Non-linear Machines as Pseudo Random Pattern Generators for Digital Testing

by

Jing Zhong
B.Eng., Beijing University of Posts and Telecommunications, 1998

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

MASTER OF SCIENCE

in the Department of Computer Science

We accept this thesis as conforming
to the required standard

[redacted]

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

[redacted]

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

[redacted]

Dr. U. Stege, Departmental Member (Department of Computer Science)

[redacted]

Dr. P. F. Driessen, External Examiner

(Department of Electrical & Computer Engineering)

Supervisor: Dr. Jon C. Muzio

ABSTRACT

Linear machines such as the Linear Feedback Shift Registers (LFSRs) and Linear Hybrid Cellular Automata (LHCA) are often used as Pseudo Random Pattern Generators (PRPGs) in testing of digital circuits. In this thesis, a slightly non-linear machine, the Geffe generator, and its application as a PRPG in testing is investigated.

In the research, the randomness properties and the transition properties of the Geffe generator are studied. Theoretical analysis and results of the transition test show that the Geffe generator has a much larger transition capability than the LFSR or LHCA.

Fault simulation is also conducted using both the linear and the non-linear machines. Improvements in fault coverage for most ISCAS'89 benchmark circuits show that the non-linear machine, the Geffe generator, exhibits much better performance over the linear machines (the LFSR and LHCA) when it is used as the PRPG to detect stuck-at faults in sequential circuits.

Examiners

[redacted]

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

[redacted]

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

[redacted]

Dr. U. Stege, Departmental Member (Department of Computer Science)

[redacted]

Dr. P. F. Driessen, External Examiner

(Department of Electrical & Computer Engineering)

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

The focus of this research is the investigation of a non-linear machine, namely the Geffe generator, and its application as a Pseudo Random Pattern Generator (PRPG) in digital testing.

Testing of a circuit in a digital system is an experiment in which the circuit is exercised and its resulting response is analyzed to ascertain whether it behaves correctly [1, page 1]. The input stimulation for the circuit may be provided by a set of deterministic vectors, or by a pseudo random generator. Some autonomous linear machines such as Linear Feedback Shift Register (LFSR) [4, page 61] and Linear Hybrid Cellular Automata (LHCA) [23] are often used as Pseudo Random Pattern Generators (PRPG) in digital testing. Very high fault coverage can be achieved when such linear machines are used to detect stuck-at faults in combinational circuits.

However, for stuck-at faults in sequential circuits, the fault coverage can be pretty low. The reason for this phenomenon is that, to detect a fault in sequential circuits, we need more than one input pattern, since the outputs of the sequential circuits depend on both the current state and the previous state of the circuits. It is the variety of pattern transitions rather than the number of patterns that determines the performance of a generator. In [28], the authors provide a theoretical analysis and empirical comparisons to demonstrate that the LHCA are better than LFSRs as generators for sequential-type faults, and they suggest that this is because of the larger number of transition pairs generated by the LHCA.

The number of transitions obtained by linear machines can be quite limited. Here we want to turn to 'slightly' non-linear machines, which have higher transition capability. In [27] several slightly non-linear machines used in stream-cipher cryptography are presented. Their application in detecting delay faults and the simulation results of the ISCAS'85 benchmark circuits are presented in [22]. In this research, a non-linear

machine, the Geffe generator, and its application as a generator for testing stuck-at faults in sequential circuits is considered.

In Chapter 2, we present the background material for the research, including some concepts and techniques in digital testing, the structure and properties of both the linear machines (LFSR and LHCA) and the non-linear ones (the Geffe generator).

In Chapter 3, we introduce the experiments which are applied to both the linear and non-linear machines, namely the empirical Knuth randomness tests [12, page 54], the transition test, and fault simulation. Experimental specifications for each test are also given.

With respect to the Knuth tests, we use only three of them: the equidistribution test, the serial test, and the permutation test. Detailed algorithms for each test are described and parameters for each generator are also given. The transition test is also defined and the experimental procedure is explained.

Fault simulation includes both the simulation of combinational circuits and the simulation of sequential circuits, using the well known sets of benchmark circuits, ISCAS'85 and ISCAS'89. The characteristics of these benchmark circuits and the two simulators (FSIM and HOPE) are introduced briefly.

In Chapter 4, all of the experimental results including analysis are presented. Comparisons are made among the three generators. Finally, a conclusion is drawn based on the results and the analysis. In the fault simulation of the sequential circuits, we evaluate not only the performance of different generators, but also evaluate the effect on the fault coverage of different initial values of the flip-flops in the benchmark circuits.

The experimental results show that the non-linear machine, the Geffe generator, has a much larger transition capability than the linear ones, the LFSR or LHCA, and, when detecting single stuck-at faults in sequential circuits, it exhibits much better performance than the LFSR or LHCA.

In Chapter 5, we conclude this research and discuss some possible future work topics.

Chapter 2 Background

In this chapter, we cover the basic background of the thesis, including digital system testing, Linear Finite State Machines (LFSMs) and non-linear machines. All of the material is discussed in [1].

2.1 Digital System Testing

Testing of a circuit is intended to ascertain whether it behaves correctly. A digital circuit is one that deals with only digital signals. At the logic level, the information processed in a digital system is represented by discrete binary logic values.

We assume that the design of the circuit has been verified and meets its specification. But factors such as the physical defects in the circuit affect its behavior. These physical defects are called faults in the system, which include fabrication errors, fabrication defects and physical failures.

Figure 2.1 shows the general infrastructure for testing. Test patterns are generated and are applied to the Circuit Under Test (CUT). These patterns are either test sets specially produced for the CUT or generated by a Pseudo Random Pattern Generator (PRPG), which is often used in Built-In Self Test (BIST). As the testing is done, the output is analyzed to decide whether the CUT is faulty or not.

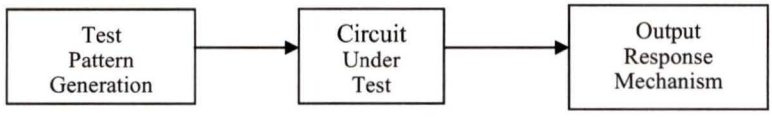


Figure 2.1 Digital system testing

Testing methods can be classified according to different criteria. If testing is performed concurrently with the normal system operation, it is called on-line testing. Otherwise if

testing is performed at a 'test mode', which is separate from the normal operation, it is called off-line testing. Testing methods can also be divided into self-testing and external-testing. Self-testing means that the testing stimuli are within the system itself while, in the external-testing, the testing stimuli are applied by an external device called a tester [1, page 4].

Testing includes many issues, such as: fault modeling, fault simulation, test generation, design for testability, signature analysis. The first three of these issues are covered below.

2.2 Fault Modeling

A fault model is the representation of the nature of the logical faults on the operation of the circuit [1, page 93]. Fault models are needed to analyze the result of a test because they allow a mathematical treatment of testing and diagnosis. According to their logic behavior, fault models can be classified as follows.

- Stuck-at faults: any fault condition that causes a logic gate to behave as though one of its inputs or its output is stuck at a logical 1 or 0 [4, page 4]. It is denoted by $s-a-v,\ v \in \{0,1\}$.

- Bridging faults: the logical fault representing a short between two signal lines which creates a new logic function. If the shorted lines form a feedback path that creates a new state in which the network can exist, and may lead to sequential behavior. If the short does not form this kind of feedback path, it is called a combinational fault [4, page 4].

- Delay faults: fault caused by signal delays in the circuit. They are used to represent two types of faults in transitions: the slow-to-rise transition, which means that it has a slower than expected transition from logic 0 to logic 1, and the slow-to-fall transitions, which means a slower than expected transition from logic 1 to logic 0 [11, page 15]. Two patterns are needed to detect a delay fault. For example, in Figure 2.2, to detect the slow-to-rise transition on line A, a transition

'0 → 1' is required to apply to the input A. In this case, the patterns (01, 11) can detect this fault.

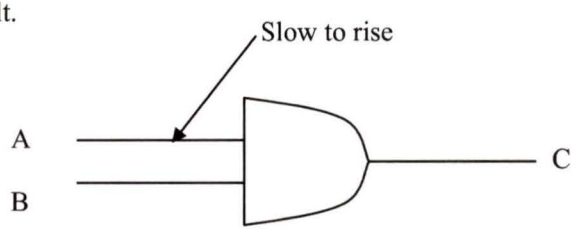


Figure 2.2 Example of a delay fault

In this thesis only single stuck-at faults are considered. This type of faults can be further divided into single stuck-at faults in combinational circuits, which need only one pattern to detect, and single stuck-at faults in sequential circuits, which, similar to delay faults, need more than one pattern to detect.

2.3 Fault Simulation

Logic simulation means to exercise a model by stimulating it with a set of input signals. It can be used for design verification, which attempts to ascertain that the design performs its specified behavior, both in function and in timing.

Logic simulation can also be used in fault simulation, which means to stimulate a circuit in the presence of faults [1, page 131]. When a test T is applied to the circuit, by comparing the fault simulation results with those of the fault-free simulation of the same circuit, one can determine whether the fault is covered or detected by the test T.

General fault simulation techniques include serial fault simulation and parallel fault simulation [1, page 134]. Serial fault simulation implies the simulation of the faults, one at a time. It consists of transforming the model of the fault-free circuit such that it models the circuit in the presence of the fault. Then the new model is simulated. The whole process is repeated for each fault and it is the simplest method of simulating faults. In

parallel fault simulation, the fault-free circuit and a fixed number of faulty circuits are simultaneously computed. The values of a signal in the fault-free circuit and the values of the corresponding signals in the faulty circuits are packed together in the same memory location of the host computer. Although this technique is much more complex for implementation, it reduces processing time greatly, since several faulty circuits can be simulated in parallel. Two simulators are used in our experiments, namely FSIM [14] and HOPE [15]. FSIM uses serial fault simulation and HOPE uses the parallel fault simulation technique.

2.4 Test Pattern Generation

The first step in testing is to generate test patterns for the circuit under test. Test pattern generation can be classified into categories including testing using a test set, exhaustive testing and pseudo random testing.

- Testing using a test set means to test a circuit with a set of input vectors which have been specifically designed for the circuit under test using a deterministic algorithm.
- Exhaustive testing means to test an n-input combinational circuit with all 2^n possible inputs. This method can guarantee 100% fault coverage of detectable faults that do not produce sequential behavior. But as n becomes larger, it is impractical to use this method since the number of patterns grows rapidly. Also this method is not effective for stuck-at faults in sequential circuits and delay faults since they need more than one pattern to detect such a fault.
- Pseudo random testing refers to testing a circuit with patterns that have many characteristics of random patterns, but the patterns are generated deterministically and hence are repeatable. Linear Finite State Machines (LFSM) are often used as the Pseudo Random Pattern Generators in testing.

2.5 Linear Finite State Machines[1]

Linear Finite State Machines (LFSMs) play an important role in digital circuits testing, as they can be used as Pseudo Random Pattern Generators as well as output compactors in testing [4, page 100]. In this section, first some definitions related to LFSM are given. After that, two specific kinds of LFSMs, namely the Linear Feedback Shift Register (LFSR) and the Linear Hybrid Cellular Automata (LHCA), are introduced.

Definition 1: A finite state machine is an algebraic structure $< S, I, Y, N, \delta >$, where S, I, and Y are finite sets of states, inputs, and outputs, respectively, N is a mapping from $S \times I$ into S, and δ is a mapping from S into Y [24, page 208].

Definition 2: A finite state machine $M =< S, I, Y, N, \delta >$ is linear if

(i) the state space S, the input space I, and the output space Y are each vector spaces over a finite field K (we let the dimensions of the spaces be n, p, and r, respectively); and

(ii) for the state vector s^i, input vector u^i, and output y^i at time i, the next-state and output functions are of the forms

$$N(s^i, u^i) = s^{i+1} = A \cdot s^i + B \cdot u^i \qquad (2.1)$$

and $$\delta(s^i) = y^i = C \cdot s^i,$$

where A, B, and C are matrices over K; s^i, u^i, and y^i are column vectors; the matrix operations are the usual matrix operations with arithmetic performed in the field K [24, page 297].

Figure 2.3 shows the block diagram of a typical linear finite state machine [24, page 297].

---

[1] All computations in this section are over GF(2).

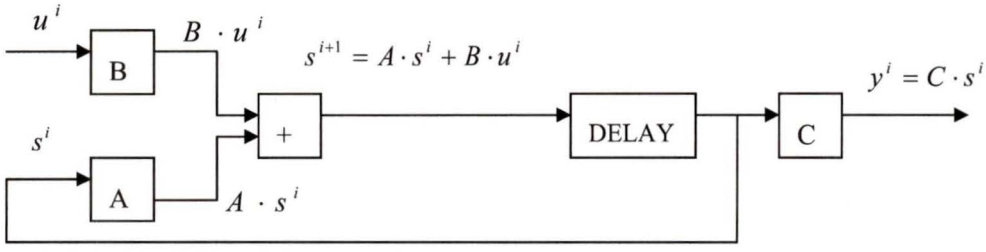


Figure 2.3 Block diagram of a Linear Finite State Machine

In this thesis, we only consider autonomous LFSMs, which means that our machines have no external inputs, thus the matrix B is a 0-matrix. The next-state function of the machine (equation 2.1) can be reduced to:

$$N(s^i, u^i) = s^{i+1} = A \cdot s^i \qquad (2.2)$$

Here A is called the state transition matrix. The characteristic polynomial of the transition matrix can also be used to describe the LFSMs. It is defined as follows [24, page 310].

Definition 3: The characteristic polynomial $f(x)$ of an $n \times n$ matrix A is the polynomial $\det(xI - A)$.

Since the primitive polynomial is very important, its definition is given below [25, page 40].

Definition 4: An irreducible polynomial $f(x)$ of degree n is said to be primitive if the smallest positive integer k for which $f(x)$ divides $x^k - 1$ is $k = 2^n - 1$. And a polynomial of degree no less than 1 is irreducible if it has no factors except scalars and scalar multiples of itself. That is, it cannot be factored into a product of lower-degree polynomials.

It is proven in [4, page 70] that only LFSMs that can be described by primitive characteristic polynomials can cycle through the maximum $2^n - 1$ non-zero states.

2.5.1 Linear Feedback Shift Register (LFSR)

A Linear Feedback Shift Register (LFSR) is a shift register that, when clocked, advances the signal through the register from one bit to the next bit in the shift register [13]. Some of the outputs are combined in exclusive-OR configuration to form a feedback mechanism. An autonomous LFSR can be formed by performing exclusive-OR on the outputs of two or more of the clocked D flip-flops together and feeding those outputs back into the inputs of the flip-flops. Figure 2.4 shows the structure of a 3-bit LFSR.

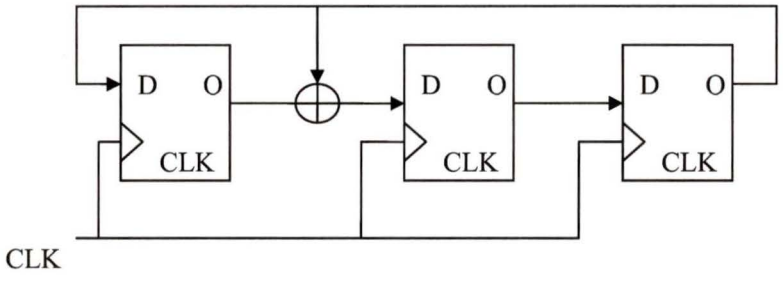
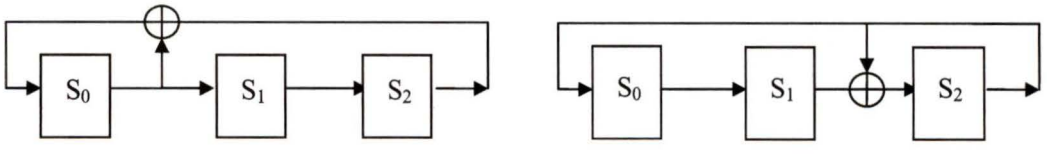


Figure 2.4 Structure of a 3-bit LFSR

According to the different positions of the XOR gates, LFSRs are generally divided into two main types: External XOR LFSR and Internal XOR LFSR. Figure 2.5 shows both types of the LFSRs. Here each cell represents a D flip-flop in a shift-register structure and '⊕' denotes an XOR gate.



External XOR LFSR Internal XOR LFSR

Figure 2.5 Two types of LFSR

We next use the state transition matrix and the characteristic polynomial to describe the internal XOR LFSR in Figure 2.5.

From the structure of the LFSR, the next state equations can be obtained as follows:

$$\begin{aligned} s_0^{i+1} &= s_2^i \\ s_1^{i+1} &= s_0^i \\ s_2^{i+1} &= s_1^i + s_2^i \end{aligned},$$

and in matrix form:

$$s^{i+1} = A \cdot s^i = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix} \cdot s^i,$$

so the transition matrix of the LFSR is

$$A = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix}.$$

The characteristic polynomial of matrix A is

$$f(x) = \det(xI - A) = \begin{vmatrix} x & 0 & 1 \\ 1 & x & 0 \\ 0 & 1 & 1+x \end{vmatrix} = x^3 + x^2 + 1.$$

It is easy to construct LFSRs with characteristic polynomials since there is a simple one-to-one correspondence between LFSRs and polynomials [4, page 66]. To implement the n-cell internal XOR LFSR from the characteristic polynomial $f(x) = 1 + \sum_{i=1}^{n} c_i x^i$, where $c_i \in \{0,1\}$, if $c_i = 1$, $1 < i < n$, the input of the $(i+1)$-th cell is the XOR result of the outputs of the i-th and the n-th cells, otherwise is the output of the i-th cell.

2.5.2 Linear Hybrid Cellular Automata (LHCA)

A Linear Cellular Automata (LCA) is a linear finite machine, composed of uniform arrays of cells in a k–dimensional space. Each cell consists of a single memory element

and a next-state computation function, which is called the computation rule, or rule. An LCA is called hybrid if two or more rules are used [8]. In this research, only autonomous one-dimensional (1-D) LHCA are considered. Since 1-D LHCA restrict communication to local neighborhoods, there are in total $2^{2^3} = 256$ rules [3]. It is shown in [21] that only 8 of the 256 rules are linear and, of these rules, only certain combinations of rule 90 and rule 150 may lead to LHCA with primitive characteristic polynomial.

Rule 90: $s_k^{i+1} = s_{k-1}^i + s_{k+1}^i$

Rule 150: $s_k^{i+1} = s_{k-1}^i + s_k^i + s_{k+1}^i$

Here s_k^{i+1} denotes the next-state of the k-th cell, and s_{k-1}^i, s_k^i s_{k+1}^i denote the current states of the $(k-1)$-th, the k-th, and the $(k+1)$-th cell, respectively.

Figure 2.6 shows the structure of a 3-cell LHCA.

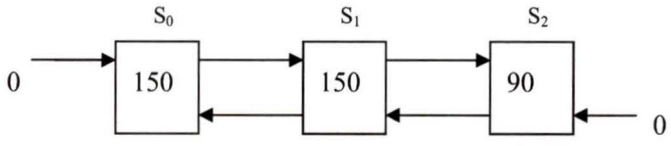


Figure 2.6 Structure of a 3-cell LHCA

The next-state equations of the three cells in Figure 2.6 are defined below.

$$s_0^{i+1} = s_0^i + s_1^i$$
$$s_1^{i+1} = s_0^i + s_1^i + s_2^i$$
$$s_2^{i+1} = s_1^i$$

As an LFSM, an LHCA can also be described by a transition matrix and a characteristic polynomial. For example, the transition matrix of the LHCA in Figure 2.6 is $\begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$.

Its characteristic polynomial is

$$f(x)=\begin{vmatrix} 1+x & 1 & 0 \\ 1 & 1+x & 1 \\ 0 & 1 & x \end{vmatrix} = x^3 + x + 1 .$$

The implementation of LHCA from characteristic polynomials is much harder since there is no one-to-one correspondence between LHCA and polynomials. The authors of [10] give a method for the synthesis of an LHCA from a given irreducible polynomial.

2.6 Non-linear Machines

In this section several 'slightly' non-linear machines and their applications in different areas are introduced.

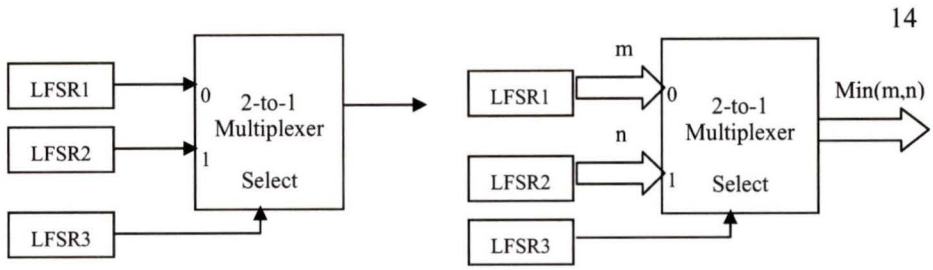
Non-linear machines refer to systems which have non-linear operations when changing from the current state to a next state. The definition of the linear function [24, page 297] is as follows.

Definition 5: A function $f: V \rightarrow V'$ is a linear function from a vector space V into a vector space V' over the same scalar field as V if, for all c_1 and c_2 in K and all v_1 and v_2 in V,

$$f(c_1 v_1 + c_2 v_2) = c_1 f(v_1) + c_2 f(v_2) .$$

In the binary field, linear operations include only modulo 2 addition, modulo 2 scalar multiplication, and unit delays. Other Boolean operations such as AND, OR, NAND, NOR are non-linear.

By adding some non-linear operations to an LFSM we can get some 'slightly' non-linear machines. One example of a possible non-linear machine is the Geffe generator [27]. It consists of three LFSRs and one 2-to-1 multiplexer, as shown in Figure 2.7 (a). At any given time, the generator outputs a bit generated by either LFSR1 or LFSR2. One bit from the pattern generated by LFSR3 decides which LFSR is chosen for the output.



(a) (b)

Figure 2.7 The Geffe generator (a) Original version (b) Modified version

In this research, since the generator is used to produce patterns for testing, we modify it into a similar form (see Figure 2.7 (b)). At any given time, the generator outputs a pattern m or n bits wide, instead of a single bit. Table 2.1 shows an example for $m = 3$ and $n = 4$. The bold bit from LFSR3 selects which pattern is chosen for the output.

Time	LFSR-1 $x^3 + x^2 + 1$	LFSR-2 $x^4 + x + 1$	LFSR-3 $x^3 + x + 1$	Output Pattern
1	001	**000**1	00**1**	000
2	**101**	1100	11**0**	101
3	111	**011**0	01**1**	011
4	110	**001**1	11**1**	001
5	011	**110**1	10**1**	110
6	**100**	1010	10**0**	100

Table 2.1 How the Geffe generator works when $m = 3$ and $n = 4$

Another example of a 'slightly' non-linear machine is the threshold generator (see Figure 2.8). This machine consists of a majority function and n LFSRs, n is an odd number. At any given time, if the the n LFSRs produce more 1's than 0's, the generator outputs a

'1', otherwise it outputs a '0'. Similar to the Geffe generator, the threshold generator could also output m bits in parallel with some modification to its structure.

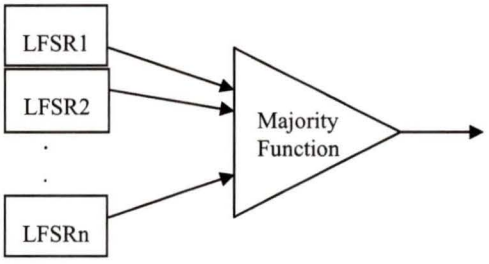


Figure 2.8 The threshold generator

The shrinking generator (see Figure 2.9) is also non-linear. The black dot in the Figure denotes a conditional check. Two LFSRs (LFSR-S and LFSR-A) are included in this machine. At any given time t_i, if the output s_i of LFSR-S is 1, then the generator outputs the output a_i of LFSR-A, otherwise it discards a_i.

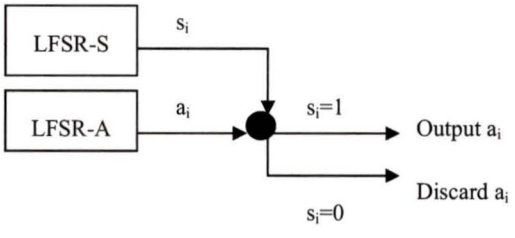


Figure 2.9 The shrinking generator

Non-linear machines are widely used in the field of cryptography [17]. Both linear machines (such as LFSR and LHCA) and non-linear machines (such as the Geffe generator) can be used as the pseudo random generator in a cipher system. Research in [18] has shown that linear machines are more vulnerable to cryptal attacks than the non-linear machines.

Similarly, we may think of using the non-linear machines as PRPG in digital testing. Will the non-linear generators have better performance than the linear ones in testing? In the following chapters, some experiments are designed and conducted to both the linear and non-linear machines to see whether the non-linear ones have superior performance.

2.7 Conclusion

In this chapter the background of this thesis is presented. Concepts and techniques in digital testing, LFSMs, and some non-linear machines are introduced. In Chapter 3, three experiments designed for both the linear and non-linear machines, namely the randomness tests, the transition test, and fault simulation, are discussed. Experimental results and the conclusions are given in Chapter 4.

Chapter 3 The Experiments

In this chapter we introduce the tests applied to both the linear and the non-linear machines, which include the randomness tests, the transition test and fault simulation. The experimental specifications are also given for each test.

3.1 Randomness Tests

The randomness tests are to evaluate the randomness of a sequence, or of a sequence of patterns. There are many methods described in the literature, including the Knuth tests [12, page 54] and Golomb's three criteria for (pseudo) random sequences [17, page 108]. In our research we use three tests from the empirical set of Knuth: the equidistrubution test, the serial test and the permutation test.

Each of the three tests is applied to a sequence $< Y_n >$ (3.1) of n integers which are distributed between 0 and $d-1$, here d is a positive integer.

$$< Y_n >= Y_0, Y_1, Y_2, ..., \qquad (3.1)$$

Below we briefly introduce the "Chi-square" statistical test that is part of all the three Knuth tests used [12, page 35].

Suppose that every observation can fall into one of k categories. We take n independent observations, with n a fairly large number, count the number of observations falling into each of the k categories and compute the quantity V given in equation (3.2) below. Here, P_s is the probability that each observation falls into category s, and N_s denotes the number of observations which fall into category s.

$$V = \sum_{1 \le s \le k} \frac{(N_s - nP_s)^2}{nP_s} \qquad (3.2)$$

The computed value of V is compared with the selected values of the chi-square distribution listed in Table 3.1 [12], with $v = k - 1$. When $v > 30$, the reference V values can be computed with the equation $V = v + 2\sqrt{v}x_p + \frac{4}{3}x_p^2 - \frac{2}{3}$. The different x_p values are listed in the last row of Table 3.1. If V lies between 5% and 95%, then the sequence passes the test. Otherwise it fails.

	p =99%	p =95%	p =75%	p =50%	p =25%	p =5%	p =1%
v=1	0.00016	0.00393	0.1015	0.4549	1.323	3.841	6.635
v=2	0.00201	0.1026	0.5753	1.386	2.773	5.991	9.210
v=3	0.1148	0.3518	1.213	2.366	4.108	7.815	11.34
v=4	0.2971	0.7107	1.923	3.357	5.385	9.488	13.28
v=5	0.5543	1.1455	2.675	4.351	6.626	11.07	15.09
v=6	0.8720	1.635	3.455	5.348	7.841	12.59	16.81
v=7	1.239	2.167	4.255	6.346	9.037	14.07	18.48
v=8	1.646	2.733	5.071	7.344	10.22	15.51	20.09
v=9	2.088	3.325	5.899	8.343	11.39	16.92	21.67
v=10	2.558	3.940	6.737	9.342	12.55	18.31	23.21
v=11	3.053	4.575	7.584	10.34	13.70	19.68	24.73
v=12	3.571	5.226	8.438	11.34	14.84	21.03	26.22
v=15	5.229	7.261	11.04	14.34	18.25	25.00	30.58
v=20	8.260	10.85	15.45	19.34	23.83	31.41	37.57
v=30	14.95	18.49	24.48	29.34	34.80	43.77	50.89
v>30	Approximately $v + 2\sqrt{v}x_p + \frac{4}{3}x_p^2 - \frac{2}{3}$						
x_p =	-2.33	-1.64	-.675	0.00	0.675	1.64	2.33

Table 3.1 Selected values of the chi-square distribution

The following example shows how the chi-square statistical test is applied in practice.

In a dice-throwing experiment, it is assumed that one can receive 1,2, 3, 4, 5, or 6 with equal probability. When two dice are used in the experiment, the probability p_s of obtaining a given total, s, on a single throw is as follows:

Value of s	2	3	4	5	6	7	8	9	10	11	12
Probability, p_s	$\frac{1}{36}$	$\frac{1}{18}$	$\frac{1}{12}$	$\frac{1}{9}$	$\frac{5}{36}$	$\frac{1}{6}$	$\frac{5}{36}$	$\frac{1}{9}$	$\frac{1}{12}$	$\frac{1}{18}$	$\frac{1}{36}$

For example, if one throws 108 times, and possible observed results could be:

Value of s	2	3	4	5	6	7	8	9	10	11	12
Observed number, N_s	2	7	8	10	16	20	14	13	12	5	1
Expected number, nP_s	3	6	9	12	15	18	15	12	9	6	3

The value V can be computed according to equation (3.2).

$$V = \frac{(2-3)^2}{3} + \frac{(7-6)^2}{6} + \cdots + \frac{(5-6)^2}{6} + \frac{(1-3)^2}{3} = 2\frac{53}{60}$$

Since $k = 11$, $v = k - 1 = 10$, the chi-square table shows that the probability that $V = 2\frac{53}{60}$ happens is between 95% and 99%. The observed values are too close to the expected ones and the results can not be considered random.

3.1.1 Equidistribution Test

The first requirement that a random sequence must meet is that its numbers are uniformly distributed in a range. Here we would like the numbers in a sequence to be uniformly distributed between 0 and $d-1$, d is a positive integer.

This test can be implemented as follows [12, page 55]: for each integer r, $0 \le r < d$, count the number of times $Y_j = r$ for $0 \le j < n$, and then apply the chi-square test using $k = d$ and probability $P_s = 1/d$ for each category.

3.1.2 Serial Test

More generally, we want pairs of successive numbers to be uniformly distributed in an independent manner and the serial test helps to measure as to how well this goal is being achieved.

To implement this test, first we count the number of times the pair $(Y_{2j}, Y_{2j+1}) = (q, r)$ occurs, for $0 \le j < n$. These counts are to be made for each pair of integers (q, r) with $0 \le q$, $r < d$, and we apply the chi-square test to these $k = d^2$ categories with probability $1/d^2$ in each category [12, page 55].

3.1.3 Permutation Test

In the permutation test, we divide the input sequence into n groups, where each group consists of t elements, that is $(Y_{jt}, Y_{jt+1}, ..., Y_{jt+t-1})$, $0 \le j < n$. The elements in each group can have $t!$ possible relative orderings; the number of times each ordering appears is counted, and a chi-square test is applied with $k = t!$ and with probability $1/t!$ for each ordering [12, page 59]. In our experiment, we use the permutation_3 test, i.e., $t = 3$.

3.1.4 Experimental Specification

In our research, we need to test the randomness of a set of binary patterns instead of a sequence of integers. To run these randomness tests, which are applicable only to a sequence, we change each binary pattern into its decimal form. The reason to do so is that

that is a one-to-one correspondence between an n-bit binary pattern and a decimal integer m, $m \in [0, 2^n - 1]$.

The software developed by our group [16] is used to run the tests. Lin Sun implemented the Knuth tests using the C language[1]. Since the algorithms of these tests are given above, here only the procedure for the experiments is given.

For an n-bit generator:

1. From a randomly chosen initial state, generate 10,000 binary patterns.
2. Change each binary pattern to its decimal form a_i, $i = 1, 2, \ldots, 10{,}000$.
3. For each decimal number a_i, apply a modulo 5 function[2], and get a new number b_i, $i = 1, 2, \ldots, 10{,}000$, where $b_i \in \{0,1,2,3,4\}$.
4. Apply the equidistribution test to the sequence b_i, $i = 1, 2, \ldots, 10{,}000$.
5. Apply the serial test to the sequence b_i, $i = 1, 2, \ldots, 10{,}000$.
6. Apply the permutation test to the sequence b_i, $i = 1, 2, \ldots, 10{,}000$.
7. Repeat steps 1-6 for $n = 10, 11, \ldots, 20$.

The parameters of the generators are defined as follows.

To generate n-bit wide patterns:

- LFSR: the n-degree LFSR with minimal weight (least possible number of non-zero coefficients) primitive polynomial [9] is chosen.

---

[1] The software is located at /local/bin/testing/ under the server Shannon in the DSD lab, Department of Victoria, University of Victoria. More information about the software is included in Appendix 3 and [16].

[2] In the serial test, the computer cannot handle the computation if the modulo function is taken with a value bigger than 10.

- LHCA: to make the results comparable to those of the other generators, we choose the n-degree LHCA with minimal weight primitive polynomial, which is the same one as that of the LFSR. Structures of this type of LHCA can be found in [9].
- The Geffe generator: in [18, page 135], the author notices that to obtain the best results in the entropy-related statistical test [19], the LFSRs in the Geffe generator should satisfy the following rule: $\gcd(d_1, d_2) = 1$, where gcd denotes the greatest common divisor and d_1, d_2 are the degrees of the LFSR1 and LFSR2 respectively. So, in this experiment, for LFSR1, we choose the n-degree LFSR with minimum weight primitive polynomial; for LFSR2, we choose the $(n+1)$-degree LFSR with minimum weight primitive polynomial. For LFSR3, since there is no particular restriction for the parameters, we use the minimum weight one with characteristic polynomial $x^9 + x^4 + 1$ in all Geffe generators regardless of the value of n.

3.2 Transition Test

The transition test counts the number of pattern transitions, or sub-state vector transitions in a sequence of patterns. The sub-state vector and the transition are defined as follows [28].

Definition 6: For a given n-cell state vector $s = (s_1, s_2, \cdots, s_n), s_p \in \{0,1\}$, $1 \le p \le n$, a k-cell substate vector w of s is defined by

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

and a transition corresponding to w is defined as

$$< (s_{i_1}, s_{i_2}, \cdots, s_{i_k}), (s_{i_1}^+, s_{i_2}^+, \cdots, s_{i_k}^+) >,$$

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

For example, for the 5-bit patterns $s = (s_1, s_2, s_3, s_4, s_5)$ in Figure 3.1, one can choose 3-bit sub-state vectors, $w = (s_1, s_3, s_4)$, shown in bold in Figure 3.1, and <(0, 1, 1), (0, 0, 1)> is called a transition. Two other sub-state vectors transitions are <(0, 0, 1), (1, 0, 0)> and <(1, 0, 0), (1, 0, 0)>.

s_1	s_2	s_3	s_4	s_5
0	0	**1**	**1**	0
0	0	**0**	**1**	1
1	0	**0**	**0**	1
1	1	**0**	**0**	0

Figure 3.1 Sub-state vectors

The number of pattern transitions, or sub-state vector transitions, is an important criterion to evaluate the performance of a generator when the patterns are used to detect faults in sequential circuits. The reason is that the outputs of a sequential circuit depend not only on the present input value but also on the past input values. It is more difficult to detect the single stuck-at faults in sequential circuits because we need a test sequence instead of a single test vector, which is enough to detect a single stuck-at fault in combinational circuits. It is the variety of transitions rather than the number of patterns that determines the performance of a sequence of patterns. In [28], the authors provide a theoretical analysis and empirical comparisons to see why the LHCA are better than the LFSRs as generators for sequential faults. They demonstrate that an n-cell LHCA with maximum length cycles has a much larger transition capability then an n-cell LFSR with maximum length cycles. Simulation results of the ISCAS'85 benchmark circuits show that when looking for delay faults, LHCA perform better than LFSRs.

Since the periods of both LFSR and LHCA are fairly small (maximum $2^n - 1$ for an n-cell LFSR or LHCA), the number of transitions obtained by these linear machines is quite

limited. In this research, we turn to the non-linear machines, which may have a larger transition capability and thus they have a better performance in detecting faults in sequential circuits. In this experiment, the transition capabilities of both the non-linear machine and the linear ones are investigated.

Experimental Specification

In the transition test, the number of sub-state vector transitions is used to evaluate the performance of generators. The parameters of the three generators are chosen the same way as those for the randomness tests in 3.1.4. The procedure of the experiments is given below.

For an n-bit generator:

1. From a randomly chosen initial state, generate 10,000 patterns.
2. Count the number of transitions of k-cell ($k = \lfloor n/2 \rfloor$) sub-state vectors, and the sub-state vectors come from the first k adjacent bits of the original patterns.
3. Count the number of transitions of k-cell ($k = \lfloor n/2 \rfloor$) sub-state vectors, which come from the middle k adjacent bits of the original patterns.
4. Count the number of transitions of k-cell ($k = \lfloor n/2 \rfloor$) sub-state vectors, which come from the 2^{nd}, 4^{th}, 6^{th}, …, $2k^{th}$ bits of the original patterns.
5. Repeat steps 1-4 for n=10, 11, …, 20.

3.3 Fault Simulation

Fault simulation is used to simulate the benchmark circuits with patterns generated by different generators so as to compare the performance of these generators with respect to fault coverage. In this part, the benchmark circuits and the tools used for the simulation are described briefly.

3.3.1 The Benchmark Circuits

Benchmark circuits are used in this research to compare the performance among different pattern generators. There are many benchmarks for digital testing, but here only the ISCAS'85 and ISCAS'89 benchmark circuits are used.

ISCAS'85 benchmark circuits

The ISCAS'85 benchmark circuits [7] are eleven combinational networks provided to authors at the 1985 International Symposium on Circuits And Systems. Each circuit is characterized in Table 3.2.

Circuit Name	Circuit Function	Total Gates	Input Lines	Output Lines	Total Faults[1]	Detectable Faults
c17	A small circuit	6 NAND	5	2	22	22
c432	Priority decoder	160(18 XOR)	36	7	524	520
c499[2]	ECAT	202(104 XOR)	41	32	758	750
c880	ALU and Control	383	60	26	942	942
c1355[2]	ECAT	546	41	32	1574	1566
c1908	ECAT	880	33	25	1879	1872
c2670	ALU and Control	1193	233	140	2747	2480
c3540	ALU and Control	1669	50	22	3428	3297
c5315	ALU and Control	2307	178	123	5350	5291
c6288	16-bit multiplier	2406	32	32	7744	7710
c7552	ALU and Control	3512	207	108	7550	7417

[1]Reduced equivalent fault set based on equivalence fault collapsing.

[2]Circuits c499 and c1355 are functionally equivalent. All XOR gates of c499 have been expanded into their 4-NAND gate equivalents in c1355.

Table 3.2 ISCAS'85 Benchmark Circuit characteristics

ISCAS'85 netlist format

The ISCAS'85 benchmark circuits are described in the ISCAS'85 netlist format [7]. Users can build their own circuits for simulation based on the format. As an example, the ISCAS'85 netlist format for the small circuit c17 (Figure 3.2) is attached in Appendix 1.

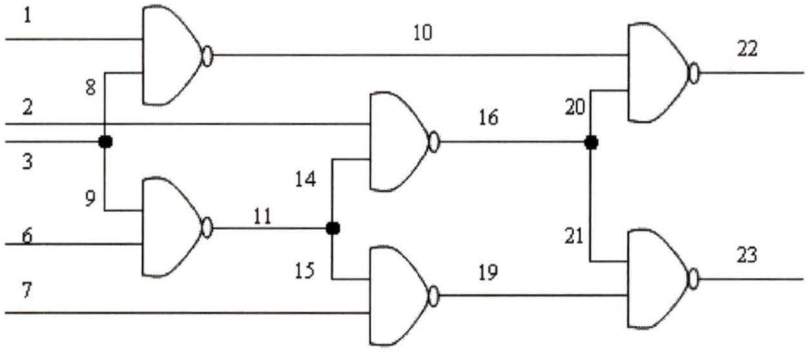


Figure 3.2 Structure of circuit c17

ISCAS'89 benchmark circuits

The ISCAS'89 benchmarks [5,6] are a set of 31 digital sequential circuits. They were distributed to participants of the Special Session on Sequential Test Generation, International Symposium on Circuits and Systems, May 1989. They are characterized in Table 3.3 [5].

Circuit Name	# of Primary Inputs	# of Primary Outputs	# of D-type Flipflops	# of AND/OR/ NOT Gates	# of Faults	Circuit Function
s27	4	1	3	10	32	Unclear
s208	11	2	8	96	215	Digital fractional multiplier, based on PLD
s298	3	6	14	119	308	Traffic light controller, based on PLD
s344	9	11	15	160	342	Removed redundancies from s349
s349	9	11	15	161	350	4-bit multiplier
s382	3	6	21	158	399	Removed redundancies from s400
s386	7	7	6	159	384	Controller
s400	3	6	21	162	424	Traffic light controller
s420	19	2	16	196	430	Digital fractional multiplier
s444	3	6	21	181	474	Traffic light controller
s510	19	7	6	211	564	Controller
s526n	3	6	21	194	553	Removed redundancies from s526
s526	3	6	21	193	555	Traffic light controller
s641	35	24	19	379	467	Removed redundancies from s713, based on PLD
s713	35	23	19	393	581	Based on PLD
s820	18	19	5	289	850	Removed redundancies from s832
s832	18	19	5	287	870	Based on PLD
s838	35	2	32	390	857	Digital fractional multiplier
s953	16	23	29	395	1079	Controller
s1196	14	14	18	529	1242	Removed redundancies from s1238
s1238	14	14	18	508	1355	A combinational circuit with randomly inserted FFs
s1423	17	5	74	657	1515	Unclear
s1488	8	19	6	653	1486	Removed redundancies from s1494
s1494	8	19	6	647	1506	Controller
s5378	35	49	179	2779	4603	Unclear
s9234	19	22	228	5597	6927	Real-chip based and rely on partial scan
s13207	31	121	669	7951	9815	Real-chip based and rely on partial scan
s15850	14	87	597	9772	11727	Real-chip based and rely on partial scan
s35932	35	320	1728	16065	39094	Unclear
s38417	28	106	1636	22179	31180	Real-chip based and rely on partial scan
s38584	12	278	1452	19253	36305	Real-chip based and rely on partial scan

Table 3.3 ISCAS'89 Benchmark Circuit characteristics

ISCAS'89 netlist format

The ISCAS'89 benchmark circuits are described in the ISCAS'89 netlist format [6], which is different from the ISCAS'85 format. As an example, the ISCAS'89 netlist format for the small circuit s27 (see Figure 3.3) is attached in Appendix 2.

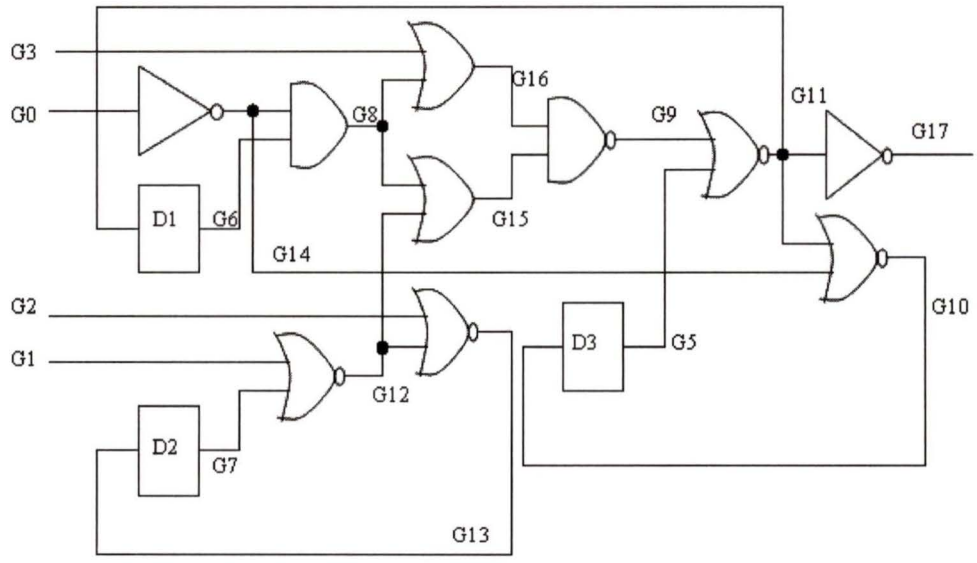


Figure 3.3 Structure of circuit s27

3.3.2 Fault Simulators

Two fault simulators are used in our research, namely FSIM [14] and HOPE [15]. FSIM was used to detect single stuck-at faults in combinational circuits; HOPE was used for single stuck-at faults in sequential circuits.

FSIM

FSIM ([14]) is a fault simulator for combinational circuits. It was developed in the Bradley Department of Electrical Engineering, Virginia Polytechnic Institute & State University (VPI&SU). It employs the parallel pattern single fault propagation technique [1, page 156].

FSIM reads circuits in ISCAS'89 netlist format, instead of the original ISCAS'85 netlist format, because the former is more flexible. With the option –I, it can also read circuits in ISCAS'85 netlist format.

FSIM can be downloaded at: http://www.ee.vt.edu/~ha/cadtools/distribution.cgi.

HOPE

HOPE ([15]) is a parallel fault simulator for synchronous sequential circuits. It is based on an earlier fault simulator called PROOFS [20], which employs several heuristics to drop faults efficiently and to avoid simulation of many inactive faults. HOPE was also developed in the Bradley Department of Electrical and Computer Engineering, Virginia Polytechnic Institute & State University. HOPE reads circuits in ISCAS'89 netlist format.

HOPE can be downloaded at: http://www.ee.vt.edu/~ha/cadtools/distribution.cgi.

3.3.3 Parameters of the Test Pattern Generators

To make the experimental results from the different generators comparable, first we specify the parameters of the three generators.

LFSR

An n-cell LFSR with primitive polynomial can cycle through all possible non-zero states and, for convenience, we narrow the category to LFSRs with minimal weight primitive

polynomials. So for an m-input benchmark circuit, the m-degree LFSR which has a minimal weight primitive polynomial is chosen. For example, for the ISCAS'85 benchmark circuit c432, which has 36 inputs, we choose the LFSR with the polynomial $x^{36}+x^{11}+1$. The characteristic polynomials of this type of LFSRs are listed in [9].

Since this type of LFSRs cycle through all possible non-zero states, they have the period of 2^n-1, n the degree of the polynomial.

LHCA

For consistency with the LFSR, LHCA which have the same characteristic polynomials as those of the LFSRs are chosen as the generator. For example, for the circuit c432, we choose the LHCA with the polynomial $x^{36}+x^{11}+1$. The structure of the LHCA is "010010010001100001011111000010010010". Here '1' means that the cell of the LHCA is defined by the rule 150 and '0' means the cell is defined by rule 90 (refer to 2.5.2). This type of LHCA as well as their characteristic polynomials can also be found in [9].

This type of LHCA also cycle through all possible non-zero states and thus have the period of 2^n-1, n the degree of the polynomial.

The Geffe generator

For the Geffe generator, three LFSRs need to be specified. In order to generate patterns for an n-input circuit, both LFSR1 and LFSR2 must have degree at least n. So for LFSR1, we choose the n-degree LFSR with minimal weight primitive polynomial. For LFSR2, we choose the $(n+1)$-degree LFSR with minimal weight primitive polynomial. For LFSR3, there is not much restriction except that the length of its cycle should not be too small when n is small (otherwise the cycle length of the Geffe generator is small). For convenience, we use the LFSR with polynomial x^9+x^4+1 for LFSR3 in all Geffe generators regardless of the value of n. As an example, for the circuit c432, the three LFSRs in the Geffe generator are chosen as follows:

LFSR1: LFSR with the characteristic polynomial of $x^{36}+x^{11}+1$;

LFSR2: LFSR with the characteristic polynomial of $x^{37}+x^{12}+x^{10}+x^{2}+1$;

LFSR3: LFSR with the characteristic polynomial of $x^{9}+x^{4}+1$.

It is stated in [27] that the Geffe generator has the period of $T_{Geffe}=LCM[(2^{n1}-1)(2^{n2}-1)(2^{n3}-1)]$, where $n1$, $n2$, $n3$ are the degrees of LFSR1, LFSR2, and LFSR3, respectively, and LCM denotes Least Common Multiple. In our experiments, with the parameters specified above, the period of the Geffe generator is:

$T_{Geffe}=LCM[(2^{n}-1)(2^{n+1}-1)(2^{9}-1)]$.

3.3.4 Experimental Specification

Fault simulation includes two parts: simulation of the combinational circuits (stuck-at faults), and simulation of the sequential circuits (stuck-at faults). In this section the experimental specification is given in detail.

Simulation of the combinational circuits (stuck-at faults)

Simulation of the combinational circuits means to stimulate the ISCAS'85 benchmark circuits with patterns generated by both linear and non-linear machines so as to compare the performance among these generators when they are used to detect stuck-at faults. We test:

- Non-linear machine versus linear machines: in this experiment, we test the performance of both linear and non-linear machines when they are used as the PRPG for combinational circuits. Comparisons are made among the results from different generators.
- Simulation of a small circuit: in this part the relation between the structure of a circuit and the 'power' of the input patterns is investigated. The small circuit c17

is analyzed as an example. For each stuck-at fault in the circuit, we find out all the patterns which can detect it by analyzing the structure of the circuit. Then, we use every single pattern to stimulate the circuit, and get the fault coverage for them respectively. Finally, we compare the results of these two parts and try to find the relation between them.

- Simulation of similar circuits: do circuits of the same function have similar properties? In this test the circuits are grouped according to their functions. For each group, we analyze the results of different circuits and see whether they exhibit similar properties.

The procedure for each test is listed below.

- Non-linear machine versus linear machines.

  For an n-input ISCAS'85 benchmark circuit, choose the three generators (LFSR, LHCA, and the Geffe generator) as specified in 3.3.3. For each generator,

  1) From 10 different randomly chosen initial states, generate 10 sets of patterns (each set contains 1,000 patterns).
  2) Apply each set of patterns as input to the benchmark circuit, get the fault coverage at the points of 1, 2, 5, 10, 20, 30, 40, 50, 100, 200, 300, 400, 500, and 1,000 patterns respectively.
  3) Get the best and median fault coverage from the 10 sets of values obtained from 2). Draw a fault coverage – number of patterns relationship curve.

- Simulation of a small circuit.

  For the small circuit c17,

  1) With the simulator, apply every single pattern as the input to stimulate the circuit. Get the fault coverage for each individual pattern.
  2) For each single stuck-at fault in the circuit, find out all the patterns that can detect it by analyzing the structure of the circuit.
  3) Compare and analyze the results from 1) and 2).

- Simulation of similar circuits.

    1) Group the results according to the function of the benchmark circuits.

    2) For each group, compare the results of different circuits. See whether they have similar properties.

Simulation of the sequential circuits (stuck-at faults)

Simulation of the sequential circuits means to stimulate the ISCAS'89 benchmark circuits with patterns generated by different generators so as to compare the performance among these generators when they are used to detect stuck-at faults in sequential circuits. The experiments include:

- Non-linear machine versus linear machines: in this experiment, comparison is made among the performance of the linear and non-linear machines when they are used as PRPG to test sequential circuits.

- Initial value of the flip-flops in the benchmark circuits: in this experiment, we change the initial value of the flip-flops in the benchmark circuits and investigate its effect on the fault coverage.

The procedure of each test is given as follows:

- Non-linear machine versus linear machines (in this test, the initial value of the flip-flops is assigned as an unknown value U).

  For an n-input ISCAS'89 benchmark circuit, choose the three generators (LFSR, LHCA, and the Geffe generator) as specified in 3.3.3. For each generator,

    1) From 10 different randomly chosen initial states, generate 10 sets of patterns (each set contains 20,000 patterns).

    2) Use each set of patterns as input to stimulate the circuit. Get the fault coverage at the points of 64, 100, 1,000, and 10,000 patterns respectively.

3) Get the best and median value of the fault coverage at each point, draw a fault-coverage – number of patterns relationship curve.

- Initial value of the flip-flops in the benchmark circuits.

For an n-input ISCAS'89 benchmark circuit, set the initial value of the flip-flops to logic '0'. Choose the three generators as specified in 3.3.3. For each generator,

1) Use the same 10 sets of patterns as those in the former test (non-linear machine versus linear machines) to stimulate the benchmark circuit, and get the fault coverage respectively (only when all 20,000 patterns are applied).

2) Get the best and median value of the fault coverage.

3) Compare the result from this experiment with that from the former one.

3.4 Conclusion

In this chapter, three kinds of tests used in our research are introduced. By following the experimental specification given in this chapter, we apply the randomness tests, the transition test, and fault simulation to both the linear machines (the LFSR and LHCA) and the non-linear machine (the Geffe generator). The experimental results as well as the analysis of these results are discussed in chapter 4.

Chapter 4 Experimental Results and Analysis

In this chapter we present the experimental results of the three tests, the randomness tests, the transition test, and fault simulation, conducted using both the linear and the non-linear machines.

4.1 Randomness Tests

As discussed in previous chapters, we run three specific tests from the empirical set designed by Knuth: the equidistribution test, the serial test, and the permutation test. The results of the randomness tests are listed in Table 4.1.

The results show that for the equidistribution test, all three generators exhibit good randomness property. Both the LFSR and LHCA pass more than half of the equidistribution tests. Actually the results for LFSRs and the LHCA are identical. The Geffe generator has better results as it passes all the 11 tests.

For the serial test, only the LHCA shows good randomness properties. It fails only two out of the eleven tests (when n equals to 10 and 12). Both the LFSR and the Geffe generator fail all the eleven serial tests.

The results of the permutation tests are similar to those of the serial test. The LHCA seems to be the only good one because it passes 9 out of 11 tests. While both the LFSR and the Geffe generator fail all the eleven permutation tests.

Analysis and Conclusion

The results of the randomness tests show that among the three generators, the LHCA seem to have the best randomness properties, while the Geffe generator exhibits similar (or slightly better) randomness properties as compared to the LFSR.

n	Equidistribution Test			Serial Test			Permutation Test		
	LFSR	LHCA	Geffe	LFSR	LHCA	Geffe	LFSR	LHCA	Geffe
10	FAIL	FAIL	PASS	FAIL	FAIL	FAIL	FAIL	FAIL	FAIL
11	FAIL	FAIL	PASS	FAIL	PASS	FAIL	FAIL	PASS	FAIL
12	FAIL	FAIL	PASS	FAIL	FAIL	FAIL	FAIL	PASS	FAIL
13	FAIL	FAIL	PASS	FAIL	PASS	FAIL	FAIL	PASS	FAIL
14	PASS	PASS	PASS	FAIL	PASS	FAIL	FAIL	PASS	FAIL
15	PASS	PASS	PASS	FAIL	PASS	FAIL	FAIL	PASS	FAIL
16	PASS	PASS	PASS	FAIL	PASS	FAIL	FAIL	FAIL	FAIL
17	PASS	PASS	PASS	FAIL	PASS	FAIL	FAIL	PASS	FAIL
18	PASS	PASS	PASS	FAIL	PASS	FAIL	FAIL	PASS	FAIL
19	PASS	PASS	PASS	FAIL	PASS	FAIL	FAIL	PASS	FAIL
20	PASS	PASS	PASS	FAIL	PASS	FAIL	FAIL	PASS	FAIL
Summary	7 Pass	7 Pass	11 Pass	0 Pass	9 Pass	0 Pass	0 Pass	9 Pass	0 Pass
	4 Fail	4 Fail	0 Fail	11 Fail	2 Fail	11 Fail	11 Fail	2 Fail	11 Fail

Table 4.1 Results of Knuth's randomness tests

Why is the randomness property of the Geffe generator similar to that of the LFSR? We take a look at the structure of the Geffe generator given in 2.6. It is composed of three LFSRs. At any given time, the Geffe generator outputs a pattern, either from LFSR1 or LFSR2, which is determined by the output of LFSR3. Although the multiplexer makes the generator 'slightly' non-linear and thus making the output a little more unpredictable, the output patterns of the Geffe generator are still parts of the output patterns of the two LFSRs (LFSR1 and LFSR2). So it is reasonable that the randomness property of the Geffe generator resembles that of the LFSR.

One more experiment is undertaken to investigate this hypothesis. We replace the three LFSRs in the Geffe generator with three LHCAs, which have the same characteristic polynomials as those of the LFSRs. Then the same three randomness tests are applied to

this modified generator Geffe*. The new results are shown in Table 4.2, together with those from the former experiment.

n	Equidistribution Test				Serial Test				Permutation Test			
	LFSR	LHCA	Geffe	Geffe*	LFSR	LHCA	Geffe	Geffe*	LFSR	LHCA	Geffe	Geffe*
10	FAIL	FAIL	PASS	FAIL	FAIL	FAIL	FAIL	PASS	FAIL	FAIL	FAIL	PASS
11	FAIL	FAIL	PASS	FAIL	FAIL	PASS	FAIL	FAIL	FAIL	PASS	FAIL	PASS
12	FAIL	FAIL	PASS	PASS	FAIL	FAIL	FAIL	PASS	FAIL	PASS	FAIL	PASS
13	FAIL	FAIL	PASS	PASS	FAIL	PASS	FAIL	PASS	FAIL	PASS	FAIL	PASS
14	PASS	PASS	PASS	FAIL	FAIL	PASS	FAIL	PASS	FAIL	PASS	FAIL	PASS
15	PASS	PASS	PASS	PASS	FAIL	PASS	FAIL	PASS	FAIL	PASS	FAIL	PASS
16	PASS	PASS	PASS	PASS	FAIL	PASS	FAIL	PASS	FAIL	FAIL	FAIL	PASS
17	PASS	PASS	PASS	PASS	FAIL	PASS	FAIL	PASS	FAIL	PASS	FAIL	PASS
18	PASS	PASS	PASS	PASS	FAIL	PASS	FAIL	PASS	FAIL	PASS	FAIL	FAIL
19	PASS	PASS	PASS	PASS	FAIL	PASS	FAIL	PASS	FAIL	PASS	FAIL	PASS
20	PASS	PASS	PASS	PASS	FAIL	PASS	FAIL	PASS	FAIL	PASS	FAIL	PASS
Sum	7 Pass	7 Pass	11 Pass	8 Pass	0 Pass	9 Pass	0 Pass	10 Pass	0 Pass	9 Pass	0 Pass	10 Pass
mary	4 Fail	4 Fail	0 Fail	3 Fail	11 Fail	2 Fail	11 Fail	1 Fail	11 Fail	2 Fail	11 Fail	1 Fail

Table 4.2 Results of Knuth's randomness tests using Geffe*

It is clear that the modified generator Geffe* exhibits similar properties to those of the LHCA. For instance, in the equidistribution test, Geffe* passes 8 out of 11 tests and the LHCA passes 7 out of 11 tests. In both the serial test and the permutation test, Geffe* passes 10 out of 11 tests and the LHCA passes 9 out of 11 tests. Also we can see that the results of Geffe* are quite different from those of the original Geffe generator.

In conclusion, among the three generators (LFSR, LHCA, and the Geffe generator), the LHCA have the best randomness properties. And the randomness properties of the Geffe generator resemble those of the linear machines of which it is composed.

4.2 Transition Test

The transition test is applied to the three generators according to the experimental specification given in 3.2. Each generator produces 10,000 vectors (each vector contains n bits). Three particular k -bit ($k = \lfloor n/2 \rfloor$) sub-state vectors are selected and the numbers of two-pattern transitions of the sub-state vectors are counted. First, to illustrate, we explain in detail how the results are obtained when n equals to 10.

First, the LFSR with the minimum weight primitive polynomial $x^{10} + x^3 + 1$ is chosen. From a randomly chosen initial state '0101100110', it generates 10,000 patterns. Then the first k ($k = \lfloor n/2 \rfloor = 5$) adjacent bits of the original patterns are picked as the sub-state vectors (bits S_1 to S_5, the bold underlined bits in Figure 4.1) and the transitions in these 10,000 sub-state vectors are counted. The result is that although there are $2^5 \times 2^5 = 1{,}024$ possible transitions, the LFSR only achieves 64 of them.

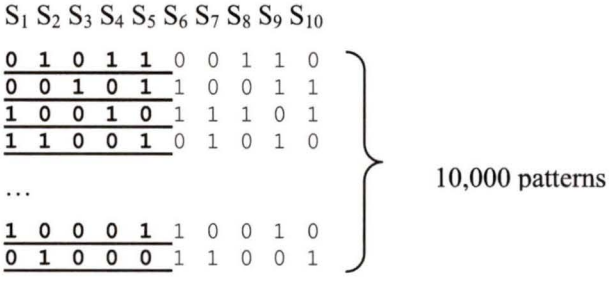


Figure 4.1 Result for transition test – LFSR

Then the sub-state vectors are changed to those taken from the middle adjacent k ($k = 5$) bits of the original patterns (bits S_3 to S_7) and the transitions are counted. This time, in the 10,000 sub-state vectors, there are still only 64 transitions.

Finally, we change the sub-state vectors to those from the 2nd, 4th, 6th, … , and $2k$ th bits (k =5) of the original patterns (bits S_2, S_4, S_6, S_8, and S_{10}) and count the number of transitions. And we get 1,023 out of 1,024 transitions.

The LHCA used in the experiment has the same characteristic polynomial as that of the LFSR: $x^{10}+x^3+1$. From a random initial state '0110011010', the LHCA generates 10,000 patterns. First the first 5 adjacent bits of the original patterns are chosen as the sub-state vectors (bits S_1 to S_5, the bold underlined bits in Figure 4.2). Similar to the LFSR, in these 10,000 sub-state vectors, the LHCA can only achieve 64 out of 1,024 possible transitions.

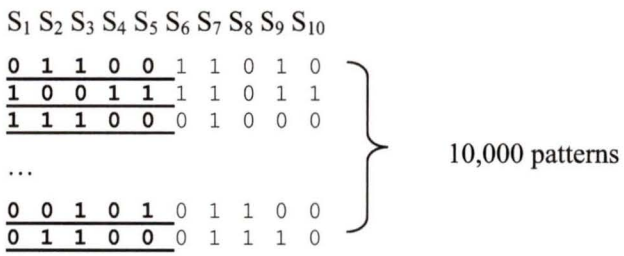


Figure 4.2 Result for transition test – LHCA

Then the sub-state vectors are changed to those from the middle adjacent 5 bits of the original patterns (bits S_3 to S_7). The results of the transition test show that in the 10,000 sub-state vectors, there are 128 transitions.

Now the sub-state vectors are changed again to those from the 2nd, 4th, 6th, … , and $2k$ th bits of the original patterns (bits S_2, S_4, S_6, S_8, and S_{10}). This time the LHCA achieves 1,023 out of 1,024 transitions.

For the Geffe generator, the three LFSRs are chosen as follows. LFSR1: with the characteristic polynomial $x^{10}+x^3+1$; LFSR2: with the characteristic polynomial $x^{11}+x^2+1$; LFSR3: with the characteristic polynomial x^9+x^4+1. From a random initial state (LFSR1: '0110011010'; LFSR2: '10001001000'; LFSR3: '101011001'), the Geffe generator generates 10,000 patterns. Similarly, if the transitions of the sub-state

vectors from the first five adjacent bits of the original patterns (bits S_1 to S_5 in Figure 4.3) are counted, the result is 1,020 transitions. If the transitions of the sub-state vectors from the middle five adjacent bits (bits S_3 to S_7) are counted, the number is 1,014. If the sub-state vectors come from the 2^{nd}, 4^{th}, 6^{th}, … , and $2k^{th}$ bits of the original patterns (bits S_2, S_4, S_6, S_8, and S_{10}), there are 1,024 transitions, which means that the Geffe generator achieves 100% of the possible transitions.

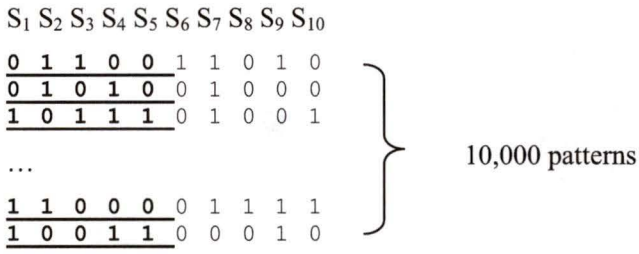


Figure 4.3 Result for transition test – Geffe

n	$k=\lfloor n/2 \rfloor$	Number of Transitions								
		A			B			C		
		LFSR	LHCA	GEFFE	LFSR	LHCA	GEFFE	LFSR	LHCA	GEFFE
10	5	64	64	1020	64	128	1014	1023	1023	1024

- A: the sub-state vectors come from the first k adjacent bits of the original patterns
- B: the sub-state vectors come from the middle k adjacent bits of the original patterns
- C: the sub-state vectors come from the 2^{nd}, 4^{th}, 6^{th}, … , $2k^{th}$ bits of the original patterns

Table 4.3 Result of the transition test for n =10

Table 4.3 shows the experimental result for the transition test when n=10.

Similarly, experimental results are derived for values of n between 10 and 20. All the results are given in Table 4.4.

n	$k=\lfloor n/2 \rfloor$	Possible Transi-tions	Number of Transitions								
			A			B			C		
			LFSR	LHCA	GEFFE	LFSR	LHCA	GEFFE	LFSR	LHCA	GEFFE
10	5	1024	64	64	1020	64	128	1014	1023	1023	1024
11	5	1024	64	64	1012	64	128	1021	1024	1024	1024
12	6	4096	128	128	2917	256	256	3004	4095	4095	3809
13	6	4096	128	128	2952	128	256	2952	4096	4096	3774
14	7	9999	256	256	4501	256	512	4533	9999	9999	7654
15	7	9999	256	256	4583	256	512	4678	8507	8492	7660
16	8	9999	512	512	5524	1024	1024	5728	9999	9999	9380
17	8	9999	512	512	5512	512	1024	5745	9615	9623	9302
18	9	9999	1024	1024	6372	2035	2036	7301	9999	9999	9845
19	9	9999	1024	1024	6383	2030	2034	6671	9999	9906	9821
20	10	9999	2042	2042	7381	2042	3735	6882	9999	9999	9963

- A: the sub-state vectors come from the first k adjacent bits of the original patterns
- B: the sub-state vectors come from the middle k adjacent bits of the original patterns
- C: the sub-state vectors come from the 2^{nd}, 4^{th}, 6^{th}, … , $2k^{th}$ bits of the original patterns

Table 4.4 Results of the transition test – number of transitions

The number of possible transitions in the 3^{rd} column of Table 4.4 is obtained as follows. The maximum number of transitions that a set of k-cell sub-state vectors can have is 2^{2k}. When $2^{2k}>9999$, the maximum number of transitions is set to be 9999, since 10,000 patterns are used in the experiment. Table 4.5 shows the percentage of transitions obtained by each generator in different tests.

n	$k=\lfloor n/2 \rfloor$	Percentage of Transitions Obtained (%)								
		A			B			C		
		LFSR	LHCA	GEFFE	LFSR	LHCA	GEFFE	LFSR	LHCA	GEFFE
10	5	6.25	6.25	99.61	6.25	12.5	99.02	99.90	99.90	100
11	5	6.25	6.25	98.83	6.25	12.5	99.71	100	100	100
12	6	3.13	3.13	71.22	2.56	2.56	73.34	99.98	99.98	92.99
13	6	3.13	3.13	72.07	3.13	2.56	72.07	100	100	92.14
14	7	2.56	2.56	45.01	2.56	5.12	45.33	100	100	76.55
15	7	2.56	2.56	45.83	2.56	5.12	46.78	85.08	84.93	76.61
16	8	5.12	5.12	55.25	10.24	10.24	52.79	100	100	93.81
17	8	5.12	5.12	55.13	5.12	10.24	57.46	96.16	96.24	93.03
18	9	10.24	10.24	63.73	20.35	20.36	73.02	100	100	98.46
19	9	10.24	10.24	63.84	20.30	20.34	66.72	100	99.07	98.22
20	10	20.42	20.42	73.82	20.42	37.35	68.83	100	100	99.64

- A: the sub-state vectors come from the first k adjacent bits of the original patterns
- B: the sub-state vectors come from the middle k adjacent bits of the original patterns
- C: the sub-state vectors come from the 2^{nd}, 4^{th}, 6^{th}, … , $2k^{th}$ bits of the original patterns

Table 4.5 Results of the transition test – percentage of transitions

Analysis and Conclusion

The results of the transition test show that in almost all cases the sub-state vectors generated by the Geffe generator have considerably more transitions than those generated by the LFSR or LHCA. For example, in both cases A and B, when 5-bit sub-state vectors are chosen from the 10-bit original patterns, the LFSR has 64 sub-state transitions, which account for only 6.25% of the total transitions. The LHCA has 64 (6.25%) and 128 (12.5%) transitions in case A and case B respectively. But for the Geffe generator, there are 1,020 (case A) and 1,014 (case B) transitions, both achieving more than 99% of the total transitions. When the value of n changes, the results are similar. In the following a theoretical explanation is given.

As discussed in 3.3.3, according to the parameters specified, the periods of the three generators are listed below, where T denotes the period and LCM denotes Least Common Multiple.

$$T_{LFSR} = 2^n - 1$$

$$T_{LHCA} = 2^n - 1$$

$$T_{Geffe} = LCM[(2^n - 1)(2^{n+1} - 1)(2^9 - 1)]$$

The maximum number of two-pattern transitions obtained by each generator is the same as their period. Clearly, $T_{Geffe} >> T_{LFSR} = T_{LHCA}$. For instance, when n =10, $T_{LFSR} = T_{LHCA} = 1023$, while T_{Geffe} can be larger than 1.07×10^9 . Although within one period, a specific two-pattern transition can appear more than once, which makes the actual number of transitions less than T_{Geffe} , the patterns generated by the Geffe generator still offer a much larger variety of transitions.

The LFSR and the LHCA have the same period thus the same number of two-pattern transitions. However, when sub-state vectors are concerned, LHCA can produce more sub-state transitions than the LFSR. Research in [28] shows that the LHCA have a much higher transition space than the LFSRs. Our experimental results (Table 4.4, case B) also support this.

In summary, both the theoretical analysis and the experimental results show that the non-linear machine (the Geffe generator) has a much richer transition capability than the linear machines (the LFSR and LHCA). And since the number of transitions determines the performance of a sequence of patterns when they are used to test sequential circuits, we hope that higher fault coverage would result if the Geffe generator rather than the LFSR or LHCA is used as the PRPG to detect single stuck-at faults in sequential circuits.

4.3 Fault Simulation

In this section the results of fault simulation are presented. The experiments are conducted according to the specifications given in 3.3.4.

Since the fault simulation includes two parts, namely the simulation of the combinational circuits, and the simulation of the sequential circuits, we analyze the results for these two parts separately.

4.3.1 Simulation of the Combinational Circuits (single stuck-at faults)

Simulation of the combinational circuits uses FSIM as the simulator, tests the ISCAS'85 benchmark circuits with patterns generated by the three different kinds of Pseudo Random Pattern Generators, and compares the performance among them.

As specified in 3.3.4, simulation of the combinational circuits includes three parts: non-linear machine versus linear machines, simulation of a small circuit, and simulation of similar circuits. In the following, the results for these three parts are given and analyzed separately.

Non-linear Machine Versus Linear Machines

Since the results for the circuit c432 are typical of the whole set, we discuss it in complete detail. Results for other ISCAS'85 benchmark circuits are given in Appendix 4.

C432 is one of the eleven ISCAS'85 benchmark circuits. It has 36 input lines and contains 524 single stuck-at faults, 520 of which are detectable.

The LFSR with a characteristic polynomial $x^{36}+x^{11}+1$ is chosen as the generator. Starting from a random initial state: '000100100110101000100100010101100110', it generates 1,000 patterns. These patterns are applied as inputs to stimulate the circuit, and

the fault coverage at the points of 1, 2, 5, 10, 20, 30, 40, 50, 100, 200, 300, 400, 500, and 1,000 patterns respectively (see Table 4.6) is recorded.

Number of patterns	Fault Coverage (%)	Number of patterns	Fault Coverage (%)
1	5.5	50	80.7
2	14.5	100	89.5
5	32. 1	200	94.3
10	46.0	300	95.4
20	68.1	400	96.0
30	73.5	500	96.0
40	78.8	1000	98.9

*IS1: Initial State of the generator: '000100100110101000100100010101100110'.

Table 4.6 Results for c432 – LFSR – IS1*

Similarly, from nine other different initial states (listed in Table 4.7), we can get nine sets of such results, and the ten sets of results are shown together in Table 4.8:

#	Initial State	#	Initial State
1	000100100110101000100100010101100110	6	001110101011011001101110110010001110
2	100011010011110100101000011111110100	7	110100000111000010010101001001001010
3	100010100011110010001100010010100111	8	110000011101100111110110100001010111
4	011111001110101110100010100000011000	9	110000100100101000010001111000011110
5	000110000111000111111110010001000110	10	010110100101000110001100110001111110

Table 4.7 Initial states – LFSR

# of patterns	Fault Coverage (%)									
	1	2	3	4	5	6	7	8	9	10
1	5.5	6.3	7.4	13.6	6.3	10.3	13.4	9.2	8.8	5.2
2	14.5	13.2	9.9	21. 4	11.1	15.8	18.1	15.3	14.9	7.3
5	32. 1	33.6	34.4	35.9	30.3	37.6	30.2	46. 6	27.9	31.9
10	46.0	46.2	48.9	55.7	42.9	53.6	47.5	60.9	45.4	46. 4
20	68.1	62.0	69.8	65.5	58.4	70.4	66.8	73.3	62.6	69.1
30	73.5	72.7	80.9	78.4	68.9	77.3	69.8	81.5	71.8	76.5
40	78.8	80.9	82.6	84.2	76.7	80.5	76.9	83.2	80.3	80.3
50	80.7	81.7	84.4	86.8	81.7	84.7	82. 1	84.9	82.4	85. 7
100	89.5	86.1	89.1	92.2	90.8	90.3	90.5	90.8	88.9	92.9
200	94.3	96.0	93.3	93.7	92.7	95.6	95.4	93.3	96.2	95.2
300	95.4	97.7	98.3	96. 4	93.3	96.0	95.8	96. 4	97.1	98.1
400	96.0	97.7	98.5	97.3	96.0	96. 4	96.9	96.8	97.9	98.3
500	96.0	97.7	99.0	97.5	96.2	96. 4	97.7	96.9	98.1	98.3
1000	98.9	98.9	99.2	98.5	99.0	97.9	99.2	99.2	98.7	99.2

Table 4.8 Results for c432 – LFSR

The average, median and maximum values of each row in Table 4.8 are computed and shown in Table 4.9.

In the same way, results for LHCA and Geffe can be obtained. We choose the LHCA with the same characteristic polynomial as LFSR: $x^{36}+x^{11}+1$. It has the structure of "010010010001100001011111000010010010". From the same ten initial states as those of the LFSR, it generates 1,000 patterns respectively. Each set of patterns is used to stimulate the circuit c432, the fault coverage at the points of 1, 2, 5, 10, 20, 30, 40, 50, 100, 200, 300, 400, 500, and 1,000 patterns is recorded (shown in Table 4.10).

# of patterns	Fault Coverage (%)			# of patterns	Fault Coverage (%)		
	Average	Median	Max		Average	Median	Max
1	8.6	8.1	13.6	50	83.5	83.4	86.8
2	14.1	14.7	21. 4	100	90.1	90.4	92.9
5	34.0	32.8	46.6	200	94.56	94.8	96.2
10	49.4	46.9	60.9	300	96.5	96. 4	98.3
20	66.6	67.5	73.3	400	97.2	97.1	98.5
30	75.1	75	81.5	500	97.4	97.6	99.0
40	80.5	80.4	84.2	1000	98.9	99.0	99.2

Table 4.9 Summary results for c432 – LFSR

# of patterns	Fault Coverage (%)									
	1	2	3	4	5	6	7	8	9	10
1	6.7	14.1	12.0	4.0	11.3	8.0	8.6	10.1	5.7	8.8
2	12.4	21. 4	16.2	11. 5	12.8	11.1	14.7	21.9	10. 7	13.7
5	29.0	32.8	29.4	31.3	32.8	30.0	32.8	36.3	25.8	26.5
10	47.3	48.1	44.3	44.3	45.4	44.3	40.1	46.6	53.4	45.0
20	65.1	64.7	65.6	69.7	61. 5	65.1	66.2	61.1	68.5	59.5
30	76.9	72.9	73.3	77. 7	73.9	78.2	71.2	71.0	77.9	74.8
40	80.5	78.8	76.0	83.4	80. 4	84.5	79.2	78.4	80.7	78.4
50	82. 1	82. 1	82.4	87.2	85.5	88.0	83.8	84.5	83.0	83.4
100	89.1	88.9	90.8	91.0	90.6	91.4	90.5	91.8	91.2	92.0
200	94.1	93.7	94.5	94.1	95.0	94.5	95.3	94.3	95.8	96.6
300	96.8	97.3	96.8	95.4	96. 4	97.5	96.2	96.0	98.1	96.9
400	97.9	97.7	97.5	95.8	96.6	97.5	96. 4	97.7	98.9	97.1
500	98.7	98.3	97.5	98.7	96.8	97.7	98.1	98.1	99.2	97.1
1000	99.2	99.2	98.9	99.2	98.9	99.0	99.0	99.2	99.2	98.3

Table 4.10 Results for c432 – LHCA

Finally, the three LFSRs in the Geffe generator are chosen as follows: LFSR1: $x^{36}+x^{11}+1$; LFSR2: $x^{37}+x^{12}+x^{10}+x^{2}+1$; LFSR3: $x^{9}+x^{4}+1$.

From ten different sets of initial states (the initial states of LFSR1 are the same as those listed in Table 4.7; the initial states of LFSR2 and LFSR3 are listed in Table 4.11), the Geffe generator produces 10 sets of patterns (1,000 patterns for each set).

	Initial States for LFSR2	Initial States for LFSR3
1	01000110011111011101001010100101111111	000101000
2	10000111111101111101010100000000110001	010110000
3	00010100010111010111001111100010000010	001110000
4	11101011100110101000011010000011010100	110100111
5	00100001000111000010100001111001011110	011110111
6	11100000100010010001001100100000001101	000011110
7	11001010001010011010110110001111011110	110001100
8	10101110001111001010110001111100101000	010101001
9	10000100011000011110100001001101101010	101000011
10	00000100001111000001110110100111010001	001000111

Table 4.11 Initial states – Geffe

Each set of patterns is applied to stimulate the circuit c432. The fault coverage at the points of 1, 2, 5, 10, 20, 30, 40, 50, 100, 200, 300, 400, 500, and 1,000 patterns is recorded (see Table 4.12).

# of patterns	Fault Coverage (%)									
	1	2	3	4	5	6	7	8	9	10
1	8.6	13.9	10.1	4.0	14.1	11.8	8.6	8.0	5.7	6.5
2	13.9	21.0	17.4	12.8	18.5	17.2	17.0	16.6	13.6	17.9
5	26.0	34.7	25.4	30.7	36.4	31.1	27.3	25.4	26.3	30.7
10	48.7	46.9	37.2	48.9	52.9	50.8	49.2	52.7	43.9	43.7
20	61.8	62.0	49.2	68.3	67.9	68.3	69.5	68.1	70.4	67.9
30	71.8	67.6	67.0	76.5	75	77.5	71.8	78.1	77.3	79.4
40	79.6	77.1	74.8	82. 1	79.6	80.7	77.1	84.2	81.3	84.7
50	82. 1	80.3	81.5	83.8	81.5	83.8	81.5	84.4	83.6	86.6
100	88.9	89.7	90.5	91.4	89.9	91.2	88.7	90.3	92.9	92.9
200	94.7	93.3	96.6	96.6	92.2	94.7	93.5	94.3	95.0	94.8
300	96.6	96.8	98.7	98.1	94.3	96.6	96.9	95.2	96.0	96.2
400	97.7	96.9	98.7	98.5	94.8	98.1	96.9	97.3	96.6	96.8
500	98.3	96.9	98.9	98.7	96.2	98.3	98.1	98.1	96.8	96.8
1000	99.2	97.1	99.0	99.2	97.9	99.0	99.0	99.0	98.9	99.0

Table 4.12 Results for c432 – Geffe

The average, median and maximum values of each row in Table 4.10 and Table 4.12 are computed and put in Table 4.13, together with those from LFSR (Table 4.9).

Based on the results in Table 4.13, the fault coverage – number of patterns relationship curves can be drawn, as shown in Figure 4.4 and Figure 4.5. In a similar manner, the simulation results for the other ISCAS'85 benchmark circuits can also be obtained (all the detailed results are included in Appendix 4).

# of patterns	Fault Coverage (%)								
	LFSR			LHCA			Geffe		
	Average	Median	Max	Average	Median	Max	Average	Median	Max
1	8.6	8.1	13.6	8.9	8.7	14.1	9.1	8.6	14.1
2	14.1	14.7	21. 4	14.6	13. 3	21.9	16.6	17.1	21.0
5	34.0	32.8	46.6	30.7	30.6	36.3	29.4	29.0	36. 5
10	49.4	46.9	60.9	45.8	45.0	53.4	47.5	48.8	52.9
20	66.6	67.5	73.3	64.7	65.1	69.7	65.4	68.0	70.4
30	75.1	75.0	81.5	74.8	74.3	78.2	74.2	75.8	79.4
40	80.5	80.4	84.2	80.0	79.8	84.5	80.1	80.2	84.7
50	83.5	83.4	86.8	84.2	83.6	88.0	82.9	82.8	86.6
100	90.1	90.4	92.9	90.7	90.9	92.0	90.6	90.4	92.9
200	94.6	94.8	96.2	94.8	94.5	96.6	94.6	94.7	96.6
300	96.5	96. 4	98.3	96.7	96.8	98.1	96.5	96.6	98.7
400	97.2	97.1	98.5	97.3	97.5	98.9	97.2	97.1	98.7
500	97.4	97.6	99.0	98.0	98.1	99.2	97.7	98.1	98.9
1000	98.9	99.0	99.2	99.0	99.1	99.2	98.8	99.0	99.2

Table 4.13 Results for c432

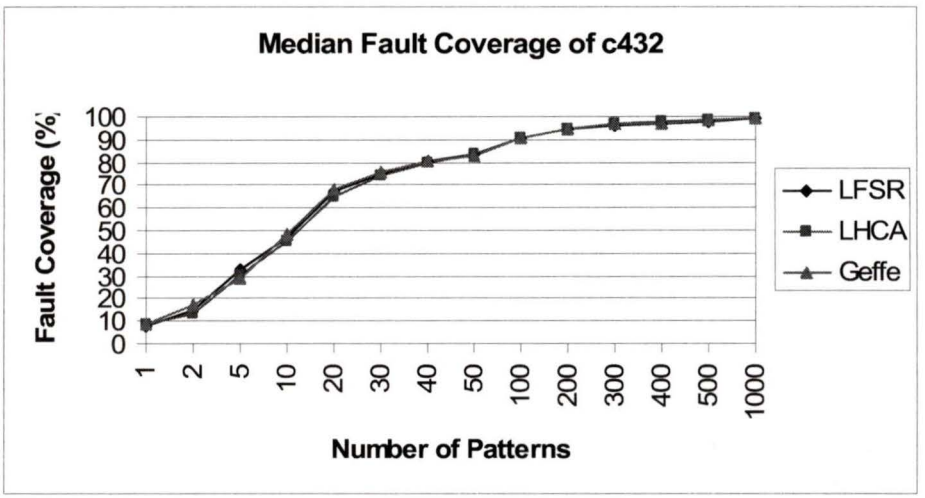


Figure 4.4 Median fault coverage of c432

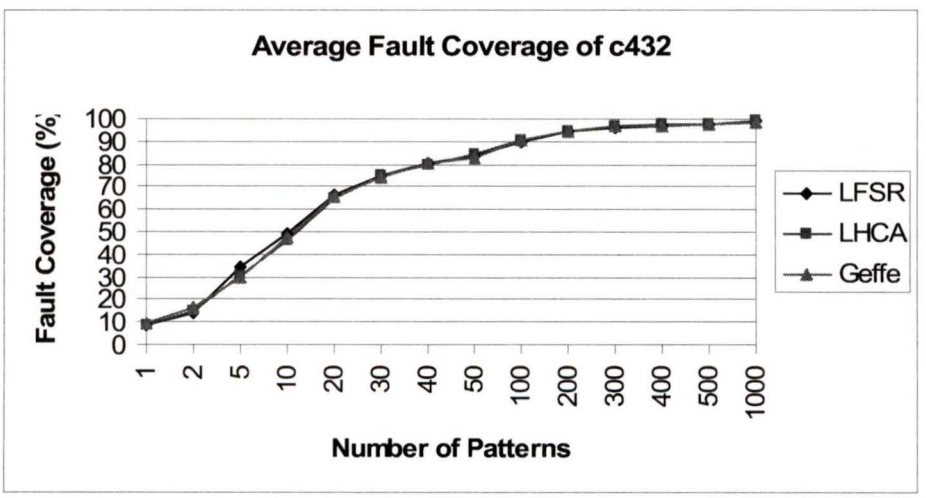


Figure 4.5 Average fault coverage of c432

Analysis and conclusion

The graphs in Appendix 4 show that the non-linear machine (the Geffe generator) does not exhibit any obvious improvement in performance over the linear machines when being used as a random pattern generator to test combinational circuits. Although for some circuits (e.g., c432, c1355), when a small number of patterns are applied, there is difference among the results (fault coverage) of different generators. But when the number of patterns increases, the fault coverage becomes almost identical, irrespective of the generator.

This confirms the results of previous research and we can explain this phenomenon. Combinational circuits are those whose output strictly depends on the present value of the inputs. For a single stuck-at fault in the combinational circuits, one pattern is sufficient to detect it. Generally, the more patterns (which are different from each other) applied to the combinational circuits, the higher fault coverage we would get. For the LFSR and LHCA, within one cycle, every pattern is different from any other. But for the Geffe generator, this might not be true since the non-linear function may produce duplicated patterns. However, the number, and likelihood, of such duplications are both very small, and assuming that patterns generated by both linear and non-linear machines have a reasonably good pseudo random distribution, we do not expect that using the non-linear machine as PRPG to stimulate the combinational circuits would lead to higher fault coverage. The experimental results also support it.

But why is there some diversity when a small number of patterns are used? The explanation is that there exist some 'powerful' patterns, which are capable of detecting more faults than other patterns (a detailed analysis is given below). So when a small number of patterns are used, some sets of patterns can show superiority to other sets. But this cannot be counted as a superior behavior of the test pattern generator itself.

Simulation of a Small Circuit

In the previous section, it has been shown that when a small number of patterns are applied to the benchmark circuits, there is some diversity amongst the fault coverage when using different initial states (see result in Table 4.8). For example, when only one single pattern was applied, the fault coverage ranged from 5.2% to 13.6%. Why are some patterns so powerful while others not? In this section we analyze the structure of a small circuit – c17, to consider the relationship between the structure of a circuit and the 'power' of input patterns.

C17 is a small circuit in the ISCAS'85 benchmark circuits package. It has 5 inputs, 2 outputs, and 22 single stuck-at faults. The structure of the circuit c17 is shown in Figure 4.6.

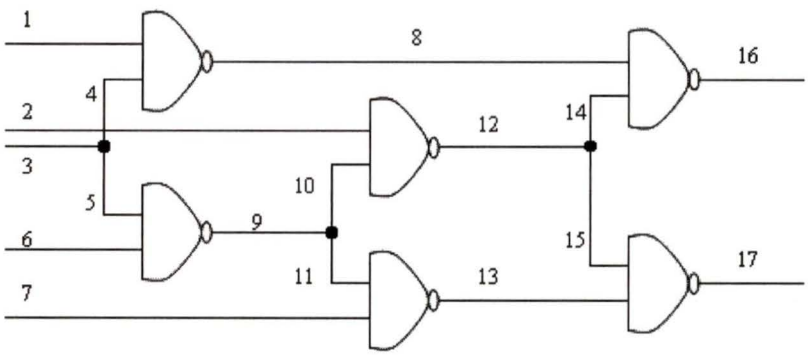


Figure 4.6 Structure of c17

First, for each single stuck-at fault in the circuit, we find out all the patterns which can detect it based on the analysis of the structure of c17.

For example, for the fault line 1 s-a-1, to detect it, the following conditions are needed: Value(line 1)=0 and Value(line 4)=1 and Value(line 14)=1. By analyzing further, we can

get the patterns which can detect this fault: 001XX and 0111X, where X stands for 'don't care value', which could be logic 0 or logic 1. The patterns that can detect the other faults can be obtained the same way (see Table 4.14).

Fault	Patterns which can detect it	# of patterns that can detect it	Fault	Patterns which can detect it	# of patterns that can detect it
1: s-a-1	001XX 0111X	6	10: s-a-1	X111X	4
2: s-a-1	X00XX 0010X 10100	11	11: s-a-1	XX111	4
3: s-a-0	101XX 1111X X111X XX111	9	12: s-a-0	XX11X X00XX 00XXX X0XX0	19
3: s-a-1	100XX X101X XX011	9	12: s-a-1	X10XX 11100 0110X	11
4: s-a-1	100XX	4	13: s-a-1	X10XX 0110X	10
5: s-a-1	X101X X0011	6	14: s-a-1	X10X0 X1100	6
6: s-a-1	X110X X0101	6	15: s-a-1	X00X1 X0101	6
7: s-a-1	X00X0 X0100	6	16: s-a-0	1X1XX X10XX X11X0	18
8: s-a-1	101XX 1111X	6	16: s-a-1	X00XX 0111X 001XX	14
9: s-a-0	X10XX X00X1 X110X X0101	18	17: s-a-0	X10XX X110X XX0X1 XX101	18
9: s-a-1	X111X X0111	6	17: s-a-1	X111X X0XX0 X011X	14

Table 4.14 Test patterns for c17

Then, with the simulator, all possible patterns are applied to the circuit. The resulting fault coverage is given in Table 4.15.

Pattern	Number of detected faults	Faults covered (%)	Pattern	Number of detected faults	Faults covered (%)
10000	7	31.8	11111	8	36. 4
01001	5	22.7	10111	8	36. 4
10010	7	31.8	00111	7	31.8
01101	6	27. 3	01110	7	31.8
11010	8	36. 4	11100	7	31.8
11101	8	36. 4	10001	6	27. 3
10011	4	18. 2	01011	8	36. 4
01111	9	40.9	10110	5	22.7
11110	8	36. 4	00101	8	36. 4
10101	7	31.8	01010	8	36. 4
00011	7	31.8	10100	8	36. 4
00110	8	36. 4	00001	7	31.8
01100	4	18. 2	00010	6	27. 3
11000	7	31.8	00100	5	22.7
11001	6	27. 3	01000	6	27. 3
11011	7	31.8	00000	5	22.7

Table 4.15 Fault coverage of individual test vector

First we check the results in Table 4.14 and 4.15 and see whether they are consistent with each other. For example, Table 4.14 shows that the pattern '11111' can detect 8 single stuck-at faults, which are: 3: s-a-0, 8: s-a-1, 8: s-a-1, 10: s-a-1, 11: s-a-1, 12: s-a-0, 16: s-a-0, 17: s-a-1. This is just the same as the result in Table 4.15. This consistency is a useful verification of the simulators.

The results in Table 4.15 show that some patterns do exhibit better performance than the others. For example, the pattern '01111' can detect 9 single stuck-at faults (covering 40.9% of the total faults) in the circuits. But for some other patterns, like '10011' and '01100', either of them can only detect 4 single stuck-at faults (covering 18. 2% of the faults). So it is the structure of the circuit that determines the 'power' of input patterns. Obviously, the inclusion of a pattern like '01111' in the sequence will lead to higher

initial fault coverage. This explains why we get various values of fault-coverage for different generators when only a small number of patterns are applied to the benchmark circuits. Contrarily, results in Table 4.14 show that some faults are easy to detect while some are hard to detect. For example, the single stuck-at fault 12: s-a-0 can be detected by 19 patterns, however, the faults, 4: s-a-1, 10: s-a-1, and 11: s-a-1, can only be detected by 4 patterns.

Simulation of Similar Circuits

In this section we group the results according to the circuits' function and see whether similar circuits have similar results.

Among the ISCAS'85 benchmark circuits (Table 3.2), three circuits (c499, c1355, and c1908) have the function of ECAT (Error Correction And Translation). Five circuits (c880, c2670, c3540, c5315, c7552) are ALU (Arithmetic Logic Unit) and Control. The results of these two groups are analyzed separately below.

ECAT

First the results of the circuits c499 and c1355 are compared since they are functionally equivalent [1, page 108], except that all XOR gates of c499 have been expanded into their 4-NAND gate equivalents in c1355. The results (in Appendix 4) show that the curves for c499 have almost the same trends as those of the curves for c1355.

Now we compare the result for circuit c1908 with those for c499 and c1355. Obviously, the figure for c1908 does not resemble the other two.

ALU and Control

For the five circuits (c880, c2670, c3540, c5315, c7552) which have the same function of ALU and Control, the results (see Appendix 4) show that the five graphs exhibit similar behavior. Also, for each figure, diversity among the curves is not obvious.

Conclusion

From the results above we notice that if a circuit *a* is transformed to its functionally equivalent circuit *b* by expanding all the XOR gates into their 4-NAND gate equivalents, then when being simulated, the two circuits show very similar properties. However, for circuits which have similar function, it is not always true.

4.3.2 Simulation of the Sequential Circuits (single stuck-at faults)

Simulation of the sequential circuits uses HOPE as the simulator, tests the ISCAS'89 benchmark circuits with patterns generated by the three PRPGs and compares the performance among them. The experiments include two parts: comparison of the performance among linear and non-linear machines (non-linear machine versus linear machines), and investigation of the effect of different initial circuits value on the fault coverage. In the following the results for these two parts are presented and analyzed separately.

Non-linear Machine Versus Linear Machines as Generators

First, we look at the circuit s1488 in detail and see how the results are obtained. Complete results for the other ISCAS'89 benchmark circuits are shown in Appendix 5.

S1488 is one of the 31 ISCAS'89 benchmark circuits. It has 8 primary inputs, 19 primary outputs, 6 D-type flipflops, 653 AND/OR/NOT gates, and 1486 collapsed faults.

The LFSR with a characteristic polynomial $x^8 + x^6 + x^5 + x + 1$ is chosen as the pattern generator. Starting from a random initial state: '01100110', it generates 20,000 patterns. In the initial experiments, we leave the flip-flops in the circuit at the default initial value, the unknown state U, and apply the patterns as input to stimulate the circuit, and record the fault coverage at the points of 64, 1,000, 10,000, and 20,000 patterns respectively (see Table 4.16).

Number of Patterns	64	1000	10000	20000
Fault Coverage (%)	32.5	32.8	32.8	32.8

*IS1: Initial State of the generator: '01100110'.

Table 4.16 Result for s1488 – LFSR – IS1*

From ten different initial states (listed in Table 4.17), we can get ten sets of such results, shown in Table 4.18.

#	Initial State	#	Initial State	#	Initial State	#	Initial State
1	01100110	4	10100111	7	01001010	10	01111110
2	11110100	5	00011000	8	01010111		
3	01000110	6	10001110	9	00011110		

Table 4.17 Initial states – LFSR

Initial State	Number of Patterns			
	64	1000	10000	20000
1	32.5	32.8	32.8	32.8
2	32.0	32.8	32.8	32.8
3	32.1	32.8	32.8	32.8
4	32.6	32.8	32.8	32.8
5	31.8	32.8	32.8	32.8
6	32.4	32.8	32.8	32.8
7	32.2	32.8	32.8	32.8
8	32.1	32.8	32.8	32.8
9	32.5	32.8	32.8	32.8
10	31.8	32.8	32.8	32.8

Table 4.18 Fault coverage (%) for s1488 – LFSR

The median and maximum values of each column in Table 4.18 are computed and put in Table 4.19.

Number of Patterns		64	1000	10000	20000
Fault Coverage (%)	Median	32.5	32.8	32.8	32.8
	Max	32.0	32.8	32.8	32.8

Table 4.19 Summary results for s1488 – LFSR

Similarly, the results for LHCA and Geffe can be obtained.

We choose the LHCA that has the same characteristic polynomial as the LFSR: $x^8+x^6+x^5+x+1$. It has the structure of "01001011". From the same initial states as those of the LFSR, it generates 20,000 patterns respectively. Each set of patterns is used to stimulate the circuit s1488, and the results are shown in Table 4.20.

Initial State	Number of Patterns			
	64	1000	10000	20000
1	37.8	40.8	40.8	40.8
2	38.6	40.8	40.8	40.8
3	36.0	40.8	40.8	40.8
4	38.6	40.8	40.8	40.8
5	36.7	40.8	40.8	40.8
6	36.1	40.8	40.8	40.8
7	33.1	40.8	40.8	40.8
8	36.0	40.8	40.8	40.8
9	38.9	40.8	40.8	40.8
10	35.5	40.8	40.8	40.8

Table 4.20 Fault coverage (%) for s1488 – LHCA

For the Geffe generator, the three LFSRs are chosen as follows: LFSR1: $x^8+x^6+x^5+x+1$; LFSR2: x^9+x^4+1; LFSR3: x^9+x^4+1.

From ten different initial states (for LFSR1, the initial states are the same as those listed in Table 4.17; for LFSR2 and LFSR3, their initial states are listed in Table 4.21), it generates 10 sets of patterns (20,000 patterns for each set).

	Initial States for LFSR2	Initial States for LFSR3
1	101000100	100010101
2	111000010	100101111
3	001001111	111100011
4	001000110	001001111
5	000101000	101110101
6	001101110	110011011
7	010010101	001000011
8	000101101	111100110
9	011110001	000010100
10	001100110	001100010

Table 4.21 Initial states – Geffe

Each set of patterns is applied as input to stimulate the circuit. The fault coverage is recorded at the points of 64, 1,000, 10,000 and 20,000 patterns respectively (shown in Table 4.22).

The median and maximal values of each column in Table 4.20 and Table 4.22 are computed and put in Table 4.23, together with those from the LFSR.

Initial State	Number of Patterns			
	64	1000	10000	20000
1	39.6	64.0	70.8	75.2
2	33.6	52.0	68.1	73.2
3	37.0	58.2	69.9	71.9
4	36.9	54.6	66.6	72.7
5	35.9	63.7	67.9	68.9
6	35.1	54.0	72.9	73.6
7	35.9	52.8	70.1	72.0
8	42.7	57.8	71.2	73.7
9	40.3	52.4	71.7	75.7
10	43.0	55.0	72.1	72.7

Table 4.22 Fault coverage for s1488 – Geffe

# of Patterns	Fault Coverage (%)					
	LFSR		LHCA		Geffe	
	Median	Max	Median	Max	Median	Max
64	32.1	32.6	36.4	38.9	37.0	43.0
1000	32.8	32.8	40.8	40.8	54.8	64.0
10000	32.8	32.8	40.8	40.8	70.5	72.9
20000	32.8	32.8	40.8	40.8	73.0	75.7

Table 4.23 Final results for s1488

From the results in Table 4.23, some fault coverage – number of patterns relationship curves can be drawn, as shown in Figure 4.7 and 4.8.

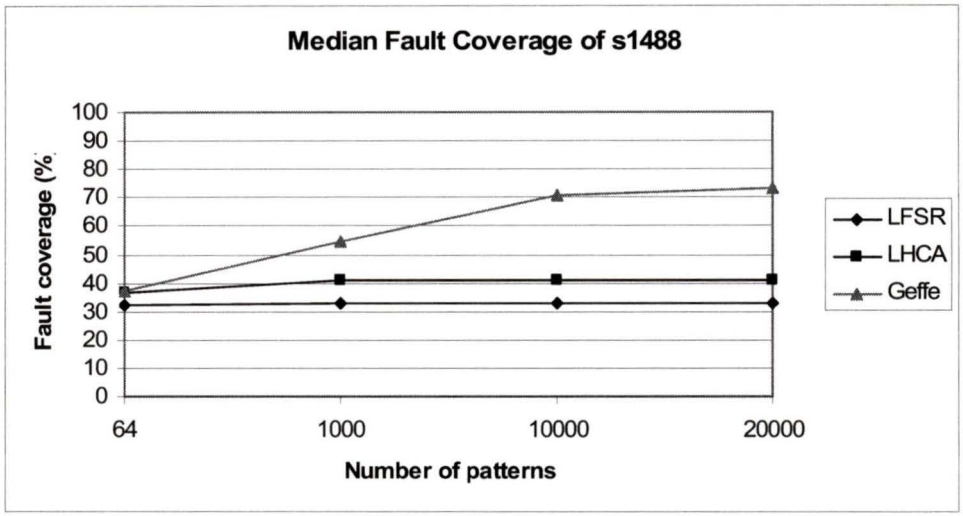


Figure 4.7 Median fault coverage of s1488

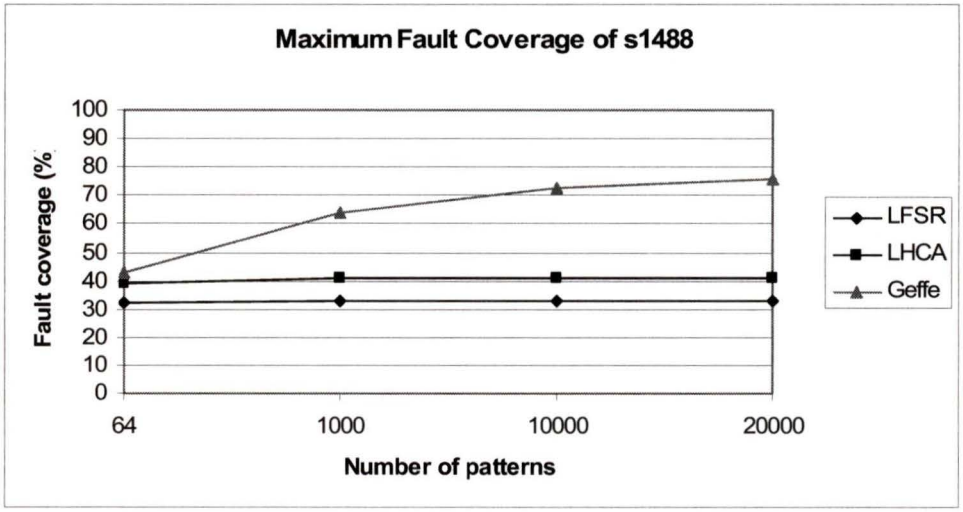


Figure 4.8 Maximum fault coverage of s1488

Similarly, the simulation results for the other ISCAS'89 benchmark circuits can be obtained. They are included in Appendix 5.

Table 4.24 lists the fault coverage when 20,000 patterns are used for all the ISCAS'89 benchmark circuits. Data in the last row of the table are the arithmetic mean of each column, which show the overall performance of the generators on the whole set of ISCAS'89 benchmark circuits.

Circuit	Fault Coverage (%)					
	LFSR		LHCA		Geffe	
	Median	Max	Median	Max	Median	Max
s27	78.1	78.1	87.5	87.5	**100**	**100**
s208	27.0	27.0	22.8	22.8	**41.2**	**42.3**
s298	20.8	20.8	**51.3**	51.3	50	**83.8**
s344	73.4	73.4	58.8	58.8	**96.2**	**96.2**
s349	73.4	73.4	59.1	59.1	**95.7**	**95.7**
s382	11.8	11.8	**13.3**	13.3	12.3	**17.0**
s386	44.5	44.5	55.2	55.2	**65.6**	**66.7**
s400	11.3	11.3	**13.4**	13.4	12.0	**17.0**
s420	22.6	22.6	**33.3**	**33.5**	30.5	32.3
s444	10.3	10.3	11.0	11.0	**11.2**	**16.2**
s510	0.0	0.0	0.0	0.0	0.0	0.0
s526	8.5	8.5	8.5	8.5	**8.6**	**12.3**
s526n	8.5	8.5	8.5	8.5	**8.7**	**12.3**
s641	86.1	86.3	86.3	86.3	**86.4**	**86.5**
s713	81.6	81.8	81.8	81.8	81.8	**81.9**
s820	40.2	40.2	42.4	42.6	**45.5**	**46.5**
s832	39.3	39.3	41.3	41.5	**44.4**	**45.5**
s838	20.2	20.2	**25.2**	**25.7**	23.8	24.0

Circuit	Fault Coverage (%)					
	LFSR		LHCA		Geffe	
	Median	Max	Median	Max	Median	Max
s953	8.3	8.3	8.3	8.3	8.3	8.3
s1196	93.6	93.6	96.1	96.1	**97.5**	**98.1**
s1238	88.4	88.4	91.2	91.2	**92.4**	**92.8**
s1423	32.2	33.9	55.3	57.8	**58.7**	**62.2**
s1488	32.8	32.8	40.8	40.8	**73.0**	**75.7**
s1494	32.1	32.1	40.0	40.0	**72.1**	**74.8**
s5378	65.5	65.5	66.4	**69.4**	66.4	69.2
s9234	0.3	0.3	0.3	0.3	0.3	0.3
s13207	6.2	6.4	6.2	6.4	6.2	6.4
s15850	0.7	0.7	0.7	0.7	0.7	0.7
s35932	83.3	83.8	62.0	62.7	83.3	**84.7**
s38417	3.5	3.5	3.5	3.5	3.5	3.5
s38584	18.7	18.7	19.7	19.7	**21.1**	**21.6**
Arithmetic Mean	*36.2*	*36.3*	*38.4*	*38.6*	*45.1*	*47.6*

Table 4.24 Final results for ISCAS'89 circuits

The chart in Figure 4.9 shows the arithmetic mean of the median fault coverage of the 30 ISCAS'89 benchmark circuits (except the circuit s510) when they are simulated with patterns generated by different types of generators.

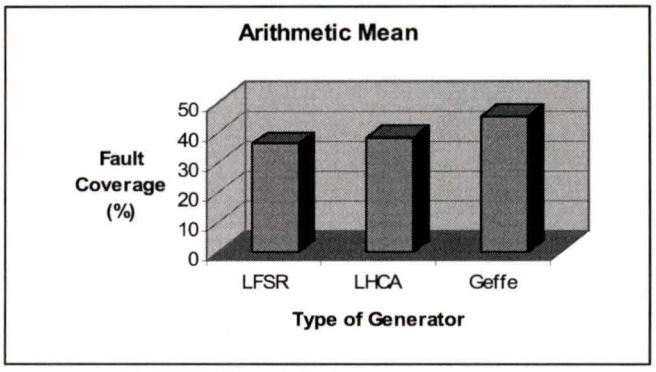


Figure 4.9 Arithmetic mean – ISCAS'89 benchmark circuits

Analysis and conclusion

The experimental results (e.g., Table 4.24, Figure 4.7, 4.8, and 4.9) show that when testing sequential circuits, the non-linear machine exhibits much better performance than the linear machines.

For instance, as shown in Figure 4.7 and 4.8, for both the LFSR and LHCA, after 1,000 patterns are applied to the circuit s1488, the fault coverage stays at the same point no matter how many more patterns are added. But for the Geffe generator, the fault coverage keeps on growing when the number of input patterns increases. When 20,000 patterns are used, the LFSR and the LHCA can achieve only 32.8% and 40.8% fault coverage respectively, while the Geffe generator can reach as high as 75.7% fault coverage (median value of fault coverage is 73.0%). Results in Table 4.24 also show that for most ISCAS'89 benchmark circuits, higher fault coverage is achieved when the Geffe generator is used as the PRPG.

This can be explained as follows for the circuits with no more than 14 inputs. Since an n-cell LFSR or LHCA with primitive polynomial has the period of $2^n - 1$, when the linear machines generate more than $2^n - 1$ vectors, they don't produce new patterns or transitions. And these repeated transitions make no extra contribution to the final fault

coverage. In this experiment each generator produces 20,000 patterns. So for the circuits with less than 15 inputs, after the first $2^n - 1$ patterns, the linear machines keep on generating redundant patterns. But the Geffe generator has a much longer period. It keeps on producing new patterns or transitions. This explains why for some circuits, when a number of patterns are applied, no matter how many more patterns generated by the linear machines are added, the fault coverage stays at the same point. But for the Geffe generator, the fault coverage keeps on growing when more patterns are produced.

Sequential circuits are those whose outputs depend not only on the present input value but also the past input value. It is more difficult to detect a single stuck-at fault in sequential circuits because we need a test sequence instead of a single test vector, which is enough to detect a single stuck-at fault in the combinational circuits. So actually it is the variety of pattern transitions rather than the number of patterns that determines the performance of a sequence of patterns. In [28] the authors provide a theoretical analysis and empirical comparisons to explain why the LHCA are better than the LFSRs as generators for sequential-type faults. They demonstrate that an n -cell LHCA with maximum length cycles has larger transition capability than an n -cell LFSR with maximum length cycles. Simulation of the ISCAS'85 benchmark circuits shows that when testing for delay faults, LHCA are better than LFSRs.

In section 3.2 and 4.2 the transition test is introduced and the experimental results of the transition test are given. The results show that patterns generated by the Geffe generator have more transitions than those generated by LFSR or LHCA. So it is not surprising that the Geffe generator performs better than LFSR and LHCA when testing single stuck-at faults in sequential circuits.

Initial Value of the Flip-flops in the Benchmark Circuits

In the former experiment, when HOPE was used to simulate the benchmark circuits, all the flip-flops in the circuits were initially set to the unknown state U (the default mode).

Technically, they can also be initialized to logic 0 or logic 1. Would different initial value of the circuits makes difference to the fault coverage?

In the following sets of experiments, using the same sets of patterns, we test the fault coverage of the ISCAS'89 benchmark circuits in which the flip-flops are initially set to logic 0. Instead of recording the fault coverage at 4 points (64, 1,000, 10,000, and 20,000 patterns), we only record the fault coverage at the point of 20,000 patterns.

We still look at the circuit s1488 in detail. For each generator (LFSR, LHCA, and Geffe), we have the same 10 sets of patterns (each contains 20,000 patterns) used for the previous experiments. Each set of patterns is used to stimulate the circuit in the same way as that in the previous experiments, except that the flip-flops in the circuit are initialized to logic 0. Table 4.25 shows the results.

Generator	Number of Patterns: 20,000									
	1	2	3	4	5	6	7	8	9	10
LFSR*	33.0	33.0	33.0	33.0	33.0	33.0	33.0	33.0	33.0	33.0
LHCA*	41.0	41.0	41.0	41.0	41.0	41.0	41.0	41.0	41.0	41.0
Geffe*	75.3	73.4	72.0	72.8	69.0	73.7	72.1	73.8	75.8	72.9

*: The flip-flops of the benchmark circuits are set to logic 0 initially.

Table 4.25 Fault coverage for s1488 – flip-flops set to logic 0

The median and maximum values of each row are computed and shown in Table 4.26.

# of Patterns	Fault Coverage (%)					
	LFSR*		LHCA*		Geffe*	
	Median	Max	Median	Max	Median	Max
20000	33.0	33.0	41.0	41.0	73.1	75.8

*: The flip-flops of the benchmark circuits are set to logic 0 initially.

Table 4.26 Final results for s1488 – flip-flops set to logic 0

Results of the other ISCAS'89 benchmark circuits are shown in Table 4.27.

Circuit	Fault Coverage (%)					
	LFSR*		LHCA*		Geffe*	
	Median	Max	Median	Max	Median	Max
s27	78.1	78.1	90.6	90.6	**100**	**100**
s208	32.6	32.6	28.4	28.4	**47.2**	**48.4**
s298	24.0	24.0	22.4	22.4	**53.6**	**86.4**
s344	77.5	77.5	66.4	66.4	**98.0**	**98.0**
s349	77.4	77.4	66.6	66.6	**97.4**	**97.4**
s382	13.0	13.0	13.0	13.0	**13.5**	**18.3**
s386	44.5	44.5	51.3	51.3	**65.6**	**66.7**
s400	12.5	12.5	12.5	12.5	**13.2**	**18.2**
s420	28.1	28.1	**39.1**	**39.1**	36.3	38.1
s444	11.4	11.4	11.6	11.6	**12.2**	**17.3**
s510	100	100	100	100	100	100
s526	9.7	9.7	9.7	9.7	**9.9**	**13.5**
s526n	9.8	9.8	9.8	9.8	**9.9**	**13.6**
s641	86.9	87.2	87.2	87.2	**87.3**	**87.4**
s713	82.3	82.4	82.4	82.4	**82.5**	**82.6**
s820	40.4	40.4	42.5	42.7	**45.6**	**46.6**
s832	39.4	39.4	41.4	41.6	**44.5**	**45.6**
s838	25.8	25.8	**30.9**	**31.4**	29.5	29.8
s953	97.8	97.8	99.1	99.1	99.1	99.1
s1196	93.6	93.6	96.1	96.1	**97.5**	**98.1**
s1238	88.4	88.4	91.2	91.2	**92.4**	**92.8**
s1423	33.3	34.9	56.7	59.3	**60.6**	**64.1**
s1488	33.0	33.0	41.0	41.0	**73.1**	**75.8**
s1494	32.3	32.3	40.2	40.2	**72.2**	**74.9**

Circuit	Fault Coverage (%)					
	LFSR*		LHCA*		Geffe*	
	Median	Max	Median	Max	Median	Max
s5378	67.1	68.1	68.5	**70.7**	**68.6**	70.4
s9234	5.9	5.9	5.9	5.9	5.9	5.9
s13207	23.8	27.2	23.6	25.9	**25.7**	**27.2**
s15850	5.5	5.5	5.5	5.5	5.5	**5.6**
s35932	83.3	83.9	62.1	62.8	83.3	**84.8**
s38417	12.8	13.9	**13.7**	**14.5**	13.2	14.2
s38584	49.4	50.4	49.0	49.4	**52.8**	**54.3**
Arithmetic Mean	*45.8*	*46.1*	*47.0*	*47.4*	*54.7*	*57.3*

*: The flip-flops of the benchmark circuits are set to logic 0 initially.

Table 4.27 Final results for ISCAS'89 circuits – flip-flops set to logic 0

The chart in Figure 4.10 shows the arithmetic mean of the median fault coverage of the 30 ISCAS'89 benchmark circuits (except the circuit s510) when they are stimulated with patterns generated by different types of generators. '*' means that the flip-flops in the circuits are initialized to logic 0.

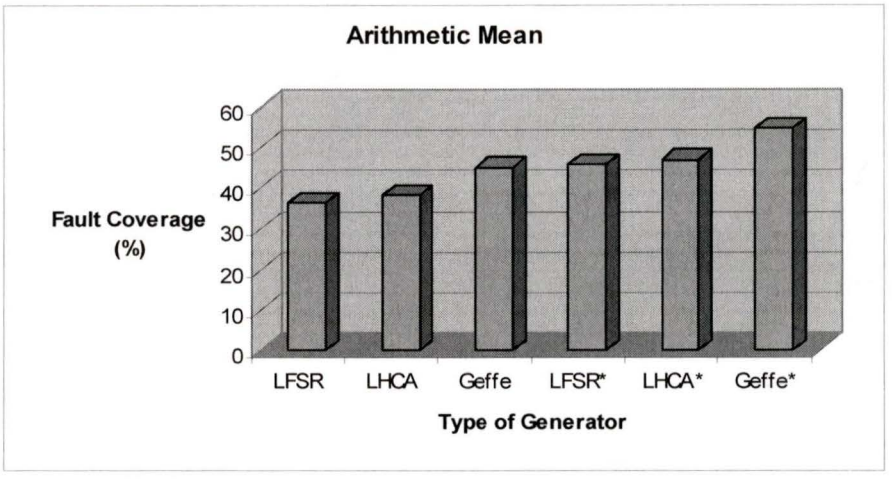


Figure 4.10 Arithmetic mean – ISCAS'89 benchmark circuits

Comparing the results in Table 4.27 with those in Table 4.24, we see that for most benchmark circuits (such as s1488), there is not much difference between the results when the flip-flops are set to logic U or logic 0 initially. But for some circuits (e.g., s510), there is a dramatic increase of the fault coverage when the initial value of the flip-flops is set to logic 0. For instance, for the circuit s510, 100% fault coverage is obtained after we initialized the flip-flops to logic 0 while the fault coverage is 0% at the default mode (logic U).

Why is there such a big difference for some circuits?

When testing sequential circuits, a sequence of patterns is applied to the Circuit Under Test (CUT), the states of the CUT advance step-by-step for time frames according to the input patterns. In the 3-value (logic 0, 1, U) fault simulation, all the flip-flops in the CUTs are set to the U state initially. When input patterns are simulated, the states of the flip-flops can be gradually set to the known states (logic 0 or logic 1), thus the fault can be further propagated and finally be detected.

There exist some U-undetectable faults which means that if the flip-flops of the circuit are set to the unknown state (U) initially, no matter what sequence of patterns is applied, these faults cannot be detected. Figure 4.11 shows such an example [26]. In the sequential circuit, the single stuck-at fault b s-a-1 cannot be detected if the D flip-flop is set to U initially, no matter what sequence of patterns is applied.

But this U-undetectable fault b s-a-1 can be detected if the D flip-flop begins at a known value. If we initialize the D flip-flop to logic 0, clearly, the sequence (10, 00, 00, 00) can detect this fault. Or if the initial value of the D flip-flop is logic 1, the sequence (10, 00) can detect the fault.

So the existence of the U-undetectable faults explains why for some sequential circuits, we can get much higher fault coverage when the initial value of the flip-flops is set to logic 0 instead of the unknown state U. The bigger difference between the results for the same benchmark circuit, the more such U-undetectable faults exist in the circuit.

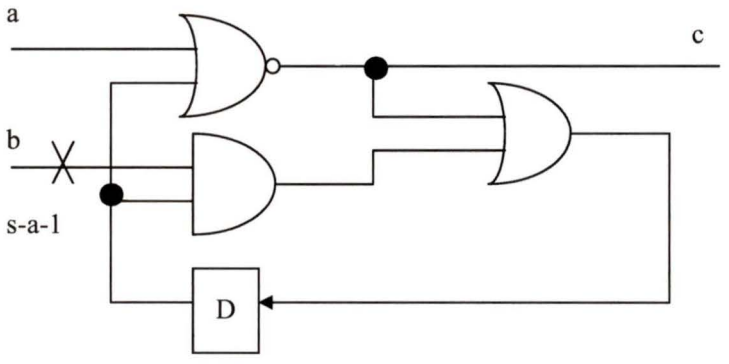


Figure 4.11 An example of a U-undetectable fault

The chart in Figure 4.10 also shows that when the initial value of the flip-flops in the circuits are set to logic 0, the LHCA performs slightly better than the LFSR, and the

Geffe generator performs much better than both LFSR and LHCA. This result supports the conclusion we reached in the earlier part of this section.

In the experiments, we also investigate some circuits with initial value of logic 1, the results are similar to those of the experiments in which the circuits are initialized to logic 0.

4.4 Conclusion

The experimental results of the randomness tests, the transition test, and the fault simulation are presented in this chapter. Analysis of the results as well as the conclusion for each test is also given.

The experimental results of the Knuth randomness tests show that the randomness properties of the Geffe generator resemble those of the linear machines of which it is composed. In our research, since the Geffe generator is made up of three LFSRs, it exhibits similar random properties as those of the LFSR. It is also shown that the LHCA has better random properties than the LFSR.

The conclusion of the transition test is that the non-linear machine has a much richer transition capability than the linear ones. Previous work [28] shows that the LHCA has a larger transition capability than the LFSR. Our experimental results show that patterns generated by the Geffe generator have considerably more transitions than those generated by the LFSR or LHCA.

Results of fault simulation show that when detecting stuck-at faults in combinational circuits, the Geffe generator is no better than the LFSR or LHCA. But for stuck-at faults in sequential circuits, the Geffe generator performs much better than the linear machines. These results also support the conclusion of the transition test.

Chapter 5 Conclusions

In this chapter, we present the contributions of our research and some of the possible directions for future work.

5.1 Contributions

The contributions of this thesis can be summarized as follows.

- Some properties of a non-linear machine, the Geffe generator, are studied. We research the randomness properties and the transition properties of the Geffe generator. Experimental results and analysis show that because of the non-linear function, the patterns produced by the Geffe generator have a very long period and offer a very large variety of transitions.
- The application of the Geffe generator as the PRPG in digital testing is investigated. Fault simulation experiments are designed and conducted using the Geffe generator. It is used as the PRPG to simulate both the ISCAS'85 and ISCAS'89 benchmark circuits. Experimental results, as well as the analysis, show that the Geffe generator has much potential to be applied in digital testing, and that it is especially good for detecting faults in sequential circuits.
- Comparison of the performance is made among the linear and non-linear machines. When we investigate the properties of the Geffe generator, we analyze those of the LFSR and LHCA as well. Comparison is made among the three generators. Experimental results show that the non-linear machine (the Geffe generator) has much richer transition capability than the linear machines (the LFSR and LHCA), and a much higher average fault coverage is obtained for the ISCAS'89 benchmark circuits when we use the Geffe generator as the PRPG

other than the LFSR or LHCA. A theoretical analysis of the experimental results is also given.

5.2 Future Work

Possible future work includes:

- Investigation of the non-linear machines as a PRPG to detect delay faults in digital testing. Similar to stuck-at faults in sequential circuits, a delay fault needs more than one pattern to detect. It is also the number of pattern transitions that determines the performance of a PRPG when it is used to detect delay faults. Since the non-linear machines have a much larger transition capability than linear machines, we hope they would exhibit better performance than the linear machines in detecting delay faults.
- Investigation of other non-linear machines and their applications in digital testing. Since the Geffe generator has shown better performance than the linear machines in testing sequential circuits, it would be interesting to investigate other non-linear machines, especially those whose area is comparable to that of the LFSR or LHCA, and their applications in testing.
- Investigation of the transition properties of the LFSR, LHCA, and other machines. In our research, the transition test experiments give some interesting results for different machines. Since linear machines can be described in mathematical forms, it would be useful to analyze their transition properties with mathematical methods.
- Investigation of other techniques for detecting faults in sequential circuits. In our experiments, we notice that for some ISCAS'89 benchmark circuits, no matter how many input patterns are applied, the fault coverage is always very low, even when the Geffe generator is used as the PRPG. For these hard-to-detect faults, it's

impractical to use (pseudo) random testing. Partial scan [2], genetic algorithms, and weighted pattern generators are some of the methods which are effective for these faults. It would be interesting to go into these areas.

Chapter 6 Refererences

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

[2] V. Agrawal, K. Cheng, and D. Johnson, T. Lin, *Designing Circuits with Partial Scan*, IEEE Design and Test of Computers, pp. 8-15, 1988.

[3] P. H. Bardell, *Analysis of Cellular Automata Used as Pseudorandom Pattern Generators*, Proc. of the International Test Conference, pp. 762-768, 1990.

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

[5] F. Brglez, D. Bryan, and K. Kozminski, *Combinational Profiles of Sequential Benchmark Circuits*, Proc. IEEE Int. Symposium on Circuits and Systems, pp. 1929-1934, 1989.

[6] F. Brglez, D. Bryan, and K. Kozminski, *Notes on the ISCAS'89 Benchmark Circuits*, MCNC, 1989.

[7] D. Bryan, *The ISCAS '85 Benchmark Circuits and Netlist Format*, 1988.

[8] K. Cattell and J. C. Muzio, *Analysis of One-Dimensional Linear Hybrid Cellular Automata over GF(q)*, IEEE Trans. on Computers, Vol. 45, pp. 782-792, 1996.

[9] K. Cattell and J. C. Muzio, *Tables of Linear Cellular Automata for Minimal Weight Primitive Polynomial of Degrees up to 300*, Technical Report, Department of Computer Science, University of Victoria, 1995.

[10] K. Cattell and J. C. Muzio, *Synthesis of One-Dimensional Linear Hybrid Cellular Automata*, IEEE Trans. on Computer Aided Design of Integrated Circuits and System, Vol. 15, pp. 325-335, 1996.

[11] F. M. Kadri, *Enhancing Transition Fault Coverage in Built-In Self-Test*, M. Sc. Thesis, Department of Computer Science, University of Victoria, 1993.

[12] D. E. Knuth, *Seminumerical Algorithms, Volume 2, The Art of Computer Programming*, Addison-Wesley, 1969.

[13] J. Koeter, *What's an LFSR?* Application Report, Texas Instruments, SCTA036A, 1996.

[14] H. K. Lee and D. S. Ha, *An Efficient Forward Fault Simulation Algorithm Based on the Parallel Pattern Single Fault Propagation*, Proc. of the International Test Conference, pp. 946-955, 1991.

[15] H. K. Lee and D. S. Ha, *HOPE: An Efficient Parallel Fault Simulator for Synchronous Sequential Circuits*, IEEE Trans. on Computer Aided Design of Integrated Circuits and System , pp. 1048-1058, 1996.

[16] S. Lin, Y. Z. Yu, and J. Zhong, *Software for Randomness Tests*, Technical paper, Department of Computer Science, University of Victoria, 2002.

[17] J. C. A. van der Lubbe, *Basic Methods of Cryptography*, Cambridge University Press, 1998.

[18] G. Marven, *Entropy Based Evaluation of Binary Sequences Produced by ALFSRs*, M. Sc. Thesis, Department of Computer Science, University of Victoria, 1994.

[19] U. M. Maurer, *A Universal Statistical Test for Random Bit Generators*, Journal of Cryptology, 1992.

[20] T. M. Niermann, W. T. Cheng, J. H. Patel, *Proofs: A Fast, Memory Efficient Sequential Circuit Fault Simulator*, Proc. ACM/IEEE Design Automation Conference, pp. 535-540, 1990.

[21] W. Pries, A. Thanailakis, and H. C. Card, *Group Properties of Cellular Automata and VLSI Applications*, IEEE Trans. on Computers, Vol. 35, pp. 1013-1024, 1986.

[22] M. Serra and G. L. Chen, *Pseudo-Random Pattern Generation and Fault Coverage of Delay Faults with Non Linear Finite State Machines with High Entropy*, Proc. IEEE On-Line Testing Workshop, 1997.

[23] M. Serra, T. Slater, J. C. Muzio, and D. M. Miller, *The Analysis of Linear Cellular Automata and their Aliasing Properties*, Trans. on Computer Aided Design of Integrated Circuits and System, Vol. 9, pp. 767-778, 1990.

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

[25] S. B. Wicker, *Error Control Systems for Digital Communication and Storage*, Prentice Hall, 1995.

[26] W. C. Wu, C. L. Lee, and J. E. Chen, *A Two-Phase Fault Simulation Scheme for Sequential Circuits*, Journal of Information Science and Engineering, Vol. 4, pp. 669-686, 1998.

[27] K. Zeng, C. Yang., D. Wei, and T. R. N. Rao, *Pseudo-random Bit Generators in Stream-cipher Cryptography*, Computer, 1991.

[28] S. Zhang, R. Byrne, J. C. Muzio, and D. M. Miller, *Quantitative Analysis for Linear Hybrid Cellular Automata and LFSR as Built-In Self-Test Generators for Sequential Faults,* Journal of Electronic Testing: Theory and Applications, Vol. 7, pp. 209-221,1995.

Appendix 1: ISCAS'85 netlist format for circuit c17

```
*c17 iscas example (to test conversion program only)
*----------------------------------------------------
*  total number of lines in the netlist ..............    17
*  simplistically reduced equivalent fault set size =     22
*        lines from primary input  gates .......     5
*        lines from primary output gates .......     2
*        lines from interior gate outputs ......     4
*        lines from **     3 ** fanout stems ...     6
*
*        avg_fanin  =  2.00,     max_fanin  =  2
*        avg_fanout =  2.00,     max_fanout =  2
*
    1     1gat inpt    1   0       >sa1
    2     2gat inpt    1   0       >sa1
    3     3gat inpt    2   0 >sa0 >sa1
    8     8fan from     3gat       >sa1
    9     9fan from     3gat       >sa1
    6     6gat inpt    1   0       >sa1
    7     7gat inpt    1   0       >sa1
   10    10gat nand    1   2       >sa1
     1     8
   11    11gat nand    2   2 >sa0 >sa1
     9     6
   14    14fan from    11gat       >sa1
   15    15fan from    11gat       >sa1
   16    16gat nand    2   2 >sa0 >sa1
     2    14
   20    20fan from    16gat       >sa1
   21    21fan from    16gat       >sa1
   19    19gat nand    1   2       >sa1
    15     7
   22    22gat nand    0   2 >sa0 >sa1
    10    20
   23    23gat nand    0   2 >sa0 >sa1
    21    19
```

Appendix 2: ISCAS'89 netlist format for circuit s27

```
# s27
# 4 inputs
# 1 outputs
# 3 D-type flipflops
# 2 inverters
# 8 gates (1 ANDs + 1 NANDs + 2 ORs + 4 NORs)

INPUT(G0)
INPUT(G1)
INPUT(G2)
INPUT(G3)

OUTPUT(G17)

G5 = DFF(G10)
G6 = DFF(G11)
G7 = DFF(G13)

G14 = NOT(G0)
G17 = NOT(G11)

G8 = AND(G14, G6)

G15 = OR(G12, G8)
G16 = OR(G3, G8)

G9 = NAND(G16, G15)

G10 = NOR(G14, G11)
G11 = NOR(G5, G9)
G12 = NOR(G1, G7)
G13 = NOR(G2, G12)
```

Appendix 3: Manual of the Software for Randomness Tests

In this part, the users manual of the software for randomness tests is given.

Manual of the Knuth tests[1]

The Knuth tests in this testbench are used to evaluate the randomness of sequences. Six tests are included: Equidistribution test, Serial-2 test, Poker-3 test, Gap test, Runup test, and Permutation test. For detailed information please refer to "The Art of Computer Programming", written by Knuth.

The tests here are for binary sequences and conversion is necessary. We convert N (N <= 100,000) successive n-bit (n <= 150) binary sequences into the corresponding decimal-value sequences, and the decimal-value sequences are further converted into the decimal-value sequences whose elements range from 0 to d-1 by taking their modulo d. Then we employ the Knuth tests on the latter decimal-value sequences. If the sequences pass a certain number of empirical tests, they are considered to be random enough.

~~~~~~~~~~~~~~~~~~~~ Example ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Suppose a binary sequence of N items, with N = 31, with each items of n bits, with n = 5, is as saved as follows in a file named "Mytest":

00001 00010 00100 01001 10010 00101 01011 10110
01100 11001 10011 00111 01111 11111 11110 11100
11000 10001 00011 00110 01101 11011 10111 01110
11101 11010 10101 01010 10100 01000 10000

Step 1: Select the name of the file.

Step 2: input an integer d for the modulo, e.g. "3".

---

[1] This part is written by Lin Sun.
~~~~~~~~~~~~~~~~~~~~

Step 3: Choose the desired Knuth test, e.g. "Gap test".

Step 4 : a sample output file may look like

*** SUMMARY OF GAP TEST RESULTS ***
Input File: Mytest
Number of Vectors:31
Modulo: 3

GapV=4.571424
Chi-Table show:
For the degree v= 4
Probability 0.95 is: 0.711
Probability 0.05 is: 9.488
So the sequences PASS the Gap Test.

Manual of the transition test

The transition test is used to count the number of sub-state vector transitions in a sequence of patterns. To run the test, the following parameters need to be specified:

1. Input file name: Path and name of the input file which contains the patterns;
2. Number of Sub-Vectors: Number of sub-state vectors to be counted, maximum=500,000;
3. Width of the Sub-Vectors: Number of bits of the sub-state vectors, maximum value=20;
4. Width of the Original Vectors: Number of bits of the original patterns in the input file;
5. Positions of the bits of the sub-vectors: The positions of the bits of the sub-state vectors in the original patterns. The positions of the sub-vectors are labeled left to right, starting from 1.

As an example, in the file "sample.txt", there are 10 patterns, each 6-bit wide. To count the transitions of all 10 3-bit wide sub-state vectors, selected from the 1st, 3rd, and 4th bits of the original patterns, the parameters are specified as follows:

Input file name: sample.txt;
Number of Sub-Vectors: 10;
Width of the Sub-Vectors: 3;
Width of the Original Vectors: 6;
Positions of the bits of the sub-vectors: 1 3 4.

"sample.txt" example file:

101010
100010
000101
011110
000011
110000
010001
000010
111111
100011

For the inputs above, we can get the following testing results:

******** SUMMARY OF RESULTS ********
******** Transition Test ********

1: Input Parameters
Input file name : sample.txt
Width of the sub-state vectors : 3
Positions of the bits of the sub vectors : 1 3 4
Number of vectors applied : 10

2: Testing Results
Number of possible transitions : 9
Number of transitions achieved : 9
Percentage of transitions achieved : 100%
******** END OF RESULTS ********

The positions of the sub-vectors are counted left to right, starting from 1.

For instance, in the example above, the 3-bit wide sub state vectors selected

from the 1st, 3rd, and 4th bits of the original patterns are:

110

100

001

011

000

100

000

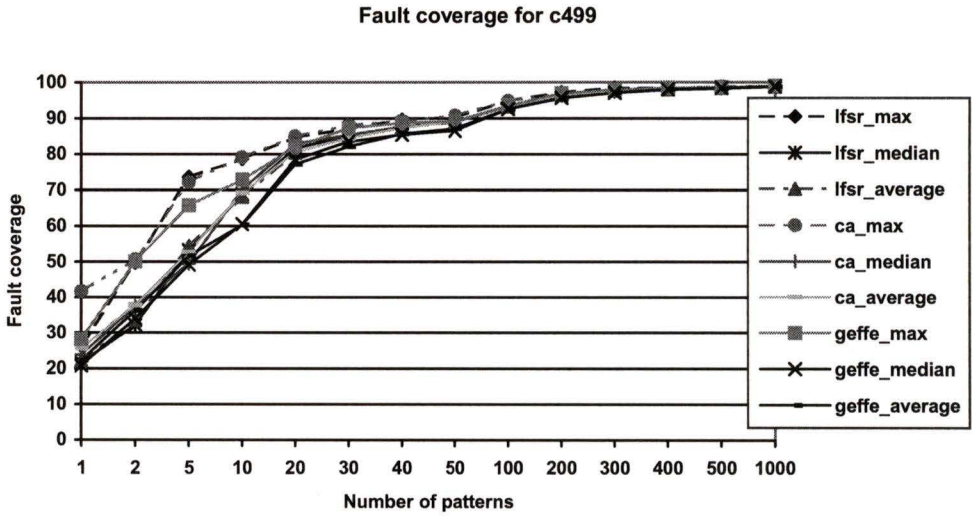
000

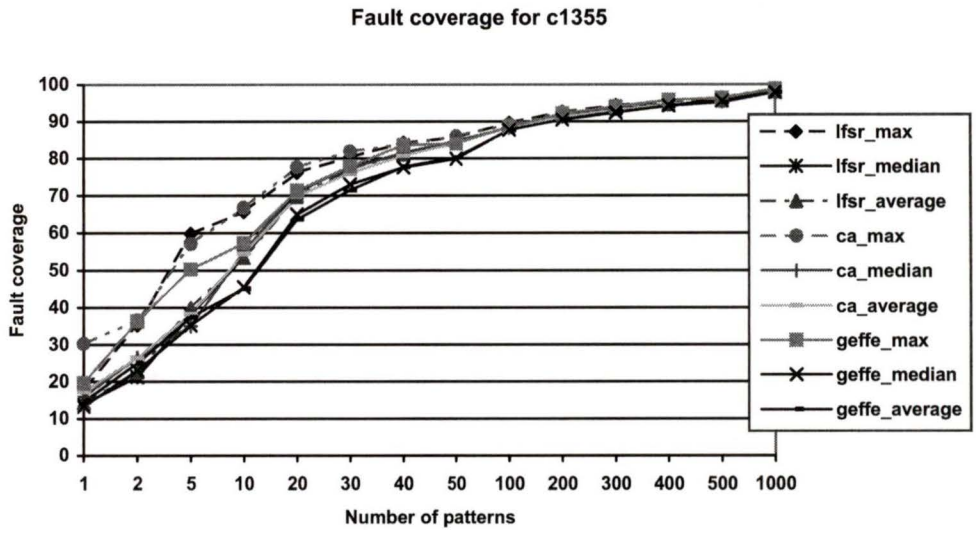
111

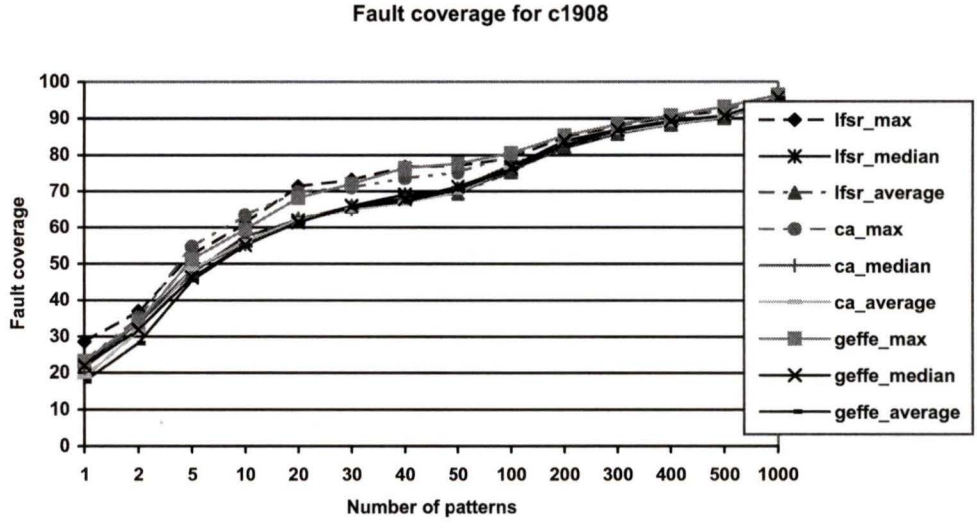
100

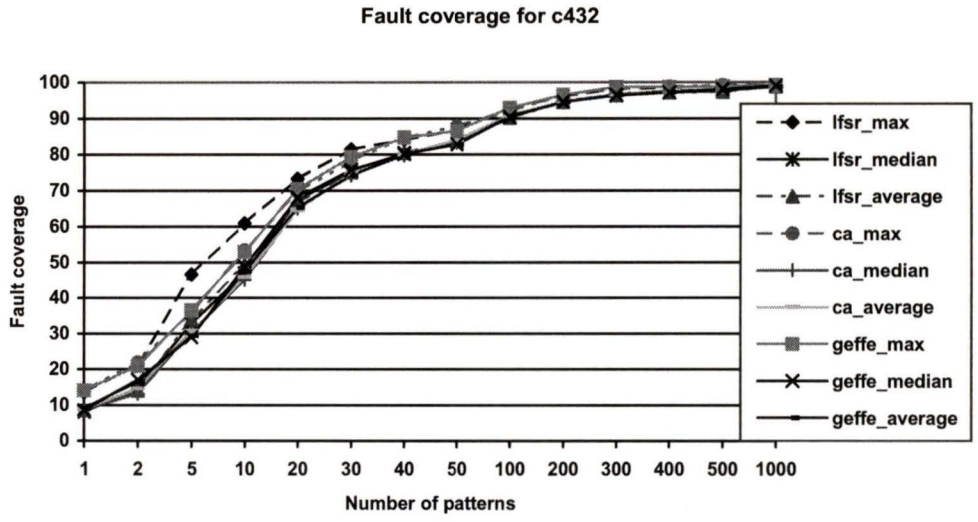
Appendix 4: Fault Simulation Results of the Combinational Circuits

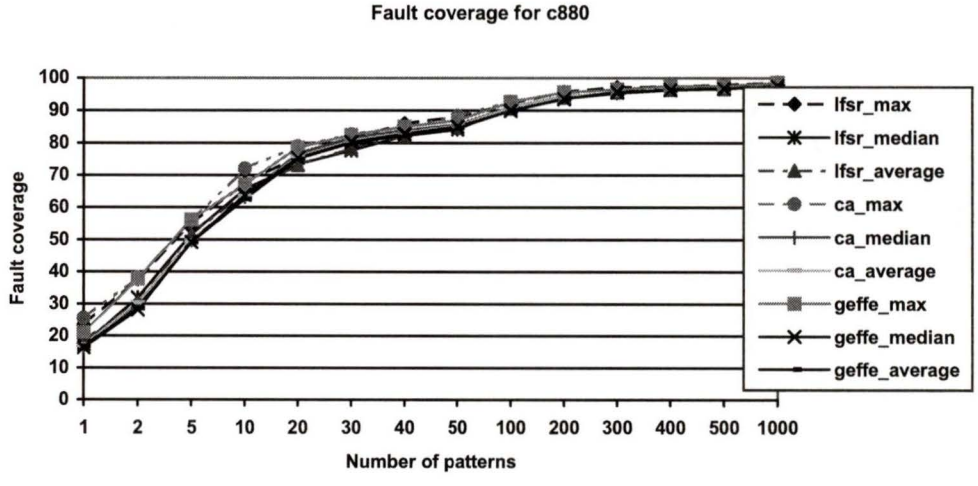
In this appendix, we give the simulation results of the ISCAS'85 benchmark circuits. Experiments are conducted according to the specifications in 3.3.4.

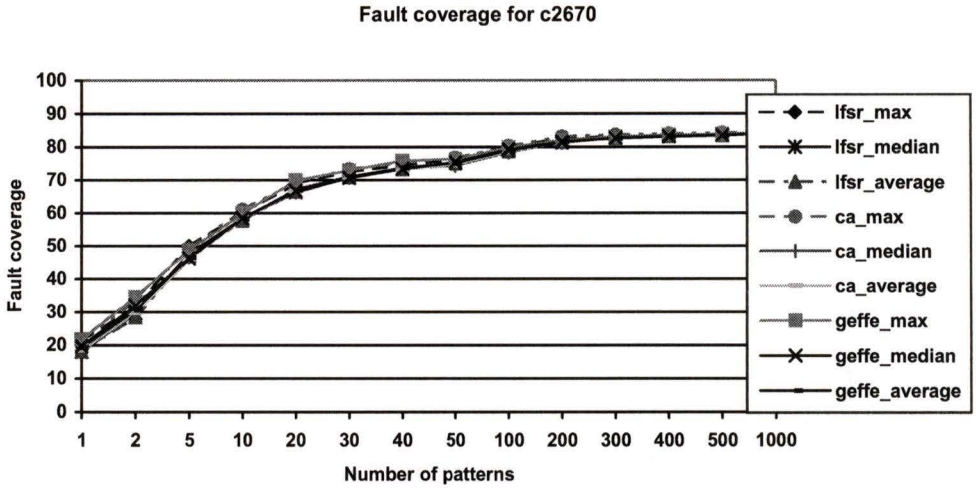


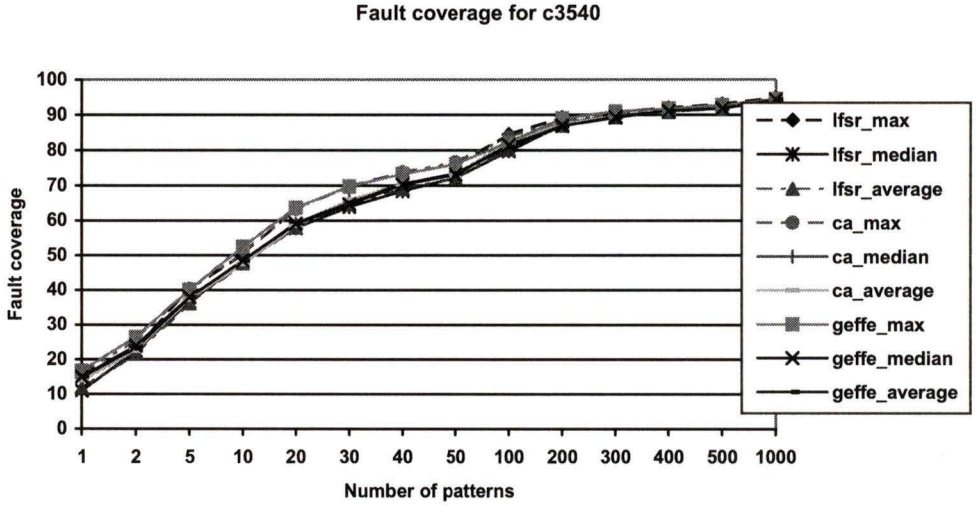

Fault coverage for c1355
Fault coverage
Number of patterns
0
10
20
30
40
50
60
70
80
90
100
1
2
5
10
20
30
40
50
100
200
300
400
500
1000
lfsr_max
lfsr_median
lfsr_average
ca_max
ca_median
ca_average
geffe_max
geffe_median
geffe_average

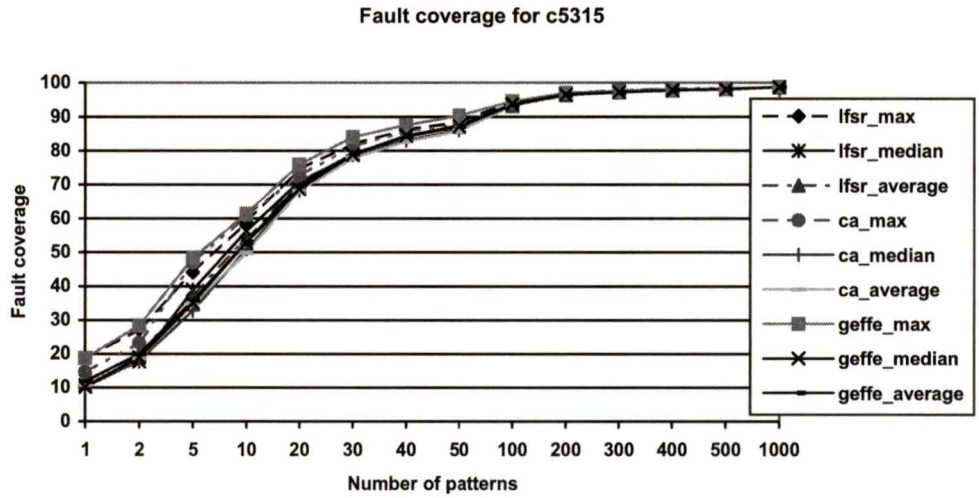

Fault coverage for c1908
Fault coverage
Number of patterns
0
10
20
30
40
50
60
70
80
90
100
1
2
5
10
20
30
40
50
100
200
300
400
500
1000
lfsr_max
lfsr_median
lfsr_average
ca_max
ca_median
ca_average
geffe_max
geffe_median
geffe_average

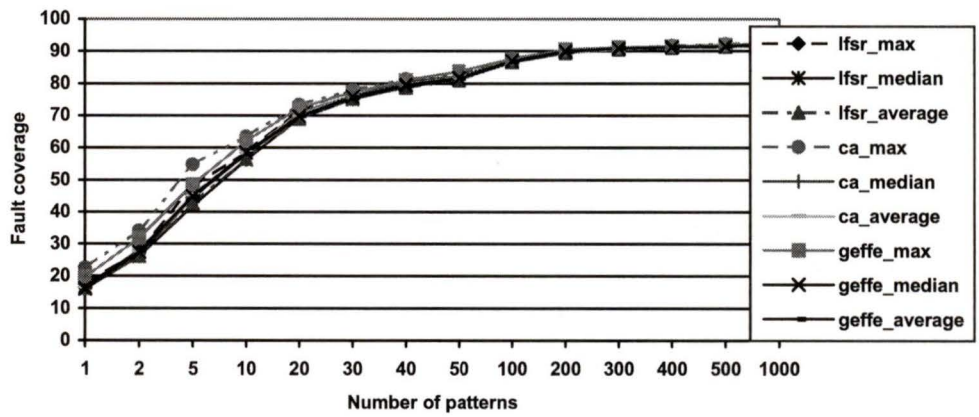

Fault coverage for c432
Fault coverage
100
90
80
70
60
50
40
30
20
10
0
1
2
5
10
20
30
40
50
100
200
300
400
500
1000
Number of patterns
lfsr_max
lfsr_median
lfsr_average
ca_max
ca_median
ca_average
geffe_max
geffe_median
geffe_average

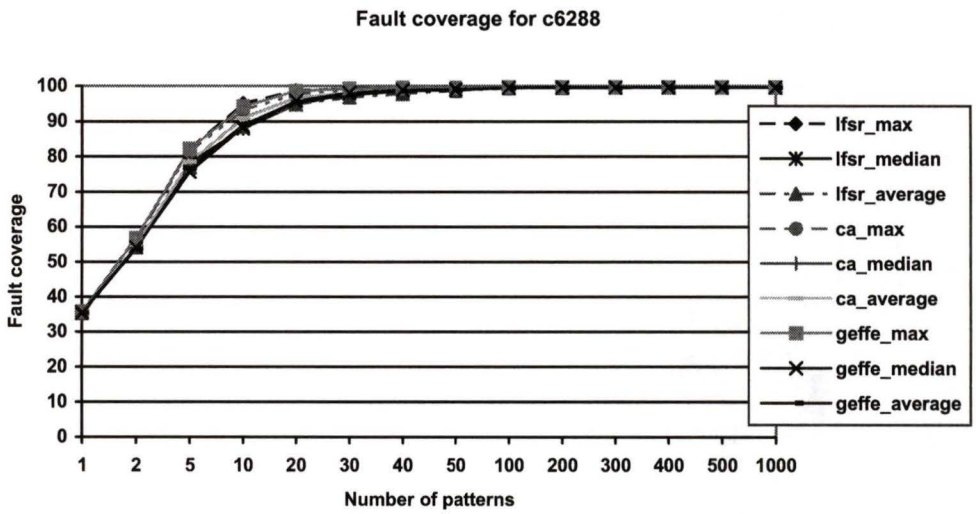

Fault coverage for c880
Fault coverage
100
90
80
70
60
50
40
30
20
10
0
1
2
5
10
20
30
40
50
100
200
300
400
500
1000
Number of patterns
lfsr_max
lfsr_median
lfsr_average
ca_max
ca_median
ca_average
geffe_max
geffe_median
geffe_average


Fault coverage for c2670
Fault coverage
100
90
80
70
60
50
40
30
20
10
0
1
2
5
10
20
30
40
50
100
200
300
400
500
1000
Number of patterns
lfsr_max
lfsr_median
lfsr_average
ca_max
ca_median
ca_average
geffe_max
geffe_median
geffe_average


Fault coverage for c3540
Fault coverage
100
90
80
70
60
50
40
30
20
10
0
1
2
5
10
20
30
40
50
100
200
300
400
500
1000
Number of patterns
lfsr_max
lfsr_median
lfsr_average
ca_max
ca_median
ca_average
geffe_max
geffe_median
geffe_average


Fault coverage for c5315
Fault coverage
100
90
80
70
60
50
40
30
20
10
0
1
2
5
10
20
30
40
50
100
200
300
400
500
1000
Number of patterns
lfsr_max
lfsr_median
lfsr_average
ca_max
ca_median
ca_average
geffe_max
geffe_median
geffe_average


Fault coverage for c7552
Fault coverage
100
90
80
70
60
50
40
30
20
10
0
1
2
5
10
20
30
40
50
100
200
300
400
500
1000
Number of patterns
lfsr_max
lfsr_median
lfsr_average
ca_max
ca_median
ca_average
geffe_max
geffe_median
geffe_average


Fault coverage for c6288
Fault coverage
100
90
80
70
60
50
40
30
20
10
0
1
2
5
10
20
30
40
50
100
200
300
400
500
1000
Number of patterns
lfsr_max
lfsr_median
lfsr_average
ca_max
ca_median
ca_average
geffe_max
geffe_median
geffe_average

The following table lists the raw data of the simulation results of the ISCAS'85 benchmark circuits.

		Number of Patterns													
Circuit		1	2	5	10	20	30	40	50	100	200	300	400	500	1000
c499	lfsr_max	26.4	49.7	73.6	78.6	84.7	87.1	89.6	90.2	94.9	97.2	98.4	98.4	98.7	98.9
	lfsr_median	21.7	32.3	53.3	68.8	81.5	85.3	87.5	89.1	93.3	96.6	97.8	98.2	98.5	98.9
	lfsr_average	22.0	33.5	54.4	68.1	80.4	85.2	87.3	88.9	93.3	96.6	97.7	98.1	98.5	98.9
	ca_max	41.7	50.7	72.2	79.2	85.0	88.0	89.3	90.8	94.9	96.8	98.0	98.4	98.8	98.9
	ca_median	23.9	37.5	49.5	70.2	82.9	85.4	87.7	89.0	93.3	96.4	97.4	98.1	98.4	98.9
	ca_average	25.6	37.9	52.8	69.2	80.6	84.5	87.2	89.2	93.3	96.3	97.4	98.1	98.3	98.9
	geffe_max	28.2	50.1	65.7	72.8	82.1	87.2	88.7	89.2	93.5	97.0	98.0	98.4	98.5	98.9
	geffe_median	20.8	34.0	49.2	60.6	78.5	83.5	85.4	86.5	92.6	95.6	97.1	98.0	98.4	98.8
	geffe_average	22.3	36.3	51.7	60.3	77.1	82.1	85.7	87.1	92.6	95.6	97.1	98.0	98.3	98.8

		Number of Patterns													
Circuit		1	2	5	10	20	30	40	50	100	200	300	400	500	1000
c1355	lfsr_max	18.2	35.3	59.8	65.7	76.2	80.3	84.2	85.6	89.6	92.6	94.3	95.3	96.3	98.7
	lfsr_median	13.9	21.3	38.1	53.5	70.9	77.9	81.0	84.0	88.5	91.7	93.4	94.5	95.7	98.1
	lfsr_average	14.2	22.6	39.9	53.4	69.7	77.2	81.1	84.0	88.5	91.7	93.5	94.5	95.6	98.0
	ca_max	30.3	36.5	57.1	66.8	77.6	81.9	83.9	86.0	89.2	92.4	94.2	95.2	96.4	98.9
	ca_median	15.9	26.5	35.0	55.1	71.4	77.3	81.6	84.7	88.2	91.2	93.3	94.6	95.3	98.0
	ca_average	17.3	26.5	38.3	54.5	69.8	76.1	80.7	84.1	88.3	91.3	93.1	94.5	95.3	98.1
	geffe_max	19.6	36.3	50.3	57.3	71.3	77.9	83.6	84.0	88.6	92.1	94.0	95.8	96.3	98.8
	geffe_median	13.2	23.1	35.2	45.4	65.0	73.0	77.5	79.9	87.7	90.5	92.4	94.3	95.6	97.9
	geffe_average	14.6	25.0	37.4	45.1	63.6	71.4	77.7	80.3	87.6	90.6	92.5	94.2	95.2	97.8

		Number of Patterns													
Circuit		1	2	5	10	20	30	40	50	100	200	300	400	500	1000
c1908	lfsr_max	28.5	37.0	52.4	61.2	71.4	73.2	76.7	77.1	79.0	85.0	87.8	90.5	92.1	96.5
	lfsr_median	22.6	33.7	47.5	57.5	61.9	66.0	69.1	69.5	76.0	81.9	85.9	88.5	90.1	95.3
	lfsr_average	23.2	33.8	47.5	57.3	62.3	65.8	68.0	69.3	75.4	82.2	85.8	88.4	90.1	95.3
	ca_max	23.3	35.3	54.7	63.3	69.8	71.1	73.7	75.2	79.7	85.4	87.9	90.2	92.4	95.5
	ca_median	21.2	33.8	48.5	55.4	62.5	65.0	68.1	70.8	77.2	83.2	86.3	88.2	90.4	95.1
	ca_average	18.9	31.4	48.5	56.2	62.1	64.8	66.9	69.6	76.5	83.1	86.2	88.4	90.4	95.0
	geffe_max	23.3	33.9	51.3	59.5	68.1	72.1	76.4	77.5	80.5	85.4	88.3	90.8	93.2	96.4
	geffe_median	22.0	31.9	46.2	55.1	61.4	65.9	67.9	71.3	76.6	83.9	87.0	89.3	90.8	95.6
	geffe_average	17.7	28.2	45.4	55.1	61.6	65.6	67.2	70.6	76.3	82.7	86.7	89.1	90.8	95.5

		Number of Patterns													
Circuit		1	2	5	10	20	30	40	50	100	200	300	400	500	1000
c432	lfsr_max	13.6	21.4	46.6	60.9	73.3	81.5	84.2	86.8	92.9	96.2	98.3	98.5	99.0	99.2
	lfsr_median	8.1	14.7	32.8	46.9	67.5	75.0	80.4	83.4	90.4	94.8	96.4	97.1	97.6	99.0
	lfsr_average	8.6	14.1	34.0	49.4	66.6	75.1	80.5	83.5	90.1	94.6	96.5	97.2	97.4	98.9
	ca_max	14.1	21.9	36.3	53.4	69.7	78.2	84.5	88.0	92.0	96.6	98.1	98.9	99.2	99.2
	ca_median	8.7	13.3	30.6	45.0	65.1	74.3	79.8	83.6	90.9	94.5	96.8	97.5	98.1	99.1
	ca_average	8.9	14.6	30.7	45.8	64.7	74.8	80.0	84.2	90.7	94.8	96.7	97.3	98.0	99.0
	geffe_max	14.1	21.0	36.5	52.9	70.4	79.4	84.7	86.6	92.9	96.6	98.7	98.7	98.9	99.2
	geffe_median	8.6	17.1	29.0	48.8	68.0	75.8	80.2	82.8	90.4	94.7	96.6	97.1	98.1	99.0
	geffe_average	9.1	16.6	29.4	47.5	65.4	74.2	80.1	82.9	90.6	94.6	96.5	97.2	97.7	98.8

		Number of Patterns													
Circuit		1	2	5	10	20	30	40	50	100	200	300	400	500	1000
c880	lfsr_max	23.6	37.6	54.9	67.6	75.8	81.3	86.0	88.1	92.0	95.8	97.1	97.3	97.8	98.6
	lfsr_median	17.3	32.0	51.7	65.5	73.3	77.7	82.1	84.2	91.0	94.0	95.5	96.4	96.7	97.6
	lfsr_average	18.3	30.5	51.4	64.7	73.2	78.3	82.3	84.7	90.6	94.1	95.6	96.4	96.7	97.7
	ca_max	25.6	38.1	55.7	72.0	78.9	82.6	84.9	88.3	92.7	95.3	96.6	96.8	97.2	98.5
	ca_median	16.5	29.9	49.7	63.2	76.7	81.5	83.9	85.7	91.0	94.4	95.3	96.4	96.9	97.8
	ca_average	17.2	30.4	49.5	63.5	75.9	80.7	83.3	85.4	90.9	94.3	95.4	96.2	96.7	97.8
	geffe_max	21.1	37.8	56.2	67.1	78.6	82.4	85.0	86.9	92.6	95.5	96.4	97.6	97.6	98.5
	geffe_median	16.3	28.1	49.2	63.7	75.3	80.3	82.7	85.0	89.9	93.5	95.6	96.3	96.7	97.6
	geffe_average	16.8	28.6	49.1	62.5	75.1	80.0	82.4	84.7	89.9	93.6	95.3	96.2	96.6	97.7

		Number of Patterns													
Circuit		1	2	5	10	20	30	40	50	100	200	300	400	500	1000
c2670	lfsr_max	21.2	31.7	49.9	59.9	69.1	72.6	74.5	76.2	79.8	82.6	83.0	83.5	83.8	84.2
	lfsr_median	18.0	29.6	47.6	58.4	66.3	70.8	73.4	75.2	78.7	81.3	82.7	83.2	83.5	84.1
	lfsr_average	18.1	28.5	47.0	57.7	66.6	70.9	73.3	75.0	78.7	81.5	82.7	83.2	83.4	84.1
	ca_max	21.1	33.6	49.2	61.0	69.4	73.4	75.1	76.6	80.2	83.1	83.6	84.0	84.2	84.4
	ca_median	18.1	30.1	46.7	58.0	67.3	71.0	73.5	74.3	78.3	81.5	82.8	83.4	83.7	84.2
	ca_average	18.4	29.7	45.9	58.1	66.7	70.9	73.2	74.6	78.5	81.6	82.8	83.4	83.7	84.2
	geffe_max	21.8	34.5	48.7	59.8	69.9	72.9	75.8	76.3	79.8	82.1	83.0	83.5	83.8	84.3
	geffe_median	19.7	31.9	46.2	58.5	66.7	70.7	73.4	75.4	79.2	81.4	82.6	83.1	83.4	84.1
	geffe_average	19.2	30.6	46.5	58.3	66.9	71.0	73.6	75.2	79.0	81.5	82.5	83.0	83.4	84.1

		Number of Patterns													
Circuit		1	2	5	10	20	30	40	50	100	200	300	400	500	1000
c3540	lfsr_max	15.4	23.5	40.2	50.4	63.4	69.7	73.7	75.8	84.5	89.2	90.8	92.0	93.1	95.0
	lfsr_median	10.9	22.5	36.5	48.1	57.8	64.0	68.5	72.2	79.7	87.3	89.6	90.9	91.9	94.2
	lfsr_average	11.8	21.6	36.2	47.7	58.0	64.7	69.1	72.2	80.2	87.0	89.4	90.9	91.8	94.2
	ca_max	15.8	25.7	40.4	50.7	63.2	69.8	73.7	76.6	83.8	88.7	90.3	91.7	92.6	94.7
	ca_median	14.6	22.8	37.2	47.7	59.0	65.7	70.5	73.1	82.2	87.8	89.7	91.0	92.1	94.5
	ca_average	14.0	22.6	37.4	47.5	58.4	66.0	70.2	73.1	82.4	87.6	89.6	90.9	92.0	94.4
	geffe_max	16.8	26.5	40.1	52.5	63.7	69.7	73.3	76.0	82.3	88.9	90.9	91.8	92.9	94.5
	geffe_median	15.0	24.0	37.9	48.5	59.1	64.6	70.2	73.1	81.4	86.9	89.2	91.1	91.9	94.3
	geffe_average	15.2	23.2	37.9	48.5	59.2	65.0	70.4	73.5	80.9	86.9	89.5	91.1	92.0	94.2

		Number of Patterns													
Circuit		1	2	5	10	20	30	40	50	100	200	300	400	500	1000
c5315	lfsr_max	18.7	27.3	44.1	58.9	74.6	82.0	86.2	88.4	94.4	96.9	97.6	98.1	98.2	98.7
	lfsr_median	10.7	17.7	38.9	56.3	70.9	78.7	84.3	87.5	93.3	96.4	97.2	97.7	98.1	98.6
	lfsr_average	11.9	19.8	37.6	54.0	70.6	79.1	84.2	87.1	93.2	96.3	97.3	97.8	98.0	98.6
	ca_max	14.6	23.3	46.9	60.9	72.7	81.2	85.9	87.6	93.9	96.7	97.7	98.2	98.3	98.8
	ca_median	10.1	18.0	32.8	50.9	68.2	78.6	83.1	86.2	93.4	96.4	97.2	97.8	98.1	98.6
	ca_average	10.8	18.6	34.9	49.8	68.0	77.8	82.7	85.8	93.2	96.4	97.3	97.8	98.1	98.6
	geffe_max	18.7	28.4	48.3	61.4	75.9	84.1	87.6	90.3	94.5	96.9	97.6	98.0	98.2	98.7
	geffe_median	10.2	19.0	35.0	52.9	68.7	78.9	84.5	87.3	93.7	96.5	97.2	97.7	98.0	98.7
	geffe_average	11.8	19.9	35.8	53.0	69.9	79.0	84.2	87.1	93.4	96.4	97.2	97.7	98.1	98.7

		Number of Patterns													
Circuit		1	2	5	10	20	30	40	50	100	200	300	400	500	1000
c7552	lfsr_max	17.6	27.7	48.0	58.6	71.4	77.9	80.2	82.3	87.2	89.8	90.9	91.7	91.8	92.7
	lfsr_median	16.0	26.2	41.8	56.3	69.1	75.1	79.0	81.3	86.6	89.4	90.8	91.2	91.5	92.3
	lfsr_average	16.2	26.2	42.6	56.5	68.9	75.2	78.8	81.0	86.6	89.4	90.6	91.1	91.4	92.3
	ca_max	22.6	34.1	54.7	63.4	73.3	78.0	81.3	82.8	87.8	90.3	91.0	91.7	92.2	92.8
	ca_median	16.1	26.8	44.4	58.3	70.8	76.2	80.5	82.2	87.3	89.6	90.5	91.0	91.4	92.0
	ca_average	16.7	27.8	45.3	58.5	70.4	75.9	79.7	81.8	87.1	89.7	90.6	91.1	91.4	92.1
	geffe_max	19.7	31.8	48.6	62.0	72.3	77.3	80.9	83.7	87.6	90.6	91.2	91.6	92.0	92.6
	geffe_median	16.1	27.5	44.9	58.5	69.7	75.9	79.5	81.6	87.0	89.9	90.8	91.2	91.5	92.4
	geffe_average	16.7	27.9	44.7	58.0	69.9	75.5	79.2	81.6	86.8	89.9	90.8	91.2	91.5	92.3

		Number of Patterns													
Circuit		1	2	5	10	20	30	40	50	100	200	300	400	500	1000
c6288	lfsr_max	35.7	56.6	82.1	95.1	98.7	99.3	99.5	99.5	99.6	99.6	99.6	99.6	99.6	99.6
	lfsr_median	35.4	54.1	78.2	88.0	94.8	98.0	98.8	99.1	99.5	99.6	99.6	99.6	99.6	99.6
	lfsr_average	35.3	54.1	76.9	88.5	94.8	96.8	97.8	98.7	99.5	99.6	99.6	99.6	99.6	99.6
	ca_max	35.7	56.7	81.3	93.1	97.7	98.9	99.4	99.5	99.6	99.6	99.6	99.6	99.6	99.6
	ca_median	35.4	54.1	78.4	90.8	96.7	98.2	99.0	99.2	99.5	99.6	99.6	99.6	99.6	99.6
	ca_average	35.3	54.8	78.5	90.7	96.7	98.2	99.0	99.2	99.5	99.6	99.6	99.6	99.6	99.6
	geffe_max	35.8	56.8	82.2	94.2	98.5	99.3	99.5	99.5	99.6	99.6	99.6	99.6	99.6	99.6
	geffe_median	35.4	54.1	75.7	88.4	95.8	98.1	98.9	99.0	99.5	99.6	99.6	99.6	99.6	99.6
	geffe_average	35.4	53.7	76.7	88.9	95.5	97.5	98.5	98.9	99.5	99.6	99.6	99.6	99.6	99.6

Appendix 5: Fault Simulation Results of the Sequential Circuits

The following table lists the experimental results of fault simulation for stuck-at faults in sequential circuits. In the experiment, the initial value of the flip-flops of the ISCAS'89 benchmark circuits is U (the unknown state).

The columns of the table denote the name of the ISCAS'89 benchmark circuit, the number of inputs of the circuit, the generator used for the simulation, median and maximum fault coverage (out of the ten sets of experiments) at the points of 64, 1000, 10000, and 20000 patterns.

Circuit	Inputs		Fault Coverage (%)							
			Number of Vectors							
			64		1000		10000		20000	
			median	max	median	max	median	max	median	max
s298	3	LFSR	20.8	20.8	20.8	20.8	20.8	20.8	20.8	20.8
		LHCA	51.3	51.3	51.3	51.3	51.3	51.3	51.3	51.3
		Geffe	29.2	40.6	42.2	80.2	50.0	83.8	50.0	83.8
s382	3	LFSR	11.8	11.8	11.8	11.8	11.8	11.8	11.8	11.8
		LHCA	13.3	13.3	13.3	13.3	13.3	13.3	13.3	13.3
		Geffe	12.0	12.3	12.3	13.5	12.3	17.0	12.3	17.0
s400	3	LFSR	11.3	11.3	11.3	11.3	11.3	11.3	11.3	11.3
		LHCA	13.4	13.4	13.4	13.4	13.4	13.4	13.4	13.4
		Geffe	11.8	12.0	12.0	13.7	12.0	17.0	12.0	17.0
s444	3	LFSR	10.3	10.3	10.3	10.3	10.3	10.3	10.3	10.3
		LHCA	11.0	11.0	11.0	11.0	11.0	11.0	11.0	11.0
		Geffe	11.2	11.2	11.2	15.4	11.2	16.2	11.2	16.2

Circuit	Inputs		Fault Coverage (%)							
			Number of Vectors							
			64		1000		10000		20000	
			median	max	median	max	median	max	median	max
s526	3	LFSR	8.5	8.5	8.5	8.5	8.5	8.5	8.5	8.5
		LHCA	8.5	8.5	8.5	8.5	8.5	8.5	8.5	8.5
		Geffe	8.6	8.6	8.6	11.9	8.6	12.3	8.6	12.3
s526n	3	LFSR	8.5	8.5	8.5	8.5	8.5	8.5	8.5	8.5
		LHCA	8.5	8.5	8.5	8.5	8.5	8.5	8.5	8.5
		Geffe	8.7	8.7	8.7	11.9	8.7	12.3	8.7	12.3
s27	4	LFSR	78.1	78.1	78.1	78.1	78.1	78.1	78.1	78.1
		LHCA	87.5	87.5	87.5	87.5	87.5	87.5	87.5	87.5
		Geffe	96.9	100.0	100.0	100.0	100.0	100.0	100.0	100.0
s386	7	LFSR	44.0	44.5	44.5	44.5	44.5	44.5	44.5	44.5
		LHCA	50.0	51.6	55.2	55.2	55.2	55.2	55.2	55.2
		Geffe	42.2	47.1	54.2	61.5	62.4	66.1	65.6	66.7
s1488	8	LFSR	32.1	32.6	32.8	32.8	32.8	32.8	32.8	32.8
		LHCA	36.4	38.9	40.8	40.8	40.8	40.8	40.8	40.8
		Geffe	37.0	43.0	54.8	64.0	70. 5	72.9	73.0	75.7
s1494	8	LFSR	31.4	31.9	32.1	32.1	32.1	32.1	32.1	32.1
		LHCA	35.7	38.2	40.0	40.0	40.0	40.0	40.0	40.0
		Geffe	36.3	42.2	54.0	63.0	69.5	72.0	72.1	74.8

Circuit	Inputs		Fault Coverage (%)							
			Number of Vectors							
			64		1000		10000		20000	
			median	max	median	max	median	max	median	max
s344	9	LFSR	69.0	72.5	73.4	73.4	73.4	73.4	73.4	73.4
		LHCA	57.9	58.8	58.8	58.8	58.8	58.8	58.8	58.8
		Geffe	79.1	84.5	91.8	93.6	95.9	96.2	96.2	96.2
s349	9	LFSR	69.1	72.6	73.4	73.4	73.4	73.4	73.4	73.4
		LHCA	57.7	59.1	59.1	59.1	59.1	59.1	59.1	59.1
		Geffe	78.7	84.3	91.4	93.1	95.4	95.7	95.7	95.7
s208	11	LFSR	27.0	27.0	27.0	27.0	27.0	27.0	27.0	27.0
		LHCA	22.8	22.8	22.8	22.8	22.8	22.8	22.8	22.8
		Geffe	25.8	31.2	34.4	40.9	36.7	42.3	41.2	42.3
s953	16	LFSR	8.2	8.2	8.2	8.3	8.3	8.3	8.3	8.3
		LHCA	8.3	8.3	8.3	8.3	8.3	8.3	8.3	8.3
		Geffe	8.2	8.3	8.3	8.3	8.3	8.3	8.3	8.3
s1423	17	LFSR	14.7	16.6	26.3	31.1	31.9	32.2	32.2	33.9
		LHCA	14.7	22.6	36.7	41.5	52.3	55.2	55.3	57.8
		Geffe	17.9	24.9	40.4	44.6	52.5	58.7	58.7	62.2
s820	18	LFSR	19.6	29.6	36.4	39.2	40.2	40.2	40.2	40.2
		LHCA	26.8	29.6	38.3	39.6	42.1	42.5	42.4	42.6
		Geffe	22.4	31.1	35.5	37.6	44.0	45.4	45.5	46.5

Circuit	Inputs		Fault Coverage (%)							
			Number of Vectors							
			64		1000		10000		20000	
			median	max	median	max	median	max	median	max
s832	18	LFSR	19.2	29.0	35.6	38.3	39.2	39.3	39.3	39.3
		LHCA	26.1	28.9	37.3	38.6	41.0	41.4	41.3	41.5
		Geffe	21.7	30.1	34.5	36.7	42.9	44.4	44.4	45.5
s420	19	LFSR	22.6	22.6	22.6	22.6	22.6	22.6	22.6	22.6
		LHCA	24.3	26.5	28.3	32.8	33.3	33.5	33.3	33.5
		Geffe	22.7	24.0	27.1	28.8	30.1	31.2	30.5	32.3
s510	19	LFSR	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
		LHCA	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
		Geffe	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
s9234	19	LFSR	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3
		LHCA	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3
		Geffe	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3
s13207	31	LFSR	5.7	5.8	6.1	6.2	6.2	6.4	6.2	6.4
		LHCA	5.6	5.8	6.1	6.2	6.2	6.4	6.2	6.4
		Geffe	5.7	5.9	6.1	6.1	6.2	6.4	6.2	6.4
s1196	14	LFSR	46.3	52.8	81.4	83.7	92.9	93.5	93.6	93.6
		LHCA	47.7	53.0	82.6	84.1	95.3	95.7	96.1	96.1
		Geffe	41.6	50.5	82.6	84.5	95.9	97.0	97.5	98.2

Circuit	Inputs		Fault Coverage (%)							
			Number of Vectors							
			64		1000		10000		20000	
			median	max	median	max	median	max	median	max
s1238	14	LFSR	41.7	47.5	76.4	79.0	87.7	88.2	88.4	88.4
		LHCA	43.4	48.3	78.2	78.9	90.5	90.8	91.2	91.2
		Geffe	38.0	46.4	77.7	79.2	90.7	91.8	92.4	92.8
s15850	14	LFSR	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7
		LHCA	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7
		Geffe	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7
s35932	35	LFSR	34.6	65.4	72.7	78.9	82.2	83.8	83.3	83.8
		LHCA	48.6	55.1	60.5	61.7	61.5	62.3	62.0	62.7
		Geffe	34.5	57.0	68.6	75.5	81.2	82.7	83.3	84.7
s5378	35	LFSR	40.5	48.6	62.8	63.7	65.3	65.5	65.5	65.5
		LHCA	41.5	50.0	62.7	64.5	66.2	69.2	66.4	69.4
		Geffe	39.8	50.4	63.0	64.5	66.0	67.7	66.4	69.2
s641	35	LFSR	63.4	68.7	80.6	82.2	85.4	85.9	86.1	86.3
		LHCA	65.0	67.2	82.4	83.3	86.1	86.3	86.3	86.3
		Geffe	63.8	66.4	82.0	83.9	86.1	86.5	86.4	86.5
s713	35	LFSR	62.0	67.3	77.2	78.5	81.1	81.4	81.6	81.8
		LHCA	64.0	81.8	78.7	79.3	81.6	81.8	81.8	81.8
		Geffe	63.1	64.2	78.3	79.9	81.6	81.9	81.8	81.9

Circuit	Inputs		Fault Coverage (%)							
			Number of Vectors							
			64		1000		10000		20000	
			median	max	median	max	median	max	median	max
s838	35	LFSR	20.0	20.2	20.2	20.2	20.2	20.2	20.2	20.2
		LHCA	20.2	24.9	22.8	24.5	25.2	25.2	25.2	25.7
		Geffe	19.3	20.4	22.1	23.7	23.7	23.9	23.8	24.0
s38417	28	LFSR	3.5	3.5	3.5	3.5	3.5	3.5	3.5	3.5
		LHCA	3.4	3.5	3.5	3.5	3.5	3.5	3.5	3.5
		Geffe	3.4	3.5	3.5	3.5	3.5	3.5	3.5	3.5
s38584	12	LFSR	7.1	9.5	15.5	17.9	18.7	18.7	18.7	18.7
		LHCA	9.0	11.2	16.2	18.3	19.7	19.7	19.7	19.7
		Geffe	7.7	10.8	16.1	17.3	19.9	20.8	21.1	21.6

Vita

Surname:	Zhong	Given Names:	Jing
Place of Birth:	Ji'an, Jiangxi Province, P. R. China		

Educational Institutions Attended:

Beijing University of Posts and Telecommunications	1994 – 1998
University of Victoria	2001 – 2003

Degrees Awarded:

B.Eng.	Beijing University of Posts and Telecommunications	1998

Partial Copyright License

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

Title of Thesis:

Non-linear Machines as Pseudo Random Pattern Generators for Digital Testing

Author:

Jing Zhong

September 13, 2003