of Int. Test Conf., pp. 885 –893, 2001.
- [17] Rajsuman, R., “IDDQ tesing for CMOS VLSI”, Artech House, Boston, 1995.
- [18] Fujiwara, H., “Logic testing and design for testability”, Cambridge:MIT Press, 1985.
- [19] Dufaza, C., “Theoretical properties of LFSRs for built-in self test”, Integration, the VLSI Journal, vol. 25(1), pp. 17-35, 1998.
- [20] Bardell, P. H., McAnney, W. H., and Savir, J., “Built-in test for VLSI: Pseudo-random techniques”, John Wiley and Sons, New York, USA, 1987.
- [21] Gupta, S. P., and Gupta, M. P., “Business Statistics”, New Delhi, 2006.

- [22] Brglez, F., and Fuziwara, H., “A neural netlist of 10 combinational bench-mark circuits and target translator in FORTRAN”, Special session on ATPG and fault simulation, Int. Symp. on Circuits and Systems, Kyoto, Japan, 1985.
- [23] Brglez, F., Bryan, D., and Kozminski, K., “Combinational profiles of sequential benchmark circuits”, Proc. of ISCAS, pp. 1929-1934, 1989.
- [24] Anon., <http://www.fm.vslib.cz/~kes/asic/iscas>, 2004b.
- [25] Lee, H. K., and Ha, D. S., “An efficient, forward fault simulation algorithm based on the parallel pattern fault propagation”, Int. Test Conference, pp. 946-955, Nashville, TN, 1991.
- [26] Eichelberger, E. B., and Lindbloom, E., “Random-pattern coverage enhancement and diagnosis for LSSD logic self-test”, IBM Journal of Research and Development, vol. 27(3), pp. 265-272, 1983.
- [27] Gloster, C. S. Jr., and Brglez, F., “Boundary scan with cellular automata-based built-in self-test”, Int. Test Conf., pp. 138-145, 1988.
- [28] Brglez, F., Gloster, C., and Kedem, G., “Hardware-based weighted random pattern generation for boundary scan”, Int. Test Conf. pp. 264-273, 1989.
- [29] Ali, L., Roslina, S., Ishak, A., Alauddin, M. A, Bambang, S.S., “Maximization of fault detection in IC testing”, Proceedings of Int. Conf. on semiconductor Electronics, pp. 557-560, Malaysia, 2002.

APPENDIX A1

COMPUTER PROGRAM FOR GENERATING RANDOM NUMBERS

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<ctype.h>
main()
{
    clrscr();
    FILE *fpt;
    char fname[10];
    int vector[70],i,j,tap1,tap2,tap3,tap4,stage,k,temp1,temp2,temp3,temp4,temp5,temp6,clk,
    width,tap[10],n,u,v;
    int pos;
    printf("\n \n No. of Stages of FLFSR =");
    scanf("%d",&stage);
    printf("\n \n Test vector Width =");
    scanf("%d",&width);
    printf("Enter Tap1=");
    scanf("%d",&tap1);
    printf("Enter Tap2=");
    scanf("%d",&tap2);
    printf("Enter Tap3=");
    scanf("%d",&tap3);
    printf("Enter Tap4=");
    scanf("%d",&tap4);
    printf("Enter the seed position=");
    scanf("%d",&pos);
    printf("How many test vectors=");
    scanf("%d",&clk);
    printf("Enter the name of the file=");
    scanf("%s",&fname);
    fpt=fopen(fname,"w");
    for(i=0;i<= stage -1;i++)
    {
        if(i==(pos-1))
            vector[i]=1;
        else
            vector[i]=0;
    }
    for(k=1;k<=clk;k++)
    {
        fprintf(fpt,"%d:",k);
```

```

    for(j=1;j<=width;j++)
    {
        temp1=vector[tap1-1];
        temp2=vector[tap2-1];
        temp3=vector[tap3-1];
        temp4=vector[tap4-1];
        for(i=0;i<stage-1;i++)
            vector[stage-i-1]=vector[stage-i-2];
        vector[0]=temp1^temp2^temp3^temp4;
        fprintf(fpt,"%d",vector[stage-1]);
    }
    fprintf(fpt,"\n");
}
getch();
}

```


APPENDIX A2

COMMAND FOR FAULT SIMULATION USING FSIM

Format: fsim [options] circuit_file [> outfile]

OPTIONS: Several options as listed below are available for

atalanta. If an option is not specified, the default value is used.

-r n Test patterns are generated internally using the random number generator (random()). The simulation stops when either n patterns are applied or all faults are detected.

(default: -r 224)

-s n Initial seed for the random number generator (random()).

If n=0, the initial seed is the current time.

(default: -s 0)

-t filename Test pattern file.

Test patterns are read from the file.

The simulation stops when either all test patterns are applied or all faults are detected.

(default: random patterns are used)

-l filename Log file is created

(default: no logfile is created)

-f filename The options are read from the named file

For example for the following command

```
fsim -t output.test c432.isc
```

--- The test patterns are read from the file " output.test".

The simulation stops when all test patterns in `output.test` are simulated or all faults are detected.

Where `output.test` is the test vector file.

For example test pattern file for the circuit c432.bench

1:00

2:00010000000000000000000000000000000000

[illegible]

4:00

5:000000000000000010100010100000000000

The test pattern begins after colon (:). After n bits, where n is the number of primary inputs of the circuit, all the following characters are ignored until the next colon (:) is read. The j'th bit of a test pattern is the value to be applied to the j'th input of the circuit. For example, c432 has 36 inputs named input1, input2, input3, input6 ... input36 which appear in the order in the netlist. The first bit of a test pattern is the value for input1, the second for input2, ..., and the last bit for input36.

APPENDIX A3

RANDOM TEST VECTORS FOR ISCAS85 BENCHMARK CIRCUIT C432.BENCH

RANDOM TEST VECTORS FOR ISCAS85 BENCHMARK CIRCUIT C432.BENCH

0:00
1:0001000000000000000000000000000000000000
2:00
3:00
4:00
5:00
6:00
7:0001110110110111000000000000000000000000
8:00
9:0001000100000000000000000000000000000000
10:00
11:00
12:0001010100010100010100010101000000000000
13:00
14:1101101101101100011100000000000000000000
15:00
16:0001000000010000000000000000000000000000
17:0001101100011011000000000000000000000000
18:1011000000000000000000000000000000000000
19:0100010100000001010001000100010100000000
20:00
21:0001110110101010101011011100000000000000
22:0001000100010000000000000000000000000000
23:0000000010001000100000000000000000000000
24:1011000110101011000110101011000110101010
25:10101011000000010101000000010100010101010
26:00010100010100010100010100000001010101010
27:000111000111000111011011011011011011011
28:01101101101101110001110001100000000000000
29:0000000010000000000000000000000000000000
30:0001000000000000000000000000000000000000
31:00
32:0001010111100000000101000101000000000000
33:00
34:0010000111011011011100000000000000000000
35:00011101101101100000110110100111000110001
36:0000000010001000000000000000000000000000
37:0000101010100001101000001011000110101010
38:1011000000000000000000000000000000000000
39:1111010111101111010001010001010100000000
40:000101010001010110000010100111101111101111
41:10111111101101101100011100011100011000110
42:110010101011101010111011011011011000110

43:000000010000000000000001000110100001
44:101100011010000000010001101000011011
45:0000000000001101001011111010001000101
46:111000011010010111110100010100000001
47:010110011110111110101011111100110101
48:100111101111101101110001110010111011
49:101101110001110010111010101110111011
50:011000010000000010111010101000010000
51:000010111011000110101010000010111011
52:000011111011000011111011000011111010
53:010001010000111011111010010110010110
54:010110010110010110010111100010110111
55:100010110111100110110110110110110110
56:110110110111110001110001110001110001
57:110111000000000000000000000000000001
58:10100000000000000000000000000000101000100
59:00000000000000000000000000001010111100000
60:000000000000000111011010110000000000
61:000000000001110011110010000000000000
62:000100010001010101000000000000000001
63:000011011010011000000000000110101010
64:110000011100000000000001101110100001
65:101110100000000101010000010101010000
66:010000000001010010111111010010111110
67:000111000111011100000111011011000001
68:110100111100111100111101001100000000
69:010100010100010000010101000111010110
70:110110110111110111010000011101101101
71:101011011100011000011010000000000001
72:101101011111010001000000000101000100
73:000110110101111000000001010000101110
74:111110101100000111011010110101000010
75:111100100001110111111110101101110101
76:010100010001010010011111111110100111
77:000101100001110100010110000001101010
78:110100001010000001111010000011101011
79:000111101110101001010100010011111110
80:111001001111111010001101010000111010
81:110010010001101001010001101001000101
82:000111110010100111100011010110000010
83:010110010110010110001011011000101010
84:101010110000101010101011000110110110
85:110110100111000010011000000000010101
86:111000000001010001000000000000011010
87:000111001110100000011100111100100001
88:110110101100000000010101111100001100

89:111110010000110110100111000100010101
90:010000011100111010011011110101010011
91:101000011010000010101100000111010000
92:110011101100111110000101001111110101
93:111011100101010100011111101111001001
94:110101001111010101001110111110100101
95:000001100010011011101000100110010101
96:111000010100101101111001011101011001
97:001100101111010011110100111100111101
98:001100011101011101101011000111101111
99:111101011111010110110111110111000001
100:101101000101010000111011000011101110
101:111010000001101101000101010000111011
102:000111100010010110001010101011111001
103:010000111011000111100010010000110001
104:000110100110000001110101010111100010
105:010000110001000011101100101001011011
106:101001000110000011110001000011101100
107:101110001001101010011000101110001001
108:101010011000101110001001101110000100
109:110100101110011110000100110100101110
110:011110000100110010001110011011001110
111:011011001110011011001110011011001110
112:011110010000011100011100011100011100
113:011100011100011100011100011011010011
114:010000000000000000000000000000000000
115:0000000000000000100001101101111000000
116:000000000000000000000000000000000000
117:0001101110100000111001000000000000000
118:00000000000000000000000000000000101001011
119:111010000100110000000000000000000000
120:0000000000000000111010011110111111110
121:011101000000000000000000000000000000
122:000100011101011101100001011001111100
123:00000000000000000000000000000000110100001
124:1011010010111110111010100100000000000
125:0000000000000000101011111010000110011
126:110110101100100011000000000000000000
127:000111001110111111101111011100010101
128:10010011010000000000000000000000100001100
129:101100011011110100001100101100011011
130:1100000000000000110111011101101000100
131:111111111011101101000100111001000000
132:000101001010101000111010010100000110
133:101000111010010001001100000111010010
134:000011000011100101110101010011000011

135:100010100111010100011100011010110110
136:000101110110000101110110000001101010
137:01100110000000010101000010111110100
138:10111111010010100101010010111111010
139:000111000111111111001111001111001111
140:001010010001001111000111111100000000
141:100000101101101101101101101010100010
142:100101100000100010010000110110111100
143:0000000000000000100001101101001101010
144:110101000011101000001110010000000000
145:000110111010000110110100010010011110
146:001111101000010011000000000101001011
147:111101000011101000001011000001011111
148:111001110100000111010011110011111110
149:001111101111010101111110000101100111
150:110100011101011011010001000001011011
151:111000111001001111101110101111100001
152:101000001110101101111000110100000001
153:000101011010110111010011010111101000
154:110100011100001011110001101011001001
155:010001011101101011111111001011100000
156:0111111111000010101011000010110111101
157:000101110000101011100010010010000100
158:111100001010111110001111111011110001
159:111001100001000000011110010110011110
160:011101000001000110111000001101111011
161:101100010011010110111011011001111101
162:101001001000101010011110101001001001
163:101010001010000011101011000110000001
164:011000101011110010000000110100110110
165:111010001101010001101001111010011001
166:111010011000101011011000001011110010
167:100110010110101111101111101111101110
168:011001000010101111111010101011110110
169:010111011011011011011010011111001111
170:100111000111000001111100110111010000
171:000000000001101010101101010110000000
172:000101001010111001011111000000000001
173:010100000100100010101000000111010010
174:011001011110100100000001110001110110
175:000101100011100100011100001111011111
176:111000110001000000000100101111101000
177:000100100000011001110110000100001100
178:101100000110001111011111100110000110
179:010111100100101110111011101101010101
180:100001110110010111101101110111110100

181:001110101010101000100000101011000100
182:110111111000010101101111111000110000
183:000011010110111001010010011001100100
184:111100100011000100001101000010101010
185:001001010100001111111100010110100100
186:110010111010111111100000110100010001
187:111001000010001110011000011110111011
188:011100010010101011101010001101001111
189:010000011110110011101010000100001000
190:100001101100110010110101111111010011
191:100111001100111110111101010111010001
192:111110110010111000111101000110001111
193:110101101111100011011110001001100110
194:111000000111111001100001001110100011
195:010000100111000100111111001000100100
196:100101111011100100111100101111110010
197:000010010100101001010000000001111110
198:100100010110101111001010011011000101
199:001010010111000001001001111000101110
200:010111101010101100010011010010100111

APPENDIX A4
 FAULT SIMULATION RESULT FOR BENCH MARK CIRCUIT C432.BENCH
 (FOR FEEDBACK POLYNOMIAL $1+X^{60}+X^{61}+X^{63}+X^{64}$)

Seed	No. of Test Vectors	Fault Coverage (%)	CPU Time (sec)
2	288	97.71	0.500
3	256	97.14	0.500
5	350	98.86	0.250
6	256	97.52	0.517
8	350	99.05	0.250
9	320	98.66	0.250
11	320	97.52	0.250
12	288	98.66	0.517
14	288	98.09	0.517
15	320	97.14	0.500
17	320	98.66	0.517
18	288	97.33	0.250
20	350	98.66	0.250
21	320	98.47	0.500
23	288	97.71	0.250
24	200	98.28	0.500
26	256	96.95	0.250
27	350	98.66	0.250
29	256	98.09	0.250
30	320	98.09	0.500
32	232	97.90	0.250
33	288	98.86	0.500
35	256	97.14	0.500
36	350	99.05	0.250
38	256	96.95	0.250
39	350	97.90	0.250
41	320	98.66	0.500
42	320	98.66	0.250
44	288	99.05	0.500
45	288	98.66	0.500
47	320	97.52	0.250
48	320	98.66	0.250
50	288	98.09	0.250
51	350	97.33	0.250
53	224	97.71	0.250
54	320	97.90	0.500
56	288	98.66	0.500
57	256	98.28	0.250
59	350	98.09	0.500
60	256	98.86	0.500

APPENDIX A5
 FAULT SIMULATION RESULT FOR BENCH MARK CIRCUIT C432.BENCH
 (FOR FEEDBACK POLYNOMIAL $1+X+X^3+X^4+X^{64}$)

Seed	No. of Test Vectors	Fault Coverage (%)	CPU Time (sec)
2	256	98.66	0.500
3	350	98.47	0.250
5	256	97.90	0.250
6	288	98.09	0.500
8	320	98.66	0.500
9	288	98.47	0.250
11	256	98.86	0.500
12	350	98.09	0.250
14	256	98.86	0.250
15	350	99.24	0.500
17	320	97.90	0.500
18	320	98.66	0.250
20	288	98.47	0.517
21	288	98.47	0.500
23	320	98.47	0.500
24	200	98.09	0.500
26	320	99.24	0.500
27	288	98.28	0.500
29	350	98.28	0.250
30	320	98.09	0.500
32	232	98.09	0.250
33	288	97.90	0.500
35	256	98.09	0.250
36	350	99.05	0.500
38	256	98.09	0.250
39	350	98.66	0.250
41	320	98.66	0.250
42	320	98.28	0.517
44	288	98.66	0.500
45	288	98.47	0.250
47	320	99.05	0.500
48	320	98.09	0.500
50	288	99.05	0.250
51	350	99.24	0.250
53	224	97.14	0.250
54	320	98.66	0.517
56	288	98.47	0.500
57	256	98.28	0.250
59	350	98.66	0.250
60	256	98.47	0.250

APPENDIX A6
 FAULT SIMULATION RESULT FOR BENCH MARK CIRCUIT C499.BENCH
 (FOR FEEDBACK POLYNOMIAL $1+X^{60}+X^{61}+X^{63}+X^{64}$)

Seed	No. of Test Vectors	Fault Coverage (%)	CPU Time (sec)
2	480	98.15	0.250
3	384	97.89	0.517
5	512	98.29	0.500
7	190	96.70	0.500
8	416	98.02	0.500
9	480	98.29	0.250
11	512	98.15	0.250
12	448	98.02	0.250
14	448	98.15	0.500
15	480	98.55	0.250
17	416	98.42	0.250
18	384	97.76	0.500
20	512	98.29	0.250
21	512	98.81	0.250
23	384	98.02	0.517
24	576	98.42	0.517
26	416	97.76	0.250
27	512	98.15	0.517
29	448	98.15	0.500
30	480	98.29	0.250
32	512	98.81	0.250
33	480	98.29	0.250
35	384	98.42	0.517
36	512	98.81	0.250
38	416	98.02	0.517
39	352	98.15	0.500
41	480	98.55	0.517
42	512	98.81	0.250
44	448	98.02	0.500
45	448	98.42	0.250
47	480	97.89	0.250
48	416	98.29	0.500
50	384	98.15	0.500
51	512	98.15	0.250
53	512	98.15	0.250
54	384	97.76	0.250
56	576	98.55	0.250
57	416	98.02	0.250
59	512	98.55	0.250
60	448	98.02	0.250

APPENDIX A7
 FAULT SIMULATION RESULT FOR BENCH MARK CIRCUIT C499.BENCH
 (FOR FEEDBACK POLYNOMIAL $1+X+X^3+X^4+X^{64}$)

Seed	No. of Test Vectors	Fault Coverage (%)	CPU Time (sec)
2	448	98.02	0.250
3	512	98.15	0.250
5	416	98.02	0.250
7	190	96.17	0.250
8	576	98.68	0.500
9	384	98.29	0.250
11	512	98.02	0.517
12	512	97.89	0.250
14	384	97.49	0.250
15	416	98.29	0.250
17	480	97.76	0.250
18	448	98.29	0.500
20	448	97.76	0.250
21	512	98.55	0.250
23	480	98.42	0.250
24	416	98.02	0.500
26	512	98.55	0.250
27	384	96.70	0.250
29	480	98.02	0.517
30	350	97.36	0.250
32	512	98.42	0.250
33	480	98.29	0.250
35	384	97.76	0.250
36	512	98.55	0.517
38	416	98.15	0.500
39	352	98.29	0.500
41	480	98.55	0.250
42	512	98.15	0.250
44	448	98.02	0.517
45	448	98.29	0.517
47	480	98.15	0.500
48	416	98.29	0.517
50	384	98.29	0.250
51	512	98.29	0.250
53	512	97.89	0.250
54	384	98.29	0.517
56	576	98.68	0.500
57	416	98.15	0.500
59	512	98.55	0.500
60	448	98.15	0.500

APPENDIX A8
 FAULT SIMULATION RESULT FOR BENCH MARK CIRCUIT C880.BENCH
 (FOR FEEDBACK POLYNOMIAL $1+X^{60}+X^{61}+X^{63}+X^{64}$)

Seed	No. of Test Vectors	Fault Coverage (%)	CPU Time (sec)
2	256	94.27	0.250
3	224	94.90	0.517
5	256	94.59	0.500
6	256	93.52	0.500
8	224	94.90	0.517
9	256	94.80	0.500
11	224	94.27	0.250
12	256	94.90	0.250
14	192	93.21	0.500
15	256	94.69	0.250
17	192	93.84	0.250
18	256	95.65	0.500
20	260	94.80	0.250
21	224	94.59	0.250
23	256	94.16	0.517
24	160	90.66	0.517
26	192	92.25	0.250
27	120	93.52	0.500
29	260	94.27	0.517
30	224	92.68	0.500
32	160	90.45	0.250
33	256	93.74	0.500
35	224	92.68	0.250
36	256	93.21	0.517
38	256	92.46	0.250
39	224	93.42	0.250
41	256	94.59	0.517
42	224	92.57	0.500
44	256	93.84	0.250
45	160	91.30	0.250
47	192	93.21	0.250
48	256	93.42	0.250
50	192	92.04	0.250
51	256	94.80	0.500
53	260	95.33	0.250
54	224	92.04	0.250
56	256	94.80	0.250
57	160	92.46	0.500
59	192	92.89	0.250
60	260	94.69	0.500

APPENDIX A9
 FAULT SIMULATION RESULT FOR BENCH MARK CIRCUIT C880.BENCH
 (FOR FEEDBACK POLYNOMIAL $1+X+X^3+X^4+X^{64}$)

Seed	No. of Test Vectors	Fault Coverage (%)	CPU Time (sec)
2	256	94.69	0.250
3	224	95.65	0.250
5	256	95.12	0.500
6	256	94.27	0.250
8	224	94.48	0.250
9	256	95.54	0.517
11	224	95.01	0.517
12	256	94.69	0.250
14	160	92.14	0.250
15	192	92.99	0.250
17	256	94.90	0.250
18	192	93.10	0.250
20	256	95.12	0.500
21	260	94.69	0.250
23	224	93.95	0.250
24	256	95.01	0.250
26	160	92.14	0.250
27	120	90.13	0.250
29	192	94.59	0.250
30	260	95.44	0.500
32	160	92.99	0.250
33	256	95.54	0.500
35	224	94.48	0.517
36	256	95.97	0.250
38	256	95.44	0.250
39	224	94.48	0.250
41	256	95.86	0.250
42	224	92.99	0.500
44	256	95.01	0.517
45	160	93.95	0.517
47	192	93.95	0.250
48	256	95.86	0.500
50	192	93.63	0.250
51	256	94.59	0.250
53	260	96.29	0.250
54	224	93.95	0.500
56	256	94.27	0.500
57	160	92.46	0.517
59	192	94.37	0.250
60	260	95.54	0.250

APPENDIX A10
 FAULT SIMULATION RESULT FOR BENCH MARK CIRCUIT C1355.BENCH
 (FOR FEEDBACK POLYNOMIAL $1+X^{60}+X^{61}+X^{63}+X^{64}$)

Seed	No. of Test Vectors	Fault Coverage (%)	CPU Time (sec)
2	1152	98.41	0.500
3	1088	98.09	0.500
5	736	96.32	0.250
6	672	95.87	0.250
8	960	98.54	0.500
9	928	97.52	0.250
11	864	97.21	0.250
12	1120	98.29	0.250
14	1056	98.29	0.517
15	640	96.57	0.517
17	640	96.89	0.250
18	576	96.06	0.517
20	832	97.84	0.517
21	180	92.00	0.517
23	896	97.01	0.250
24	800	96.82	0.500
26	768	97.27	0.250
27	960	98.09	0.250
29	704	95.74	0.250
30	512	94.41	0.500
32	512	96.38	0.250
33	1152	98.16	0.500
35	1088	97.84	0.250
36	512	96.25	0.517
38	736	97.59	0.250
39	672	96.57	0.500
41	960	98.03	0.250
42	928	98.16	0.250
44	864	97.01	0.500
45	1120	98.48	0.517
47	1056	97.71	0.500
48	640	96.13	0.500
50	640	96.44	0.500
51	576	96.44	0.250
53	832	97.46	0.500
54	896	98.16	0.250
56	816	97.21	0.500
57	768	96.76	0.250
59	960	97.52	0.500
60	728	96.44	0.250

APPENDIX A11
 FAULT SIMULATION RESULT FOR BENCH MARK CIRCUIT C1355.BENCH
 (FOR FEEDBACK POLYNOMIAL $1+X+X^3+X^4+X^{64}$)

Seed	No. of Test Vectors	Fault Coverage (%)	CPU Time (sec)
2	704	96.95	0.517
3	960	97.65	0.250
5	768	96.70	0.517
6	800	96.70	0.517
8	896	97.78	0.250
9	832	97.90	0.500
11	576	95.81	0.517
12	640	96.13	0.517
14	640	95.55	0.517
15	1056	97.59	0.250
17	1120	98.29	0.250
18	864	98.09	0.500
20	928	98.09	0.500
21	180	91.49	0.500
23	960	98.09	0.517
24	672	97.46	0.500
26	736	97.78	0.250
27	1088	98.41	0.500
29	1152	98.48	0.517
30	512	96.70	0.250
32	512	95.62	0.250
33	1152	97.65	0.500
35	1088	97.84	0.517
36	512	95.62	0.517
38	736	97.27	0.250
39	672	96.63	0.250
41	960	98.41	0.500
42	928	97.78	0.250
44	864	96.13	0.500
45	1120	98.29	0.250
47	1056	98.35	0.500
48	640	96.70	0.250
50	640	96.76	0.250
51	576	96.00	0.250
53	832	97.78	0.517
54	896	97.59	0.500
56	820	97.46	0.517
57	780	97.52	0.500
59	960	98.22	0.500
60	712	97.08	0.500

APPENDIX A12
 FAULT SIMULATION RESULT FOR BENCH MARK CIRCUIT C1908.BENCH
 (FOR FEEDBACK POLYNOMIAL $1+X^{60}+X^{61}+X^{63}+X^{64}$)

Seed	No. of Test Vectors	Fault Coverage (%)	CPU Time (sec)
2	1632	97.07	0.767
3	1184	97.13	0.517
5	1376	96.91	0.250
6	1120	96.06	0.250
8	1312	95.74	0.500
9	960	95.26	0.250
11	1280	96.97	0.250
12	992	95.37	0.767
14	992	96.91	0.767
15	1504	96.65	0.767
17	1344	95.90	0.250
18	1408	97.50	0.517
20	1344	97.34	0.500
21	1600	97.07	0.500
23	1472	97.07	0.517
24	880	96.33	0.500
26	1312	96.49	0.500
27	1568	97.98	0.517
29	1184	96.33	0.500
30	864	94.52	0.250
32	992	95.42	0.517
33	1632	98.24	0.517
35	1184	96.33	0.250
36	1376	97.29	0.517
38	1120	96.17	0.500
39	1312	96.65	0.250
41	960	92.87	0.500
42	1280	96.81	0.500
44	992	96.06	0.517
45	992	95.37	0.500
47	1504	98.08	0.517
48	1344	96.59	0.500
50	1408	96.22	0.250
51	1344	97.23	0.500
53	1600	97.77	0.517
54	1472	96.81	0.250
56	1312	96.91	0.517
57	1568	98.14	0.500
59	1184	96.28	0.500
60	864	96.28	0.250

APPENDIX A13
 FAULT SIMULATION RESULT FOR BENCH MARK CIRCUIT C1908.BENCH
 (FOR FEEDBACK POLYNOMIAL $1+X+X^3+X^4+X^{64}$)

Seed	No. of Test Vectors	Fault Coverage (%)	CPU Time (sec)
2	864	92.82	0.500
3	1184	96.12	0.517
5	1568	98.46	0.517
6	1312	95.42	0.250
8	1472	96.69	0.500
9	1600	97.18	0.250
11	1344	96.70	0.250
12	1408	97.23	0.500
14	1344	96.81	0.250
15	1504	96.86	0.517
17	992	94.36	0.500
18	992	96.06	0.250
20	1280	96.38	0.767
21	960	96.43	0.250
23	1312	96.43	0.517
24	880	94.41	0.250
26	1120	94.99	0.767
27	1376	96.86	0.767
29	1184	96.70	0.250
30	1632	96.64	0.250
32	992	94.84	0.500
33	1632	96.06	0.500
35	1184	95.42	0.250
36	1376	97.61	0.500
38	1120	97.61	0.250
39	1312	95.42	0.250
41	960	95.05	0.500
42	1280	97.02	0.250
44	992	95.42	0.517
45	992	96.17	0.500
47	1504	97.18	0.517
48	1344	96.65	0.250
50	1408	95.74	0.500
51	1344	96.54	0.250
53	1600	97.71	0.250
54	1472	97.82	0.250
56	1312	96.43	0.500
57	1568	97.45	0.500
59	1184	95.32	0.517
60	864	94.57	0.250

APPENDIX A14
 FAULT SIMULATION RESULT FOR BENCH MARK CIRCUIT C2670.BENCH
 (FOR FEEDBACK POLYNOMIAL $1+X^{60}+X^{61}+X^{63}+X^{64}$)

Seed	No. of Test Vectors	Fault Coverage (%)	CPU Time (sec)
2	416	83.25	0.517
3	448	82.85	0.250
5	288	82.82	0.767
6	992	84.02	0.767
8	736	83.91	0.767
9	320	83.15	0.250
11	480	83.11	0.500
12	320	82.71	0.500
14	768	83.95	0.250
15	672	83.22	0.767
17	640	83.51	0.250
18	672	83.66	0.517
20	736	83.76	0.250
21	672	83.62	0.250
23	832	84.24	0.250
24	768	83.87	0.250
26	448	83.33	0.250
27	672	84.09	0.250
29	412	83.11	0.517
30	288	83.22	0.500
32	416	83.84	0.500
33	448	83.51	0.517
35	288	83.00	0.500
36	250	83.55	0.500
38	992	84.35	0.500
39	736	83.98	0.250
41	320	82.09	0.500
42	480	83.98	0.250
44	320	82.34	0.517
45	768	83.95	0.500
47	672	84.17	0.500
48	640	84.35	0.517
50	672	83.18	0.500
51	736	84.09	0.500
53	672	83.84	0.517
54	832	84.06	0.517
56	768	84.02	0.517
57	448	83.00	0.500
59	672	83.87	0.500
60	412	82.96	0.250

APPENDIX A15
 FAULT SIMULATION RESULT FOR BENCH MARK CIRCUIT C2670.BENCH
 (FOR FEEDBACK POLYNOMIAL $1+X+X^3+X^4+X^{64}$)

Seed	No. of Test Vectors	Fault Coverage (%)	CPU Time (sec)
2	412	82.96	0.517
3	672	83.76	0.767
5	448	83.29	0.250
6	768	84.09	0.767
8	832	83.76	0.517
9	672	83.87	0.250
11	736	84.17	0.250
12	672	83.84	0.517
14	640	83.87	0.517
15	672	83.91	0.250
17	768	83.76	0.517
18	320	81.40	0.517
20	480	83.73	0.517
21	320	82.38	0.517
23	736	84.13	0.500
24	992	84.06	0.767
26	288	83.22	0.517
27	448	83.40	0.767
29	416	83.25	0.250
30	288	82.20	1.033
32	416	83.40	0.250
33	448	83.58	0.517
35	288	83.29	0.250
36	250	82.71	0.517
38	992	84.42	0.500
39	736	84.20	0.517
41	320	83.15	0.250
42	480	83.87	0.250
44	320	82.93	0.517
45	768	84.02	0.500
47	672	83.91	0.517
48	640	83.95	0.250
50	672	83.84	0.250
51	736	83.73	0.517
53	672	84.17	0.500
54	832	83.80	0.517
56	768	83.95	0.517
57	448	83.25	0.250
59	672	83.91	0.517
60	412	83.62	0.250

APPENDIX A16
 FAULT SIMULATION RESULT FOR BENCH MARK CIRCUIT C3540.BENCH
 (FOR FEEDBACK POLYNOMIAL $1+X^{60}+X^{61}+X^{63}+X^{64}$)

Seed	No. of Test Vectors	Fault Coverage (%)	CPU Time (sec)
2	896	93.58	0.250
3	864	93.85	0.250
5	896	93.90	0.250
6	960	93.85	1.033
8	800	93.09	0.517
9	928	94.08	1.017
11	640	92.50	0.767
12	992	94.46	0.767
14	864	93.61	0.500
15	800	93.61	0.517
17	540	91.57	0.517
18	896	93.23	0.767
20	768	92.94	0.250
21	800	93.29	1.017
23	960	94.11	0.517
24	832	93.44	0.500
26	800	93.26	0.517
27	928	93.87	0.250
29	928	94.20	0.767
30	736	91.83	0.517
32	640	93.15	0.517
33	896	93.87	0.517
35	864	92.97	0.767
36	896	92.68	0.517
38	960	94.49	0.517
39	800	92.88	0.250
41	928	93.47	0.500
42	640	92.01	0.517
44	992	93.55	0.517
45	864	93.79	0.517
47	800	93.76	0.517
48	896	93.79	0.517
50	768	92.77	0.500
51	800	94.11	0.767
53	960	94.20	0.517
54	832	93.99	0.517
56	800	92.68	0.517
57	928	93.50	0.500
59	928	94.11	0.517
60	736	93.44	0.517

APPENDIX A17
 FAULT SIMULATION RESULT FOR BENCH MARK CIRCUIT C3540.BENCH
 (FOR FEEDBACK POLYNOMIAL $1+X+X^3+X^4+X^{64}$)

Seed	No. of Test Vectors	Fault Coverage (%)	CPU Time (sec)
2	736	93.76	0.517
3	928	94.78	0.767
5	928	94.31	0.500
6	800	93.29	0.517
8	832	94.08	0.517
9	960	94.55	0.517
11	800	93.61	0.517
12	768	94.14	0.250
14	896	93.44	0.500
15	800	93.50	0.517
17	540	91.80	0.517
18	864	93.99	0.500
20	992	94.22	0.517
21	640	93.41	0.517
23	928	94.60	0.517
24	800	93.32	0.767
26	960	94.52	0.517
27	896	93.96	0.500
29	864	93.52	0.517
30	896	93.64	0.517
32	640	92.04	0.500
33	896	94.14	0.517
35	864	94.05	0.500
36	896	93.85	0.767
38	960	94.78	0.517
39	800	92.39	0.517
41	928	93.70	0.517
42	640	93.35	0.517
44	992	93.55	0.517
45	864	93.90	0.517
47	800	93.47	0.500
48	896	93.90	0.517
50	768	93.52	0.517
51	800	93.87	0.517
53	960	94.31	0.517
54	832	93.96	0.517
56	800	93.29	0.517
57	928	93.06	0.517
59	928	94.55	0.250
60	736	93.32	0.500

APPENDIX A18
 FAULT SIMULATION RESULT FOR BENCH MARK CIRCUIT C5315.BENCH
 (FOR FEEDBACK POLYNOMIAL $1+X^{60}+X^{61}+X^{63}+X^{64}$)

Seed	No. of Test Vectors	Fault Coverage (%)	CPU Time (sec)
2	768	98.51	0.517
3	1152	98.75	0.767
5	1056	98.73	0.767
6	640	98.34	0.500
8	640	98.17	0.517
9	864	98.52	0.517
11	960	98.62	0.500
12	960	98.62	0.767
14	768	98.39	0.500
15	1088	98.79	0.517
17	928	98.49	0.517
18	960	98.79	0.517
20	928	98.60	0.517
21	1088	98.75	0.517
22	600	98.26	0.517
24	832	98.52	0.517
26	1056	98.65	0.250
27	1088	98.71	0.517
29	736	98.36	0.517
30	1248	98.80	0.767
32	640	98.22	0.517
33	780	98.32	0.517
35	1140	98.77	0.517
36	1056	98.64	0.500
38	640	98.28	0.517
39	640	98.28	0.500
41	864	98.54	0.517
42	960	98.75	0.767
44	960	98.65	0.767
45	768	98.54	0.250
47	1088	98.64	0.767
48	928	98.69	0.767
50	960	98.54	0.517
51	928	98.60	0.767
53	1088	98.77	0.517
54	832	98.69	0.250
56	1056	98.64	0.250
57	1088	98.62	0.517
59	740	98.45	0.517
60	1236	98.69	0.517

APPENDIX A19
 FAULT SIMULATION RESULT FOR BENCH MARK CIRCUIT C5315.BENCH
 (FOR FEEDBACK POLYNOMIAL $1+X+X^3+X^4+X^{64}$)

Seed	No. of Test Vectors	Fault Coverage (%)	CPU Time (sec)
2	1248	98.75	0.517
3	736	98.43	0.767
5	1088	98.69	0.517
6	1056	98.71	0.767
8	832	98.51	0.517
9	1088	98.67	0.767
11	928	98.60	0.517
12	960	98.47	0.500
14	928	98.60	0.517
15	1088	98.73	0.517
17	768	98.71	0.517
18	960	98.56	0.517
20	960	98.60	0.517
21	864	98.67	0.517
22	560	98.30	0.517
24	640	98.28	0.500
26	640	98.36	0.767
27	1056	98.71	0.517
29	1152	98.65	0.767
30	740	98.58	0.517
32	768	98.28	0.767
33	1152	98.69	0.517
35	1056	98.65	0.500
36	640	98.22	0.500
38	640	98.41	0.500
39	864	98.36	0.517
41	960	98.64	0.517
42	960	98.79	0.767
44	768	98.56	0.517
45	1088	98.71	0.500
47	928	98.62	0.517
48	960	98.71	0.517
50	928	98.54	0.517
51	1088	98.62	0.767
53	832	98.45	0.517
54	1056	98.67	0.767
56	1088	98.71	0.517
57	736	98.58	0.517
59	1248	98.71	0.767
60	640	98.04	0.517

APPENDIX A20
 FAULT SIMULATION RESULT FOR BENCH MARK CIRCUIT C6288.BENCH
 (FOR FEEDBACK POLYNOMIAL $1+X^{60}+X^{61}+X^{63}+X^{64}$)

Seed	No. of Test Vectors	Fault Coverage (%)	CPU Time (sec)
2	96	99.30	0.767
3	96	99.25	0.767
5	96	99.29	0.767
6	96	99.21	0.517
8	64	98.39	0.517
9	64	98.35	0.767
11	60	99.11	0.517
12	64	98.75	0.517
14	96	99.33	0.767
15	96	99.21	0.767
17	96	98.84	0.767
18	64	97.57	0.767
20	64	97.68	0.517
21	96	99.03	0.767
23	96	98.83	0.517
24	96	99.02	0.767
26	64	98.22	0.767
27	64	98.57	0.767
29	128	99.48	0.767
30	128	99.46	0.767
32	64	97.81	0.767
33	128	99.51	0.767
35	128	99.54	0.767
36	96	99.15	0.767
38	96	99.21	0.767
39	64	98.46	0.517
41	64	98.35	0.517
42	64	98.72	0.767
44	96	99.43	0.767
45	96	99.46	0.767
47	96	99.21	0.767
48	64	97.92	0.767
50	64	97.57	0.767
51	96	98.74	0.767
53	96	99.06	0.517
54	96	99.04	0.767
56	64	98.06	0.767
57	64	98.19	0.767
59	128	99.42	0.767
60	128	99.46	0.767

APPENDIX A21
 FAULT SIMULATION RESULT FOR BENCH MARK CIRCUIT C6288.BENCH
 (FOR FEEDBACK POLYNOMIAL $1+X+X^3+X^4+X^{64}$)

Seed	No. of Test Vectors	Fault Coverage (%)	CPU Time (sec)
2	128	99.56	0.767
3	128	99.56	0.517
5	64	99.30	0.517
6	64	99.43	0.767
8	96	99.46	0.767
9	96	99.48	0.767
11	60	99.47	0.767
12	96	99.33	0.767
14	64	99.30	0.767
15	64	99.46	0.517
17	96	99.48	0.767
18	96	99.33	0.767
20	96	99.54	0.517
21	64	99.38	0.767
23	64	99.38	0.767
24	64	99.30	0.767
26	64	99.37	0.517
27	64	99.15	0.767
29	96	99.41	0.767
30	96	99.37	0.517
32	64	99.48	1.017
33	96	99.34	0.767
35	96	99.38	0.517
36	96	99.46	0.767
38	96	99.54	0.767
39	64	99.14	0.767
41	64	99.26	0.767
42	64	99.02	0.767
44	96	99.35	0.767
45	96	99.51	0.767
47	96	99.56	0.767
48	64	99.23	0.767
50	64	99.23	0.517
51	96	99.48	0.767
53	96	99.46	0.767
54	96	99.51	0.767
56	64	99.30	0.767
57	64	99.38	0.517
59	128	99.54	0.767
60	128	99.51	0.517

APPENDIX A22

SCREENSHOT OF FAULT SIMULATION RESULT OF CIRCUIT C432.BENCH
(FOR FEEDBACK POLYNOMIAL $1+X^{60}+X^{61}+X^{63}+X^{64}$)

```

C:\>C:\WINDOWS\system32\cmd.exe

C:\bin>fsim -t output1.test c432.bench
*****
*                                     *
*           Welcome to fsim (version 1.2)           *
*                                     *
*           Dong S. Ha <ha@vt.edu>                  *
*           Web: http://www.ee.vt.edu/ha             *
*           Virginia Polytechnic Institute & State University *
*                                     *
*****

*****      SUMMARY OF FAULT SIMULATION RESULTS      *****
1. Circuit structure
   Name of the circuit           : combinational_logic_example_"c432
   "
   Number of primary inputs      : 36
   Number of primary outputs     : 7
   Number of gates               : 160
   Level of the circuit          : 17

2. Simulation parameters
   Simulation mode                : file <output1.test>

3. Simulation results
   Number of test patterns applied : 200
   Fault coverage                  : 98.282 %
   Number of collapsed faults     : 524
   Number of detected faults      : 515
   Number of undetected faults    : 9

4. Memory used                   : 168112 Kbytes

5. CPU time
   Initialization                 : 0.250 secs
   Fault simulation                : 0.000 secs
   Total                          : 0.250 secs

```

APPENDIX A23

SCREENSHOT OF FAULT SIMULATION RESULT OF CIRCUIT C499.BENCH
(FOR FEEDBACK POLYNOMIAL $1+X^{60}+X^{61}+X^{63}+X^{64}$)

```

C:\WINDOWS\system32\cmd.exe
C:\bin>fsim -t output2.test c499.bench
*****
*                               *
*       Welcome to fsim (version 1.2)       *
*                               *
*       Dong S. Ha (ha@vt.edu)               *
*       Web: http://www.ee.vt.edu/ha         *
*       Virginia Polytechnic Institute & State University *
*                               *
*****

***** SUMMARY OF FAULT SIMULATION RESULTS *****
1. Circuit structure
   Name of the circuit                : combinational_logic_example__"c49
9"
   Number of primary inputs           : 41
   Number of primary outputs          : 32
   Number of gates                     : 202
   Level of the circuit               : 11

2. Simulation parameters
   Simulation mode                     : file (output2.test)

3. Simulation results
   Number of test patterns applied     : 190
   Fault coverage                      : 96.702 %
   Number of collapsed faults          : 758
   Number of detected faults           : 733
   Number of undetected faults         : 25

4. Memory used                        : 168128 Kbytes

5. CPU time
   Initialization                     : 0.250 secs
   Fault simulation                    : 0.000 secs
   Total                              : 0.250 secs
  
```

APPENDIX A24
 SCRRENSHOT OF FAULT SIMULATION RESULT OF CIRCUIT C1908.BENCH
 (FOR FEEDBACK POLYNOMIAL $1+X^{60}+X^{61}+X^{63}+X^{64}$)

```

C:\WINDOWS\system32\cmd.exe
C:\bin>fsim -t output1.test c1908.bench
*****
*                               *
*       Welcome to fsim (version 1.2)       *
*                               *
*       Dong S. Ha (ha@vt.edu)              *
*       Web: http://www.ee.vt.edu/ha        *
*       Virginia Polytechnic Institute & State University *
*                               *
*****

*****      SUMMARY OF FAULT SIMULATION RESULTS      *****
1. Circuit structure
   Name of the circuit                : combinational_logic_example_"c190
8"
   Number of primary inputs           : 33
   Number of primary outputs          : 25
   Number of gates                    : 880
   Level of the circuit               : 40

2. Simulation parameters
   Simulation mode                    : file (output1.test)

3. Simulation results
   Number of test patterns applied    : 880
   Fault coverage                     : 96.328 %
   Number of collapsed faults         : 1879
   Number of detected faults          : 1810
   Number of undetected faults        : 69

4. Memory used                       : 168345 Kbytes

5. CPU time
   Initialization                    : 1.017 secs
   Fault simulation                   : 0.000 secs
   Total                             : 1.017 secs
  
```

APPENDIX A25
MATLAB PROGRAM FOR MEASURING CV OF PRV SEQUENCES

```
clc, clear all, close all
format short
load tvf.mat

x = seed06;
cv_x(1) = std(x)/mean(x)*100;

x = seed15;
cv_x(2) = std(x)/mean(x)*100;

x = seed23;
cv_x(3) = std(x)/mean(x)*100;

x = seed24;
cv_x(4) = std(x)/mean(x)*100;

x = seed26;
cv_x(5) = std(x)/mean(x)*100;

x = seed35;
cv_x(6) = std(x)/mean(x)*100;

x = seed38;
cv_x(7) = std(x)/mean(x)*100;

x = seed42;
cv_x(8) = std(x)/mean(x)*100;

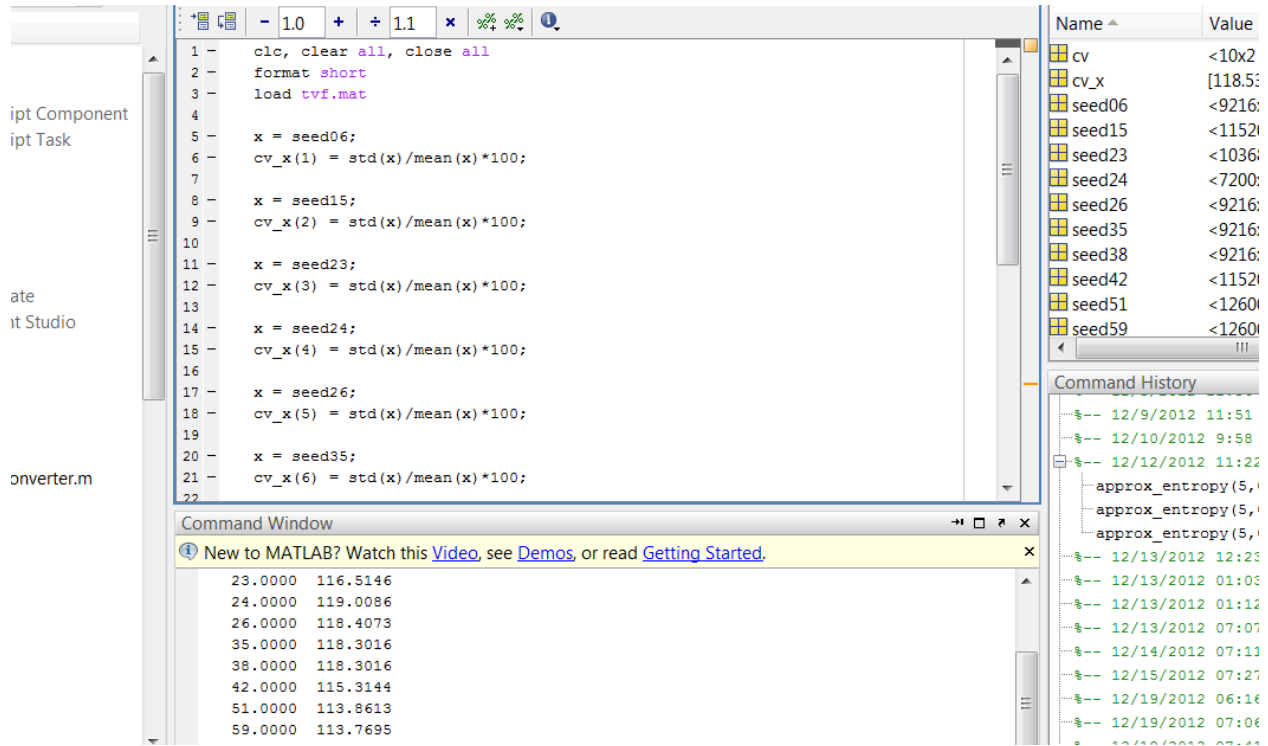
x = seed51;
cv_x(9) = std(x)/mean(x)*100;

x = seed59;
cv_x(10) = std(x)/mean(x)*100;

cv(:,2)=cv_x';
cv(:,1)=[6 15 23 24 26 35 38 42 51 59]';

cv
```

APPENDIX A26 SCREENSHOT OF THE OUTPUT OF MATLAB PROGRAM FOR MEASURING CV FOR C432.BENCH



OUTCOME OF THIS PROJECT

- [1] Ahmed, M. T., and Ali, L., “Implementation of Fibonacci Test Pattern Generator for Cost Effective IC Testing”, Proceedings of International Conference on Informatics, Electronics & Vision, Dhaka, Bangladesh, 2012.
- [2] Ahmed, M.T., and Ali, L., “Implementation of Fibonacci Test Pattern Generator for Cost Effective IC Testing”, International Journal of Electronics & Informatics (Invited for submission).