The Selection of Better Polynomials to Reduce Aliasing in Signature Analysis

by

Mahbub Hassan
B.Sc., Middle East Technical University, 1989

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. M. Serra, Co-Supervisor (Department of Computer Science)

[redacted]

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

[redacted]

Dr. F. El-Guibaly, Outside Member (Department of Electrical Engineering)

[redacted]

Dr. A. Ivanov, External Examiner (Dept. of Elec. Eng., Univ. of British Columbia)

Co-Supervisors: Dr. M. Serra, Dr. J. C. Muzio

Abstract

This thesis is based on a search of better performing polynomials for signature analysis when used for the testing of digital circuits. Primitive polynomials are classified based on a number of different criteria and the evaluation of the performances of the polynomials of different classes is undertaken. Based on the statistics of "good" polynomials for different classes, a selection is suggested for the primitive polynomials to reduce aliasing. Finally this selection is tested by the simulation of a number of benchmark circuits.

Examiners:

[redacted]

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

[redacted]

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

[redacted]

Dr. F. El-Guibaly, Outside Member (Department of Electrical Engineering)

[redacted]

Dr. A. Ivanov, External Examiner (Dept. of Elec. Eng., Univ. of British Columbia)

Acknowledgements

I would like to thank Dr. Serra and Dr. Muzio very much for their profound help in directing my research and reading many drafts of my thesis. I would also like to thank Mr. Zhang for his assistance with the simulation part of my thesis.

Contents

Abstract ii

Acknowledgements iv

Contents v

List of Tables viii

List of Figures x

1 Introduction 1

2 Background 4

2.1 Principles of Testing 4

2.2 Test Pattern Generation 9

2.3 Signature Analysis 11

2.4 Linear Feedback Shift Register 12

2.4.1 Definition 12

2.4.2 Characteristic Polynomial of LFSR 15

2.4.3 Polynomial Division By LFSR 16
2.4.4 The Testing Applications of LFSRs 17
2.5 Cellular Automata Registers 18
2.5.1 Definition 18
2.5.2 Testing Application 19
2.6 Similarity Between LFSRs and CARs 20
2.7 Primitive Polynomials 21
2.8 Probability of Aliasing 23
2.8.1 Markov Process 23
2.8.2 Asymptotic Value of the Probability of Aliasing 29
2.8.3 An Algorithm to Calculate Probability of Aliasing 32

3 The Behavior of Signature Analyzers 35

3.1 Previous Work 36
3.2 Statistical Approach to The Identification of Low Aliasing Polynomials 38

4 Statistical Results 42

4.1 Statistics Based On the Degrees of the Polynomials 45
4.2 Statistics Based On the Roots of the Polynomials 45
4.3 Statistics Based On Weights 49
4.4 Statistics Based On Distribution of Non-zero Coefficients 54
4.4.1 Transition Count 54
4.4.2 Clusters 55

4.5 Selection of Better Primitive Polynomials 60

4.6 Statistical Significance of the Experimental Results 62

4.6.1 Standard Estimation Methods 62

4.6.2 Sampling Method of Our Experiment 63

5 Simulation **64**

5.1 Average Aliasing 65

5.2 Percentage of Test Lengths Having Aliasing 66

5.3 Simulation Results 66

5.4 Significance of the Results 70

6 Conclusion and Future Work **75**

6.1 Conclusion 75

6.2 Future Work 76

Bibliography **78**

A Primitive Polynomials and Their Reciprocals **82**

B Manual Page for 'sig' **84**

List of Tables

2.1 Different states of LFSR $x^4 + x + 1$ while dividing $x^6 + x^2 + 1$. 17

2.2 Sequence of states for ALFSR $x^3 + x^2 + 1$. 22

4.1 Percentage of *bad* polynomials of up to degree 16 for different types of primitive polynomials based on their roots. 47

4.2 Possible weights for primitive polynomials of degree 6-14. 49

4.3 Percentage of *bad* polynomials up to degree 16 for different polynomials based on their weights for $p = 0.1$. 50

4.4 Possible transition counts for primitive polynomials of degree 6-14. . . 55

4.5 Percentage of *bad* polynomials up to degree 16 for different polynomials based on their transitions for $p = 0.1$. Degree 2 primitive polynomial has no transition. 59

4.6 Percentage of *bad* polynomials up to degree 16 for different polynomials based on the distribution of non-zero coefficients. 60

4.7 Percentage of *bad* polynomials up to degree 16 for the selected and non-selected polynomials for $p = 0.1$. 61

5.1 AP(t) for different values of t. 65

5.2 Average AAP and APP for all the primitive polynomials of degree 14. 71

5.3 Average AAP and APP for all the primitive polynomials of degree 15. 72

5.4 Average AAP and APP for all the primitive polynomials of degree 16. 73

List of Figures

2.1 A digital circuit with seven lines. 6

2.2 A CMOS NAND gate with a short between two lines. 7

2.3 Overview of data compaction testing scheme. 11

2.4 A four stage linear feedback shift register. 13

2.5 A four stage type 2 LFSR. 14

2.6 A general LFSR. 15

2.7 An example of linear one-dimensional cellular automata. 18

2.8 An ALFSR implementing the polynomial $x^3 + x^2 + 1$. 21

2.9 State transition diagram for the LFSR $x^2 + x + 1$. 24

2.10 Transition diagram of Markov process for the LFSR $x^2 + x + 1$. . . . 26

3.1 Aliasing probability against test length for the primitive polynomial $x^9 + x^7 + x^4 + x^2 + 1$. 37

3.2 Aliasing probability against test length for the *bad* primitive polynomial $x^9 + x^4 + 1$. 39

4.1 AP graphs for different values of p for the primitive polynomial $x^5 + x^2 + 1$. 43

4.2 Percentage of *bad* primitive polynomials for various degrees. 46

4.3 Percentage of *bad* polynomials for different types (E, F, G, H) for various degrees of polynomials. 48

4.4 Percentage of *bad* polynomials for all possible weights for degree 14 primitive polynomials for $p = 0.1$. There are no primitive polynomial of weight 3 and 15 for degree 14. 51

4.5 Percentage of *bad* polynomials for all possible weights for degree 15 primitive polynomials for $p = 0.1$. 52

4.6 Percentage of *bad* polynomials for all possible weights for degree 16 primitive polynomials for $p = 0.1$. There are no primitive polynomials for weight 3 and 17 for degree 16. 53

4.7 Percentage of *bad* polynomials for all possible number of transitions for degree 14 for $p = 0.1$. There are no primitive polynomials with transition count 0 for degree 14. 56

4.8 Percentage of *bad* polynomials for all possible number of transitions for degree 15 for $p = 0.1$. 57

4.9 Percentage of *bad* polynomials for all possible number of transitions for degree 16 for $p = 0.1$. There are no primitive polynomials with transition count 0 for degree 16. 58

5.1 AAPs for various degrees of primitive polynomials for output 1 of the circuits SN74181, sao2.pla, and misex3c.pla for stuck-at faults. 68

5.2 APPs for various degrees of primitive polynomials for output 1 of the circuits SN74181, sao2.pla, and misex3c.pla for stuck-at faults. 69

Chapter 1

Introduction

The rapid technological development of recent years has made it possible to increase enormously the circuit density on a chip. With the increasing density, the testing of the chips has become much more difficult, since there is very limited access to the interior points of the circuitry. The normal testing approach of externally stimulating the chip and observing the outputs, has, in many cases, become too time consuming and expensive.

In an attempt to reduce the complexity of testing, techniques have been developed to move some or all of the testing functions onto the chip itself, or onto the board on which the chips are mounted. This way one can drastically reduce both the testing time and equipment. However, incorporating the testing circuitry onto the chip results in increased circuit and area overhead. Keeping this overhead to a minimum is, therefore, a major issue in these testing schemes [19].

A major problem faced in testing is the large volume of data to be processed. When the testing hardware is included in the chip, there is no alternative but to use some form of data compaction. There are several possible implementations for compaction circuits. However, these techniques all introduce some information loss and the possibility of accepting a faulty chip. A great deal of research is conducted

in this area to minimize this effect of data compaction.

There is a relationship between the implementation of such linear compaction circuits and a class of polynomials. This enables us to carry out a formal analysis of such structures and, under the normal assumption of errors, it is known that a certain class of polynomials ("primitive polynomials") performs better. The degree of the polynomial directly corresponds to the size of the compaction circuit. The number of possible polynomials available for a particular degree, even in the restricted class, is very large: for example, there are 1800 such polynomials for degree 15 alone.

The main goal of this thesis is to investigate possible criteria to discriminate amongst the large number of such primitive polynomials for better performance. This is accomplished through the development of some criteria for dividing the search space into several groups. The polynomials of each group are analyzed theoretically to compare the performance of different groups. Finally, a few benchmark circuits are simulated to observe how the selection criteria work in practice.

Discussions of the types of faults and their effects in VLSI circuitry and various implementations of compaction circuits can be found in a number of standard texts [2, 12, 19]. These topics are briefly reviewed in Chapter 2 of this thesis which provides background material, explains the principles of testing, and introduces different compaction circuits. In this Chapter, we also present a method for the theoretical analysis of the effectiveness of the compaction circuits.

In the first half of Chapter 3, a summary of the previous research in the area of selecting better compaction circuits is provided. A new *statistical approach* to this problem is introduced in the second half. In this approach, a set of criteria is defined to classify further the primitive polynomials into several groups to investigate the differing performances of the groups.

In Chapter 4, we explain the procedures to conduct the statistical tests and

present the results of these experiments. Based on these results, some selection criteria are proposed for the compaction circuits.

In Chapter 5, some benchmark circuits are simulated. The results are tabulated for different types of compaction circuits to evaluate the effectiveness of the selection criteria proposed in Chapter 4.

The new approach introduced in this thesis provides some selection criteria for linear compaction circuits. It is, however, not easy to evaluate the effectiveness of this selection in general, for all degrees and for all circuits. The results from Chapter 4 and Chapter 5 are summarized in the concluding chapter.

Chapter 2

Background

This chapter describes the basic material required for the exposition of the thesis. Initially, the general principles of testing are explained, including definitions of testing, fault models, and test pattern generation. Further topics discuss data compaction techniques using linear feedback shift registers and cellular automata registers, the errors resulting from such techniques and algorithms to calculate the probability of such errors.

2.1 Principles of Testing

A combinational digital circuit can be described by functions detailing the relationship between a set of primary inputs and a set of primary outputs. A certain output response is expected when the circuit is stimulated with some input combination. The digital system is said to be in error when the output response deviates from the expected one. The reason for this occurrence is that one or more *faults* are present in the system. A fault can be of two types — design fault or physical fault. Design faults are the faults in the design of the system, e.g., an incorrect logical connection between two modules. On the other hand, physical faults are due to problems in

the manufacturing process or degradation during operation. The former problems fall within the more precise category of "verification" of circuits. In this thesis, only testing of a digital system is discussed, concerned only with detecting physical faults.

There is a wide range of physical faults that can be present in a digital system. They differ according to the technology (bipolar, MOS, etc.), the density of integration, operating temperature, operating voltage and many other factors. Some common physical faults that occur during fabrication include faulty devices, breaks in lines at some level (polysilicon, diffusion, metal, etc.) and shorts between levels [19].

The primary interest in this work is in determining if a circuit is faulty or not, rather than locating the fault. It is much easier to detect the logical effect of a physical fault rather than discovering the exact physical attributes. The abstraction which maps a large number of physical faults into a smaller number of logical faults is called a *fault model.* There are different fault models used in the literature. The most common ones are: stuck-at fault, stuck-open fault, bridging fault and delay fault. Fault models are used more extensively in the circuits simulations of Chapter 5, while for the theoretical analysis of the compaction techniques only the resulting errors in the output stream of the circuit under test (CUT) are of interest. The stuck-at fault model is the most commonly used and is discussed below.

The stuck-at model assumes that the physical faults result in one or more lines being permanently stuck at logic 1 or 0. For example, if line 5 in figure 2.1 is stuck-at 1, then it will always be 1 irrespective of the input combination at a and b. If a fault involves only one line stuck-at-1 or stuck-at-0 then it is called a *single* stuck-at fault. Thus, in a circuit with r lines, $2r$ single stuck-at faults are possible. If we consider the single stuck-at fault model, there are a total of 14 faults in the circuit of figure 2.1 — $1/0, 1/1, 2/0, 2/1, \ldots, 7/0, 7/1$, where i/0 represents line i stuck at 0 and i/1

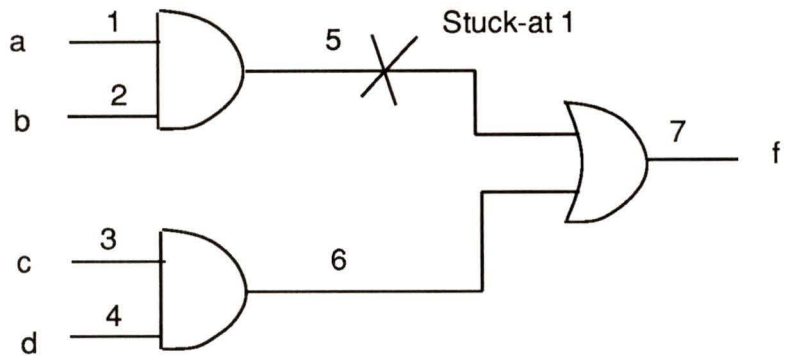
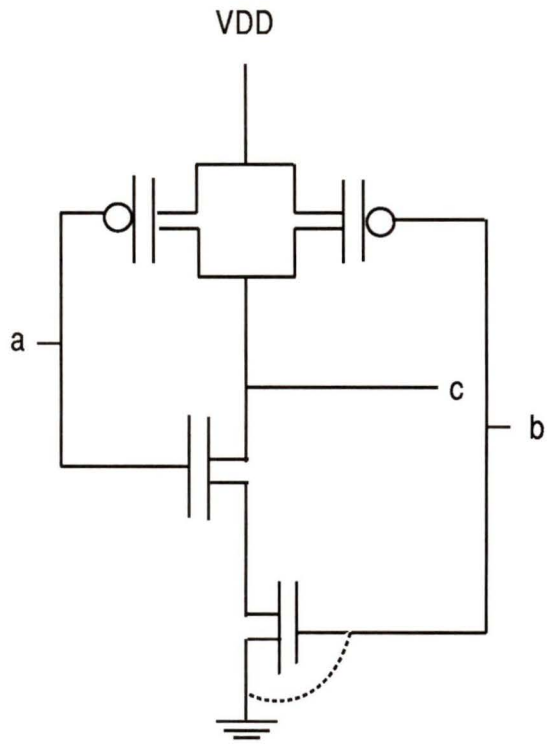


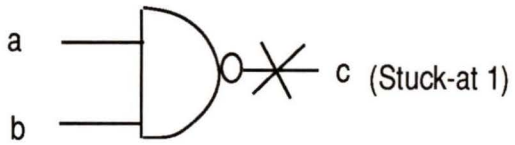
Figure 2.1: A digital circuit with seven lines.

denotes line i stuck at 1. On the other hand, if more than one line is stuck at 0 or 1, it is called a *multiple* stuck-at fault. The number of possible multiple faults in a circuit can be prohibitively large. Since each line can be in any of the three states — stuck-at 1, stuck-at 0, or fault-free, 3^r combinations are possible for a circuit with r lines. Discarding the combination where all the lines are fault-free, we have $(3^r - 1)$ possible multiple stuck-at faults. For the example of figure 2.1, $(3^7 - 1)$ or 2186 multiple stuck-at faults are possible. Usually, a circuit is tested against single stuck-at faults. In fact, many multiple stuck-at faults are tested implicitly while testing single stuck-at faults.

To illustrate how physical faults at transistor or circuit level are functionally equivalent to stuck-at faults at gate level, an example of a NAND gate is explained below. Physical failures such as open lines, shorts between lines at the transistor level may cause stuck-at faults at gate level. Figure 2.2(a) shows a CMOS transistor level realization of a NAND gate with two inputs a and b and one output c. Figure 2.2(c) shows the truth table for the fault-free NAND gate while figure 2.2(d) shows the truth table when input b is shorted to ground. Due to zero voltage to b, the bottom n-transistor never conducts and thus never passes GND (0) to c while the right hand side p-transistor is always active and passes VDD (1) to c. Thus we see that the short in the transistor level shown by the dotted line results in line c being



(a)



(b)

Fault-Free

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

(c)

Faulty

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

(d)

Figure 2.2: A CMOS NAND gate with a short between two lines.

stuck at 1 at the gate level which is shown in figure 2.2(b). Faults other than open lines and shorts may also cause stuck-at faults.

Another widely used fault model is *delay fault model*, which can be seen as "gate delay" or "path delay". We consider the latter. Let $c_1, c_2, \ldots, c_q$ be the correct (expected) bit sequence for a line of a circuit. The delay faults considered are termed *slow-to-rise* and *slow-to-fall* and yield the sequence $d_1, d_2, \ldots, d_q$ defined as follows [29]:

$$\textit{slow-to-rise} \quad d_i = \begin{cases} 1 & c_i = 1 \textit{ and } c_{i-1} = 1, \quad 1 < i \leq q, \\ 0 & \textit{otherwise}, \end{cases}$$

$$\textit{slow-to-fall} \quad d_i = \begin{cases} 0 & c_i = 0 \textit{ and } c_{i-1} = 0, \quad 1 < i \leq q, \\ 1 & \textit{otherwise}. \end{cases}$$

A circuit with r lines has $2r$ potential single delay faults.

Here a number of terms are defined [19] to introduce the testing procedure of a VLSI circuit. Given a circuit C, a *fault set* F is defined as a collection of faults under some fault model. We consider here only combinational faults, that is, faults which do not change a combinational circuit to a sequential one. Each input combination to a circuit is called an *input vector*. A *test pattern* is an input vector, which causes an incorrect output in the presence of certain specific faults belonging to F, when applied to the input of the circuit C. A fault f$\in$F is said to be *detected* or *covered* by a test pattern if that test pattern produces an incorrect output in the presence of f. A set of test patterns forms a *test set* T. The *fault coverage* of T for F is the fraction of faults in F covered by T. For T to be a complete test set for F, it is essential that for every fault f$\in$F, there must be at least one test pattern in T which detects f. A complete test set should consequently have 100% fault coverage. However, there may be untestable and redundant lines [19] in a circuit, so that the theoretical maximum of 100% coverage is not achievable in practice. The number

of test patterns in a test set is called the *test length.* A *minimal* test set T_m is a complete test set with minimal test length.

General testing procedures consist of 3 steps: test generation, test application, and test verification [19]. Fault simulation may also be necessary to calculate the fault coverage. In the test generation process, a test set is generated either algorithmically or pseudo-randomly. Once a test set is generated, the next step is to apply the test patterns to the inputs of the circuit under test (CUT). It is important to notice that a test set is generated once, while test patterns are applied many times over many CUTs. Hence, it is desirable to have a short test set in order to reduce testing time. Finally, in the test verification process, the response of the CUT is stored and compared with the precomputed correct response to see if the circuit is faulty or not. The test generation process is described briefly in the following subsection and the test verification is discussed in section 2.3.

2.2 Test Pattern Generation

Generating a test set algorithmically is a complex procedure. There are many algorithms (D algorithm, PODEM etc.) [6, 12] which use the gate level structure of the circuit to find a test pattern for a specific gate level fault. These algorithms are not discussed here. Usually a single test pattern can detect more than one fault and further work is necessary to reduce the fault set appropriately. A test set T is usually built up incrementally by using the following procedure [19]:

Begin
  $T = \Phi$;
  While F is not empty do
  Begin

Select a fault f∈F;
Determine algorithmically a test pattern t which detects f;
Determine F' = the set of faults detected by t;
$F \leftarrow F - F'$;
end;
end.

The above procedure may not lead to a minimal test set T_m. To obtain a T_m one has to generate test patterns for each f∈F explicitly, which is a very costly procedure. It is further possible to eliminate certain faults in F at the beginning by exploiting some dependencies (functional equivalence, fault covering, etc [19]) among the faults, which reduces both generation time and test length of the test set. Deterministic algorithms can be quite costly. Often complete fault coverage is also difficult to obtain.

From empirical data, it was also observed that a pseudo-randomly generated set of input patterns can detect a number of faults. Thus, instead of using a deterministic test set, one feasible testing method consists of applying a large number of pseudo-random input patterns L to the CUT. For many circuits, it is observed that the first 70% of the possible single stuck-at faults are detected quite easily [2], while it is very difficult to detect the rest of the faults. By simulation, one can determine the fault coverage achieved, and then, possibly, proceed to generate specific extra patterns by a deterministic algorithm for the faults yet uncovered. The greatest advantage of this pseudo-random technique to cover the first 70% of the single stuck-at faults is that it is much faster. The main problem with such an approach is the large volume of output responses produced due to the long L applied to the CUT. One testing scheme, called *signature analysis*, is described in the next subsection, which

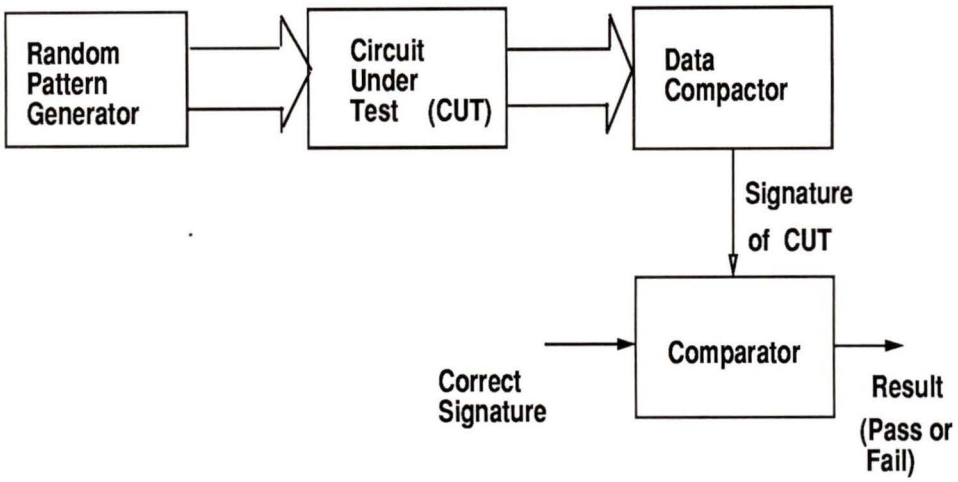


Figure 2.3: Overview of data compaction testing scheme.

compacts this large volume of data to facilitate the test verification process.

2.3 Signature Analysis

The large volume of output data, generated by applying a large number of pseudo-random input patterns to the CUT, has to be compared with the expected correct response and thus stored in memory.

A possible solution to this problem of storing and comparing a large amount of data is to compact the expected circuit response into a few bits, called the *signature* of the circuit. When the CUT is being tested and faults may be present, all output vectors are also compacted to signatures, which are compared to the correct signature to detect the faults. This process is called *signature analysis* [2, 5, 19, 25].

A schematic view of the data compaction testing scheme is shown in figure 2.3. The pattern generator produces pseudo-random patterns (up to exhaustive testing with $\leq 2^n$ vectors, where n is the number of inputs to the CUT) which are applied to the CUT. The data compactor compacts the output vectors (up to length $\leq 2^n$) into a short signature, which is compared with its correct counterpart in the comparator

to produce a pass/fail result. If the two signatures match, the circuit is passed as a good circuit, otherwise the circuit is designated as being faulty.

Data compaction solves the storage problem and simplifies the comparison process, as only the short signature needs to be examined. However, there is a major drawback of this process. The output response r_1 from a faulty circuit may produce a signature that is identical to the signature of output r_2 of the fault free circuit, where $r_1 \neq r_2$. This phenomenon is called *aliasing* [2, 5, 25] and it is unavoidable since the compaction process introduces some loss of information. One wants, in general, to reduce the 'probability' of aliasing as much as possible. In this thesis, aliasing problems in digital circuit testing are examined. Given the test length and the structure of the compactor, the probability of aliasing can be estimated using methods described in section 2.8. An exact calculation of the number of faults aliased for a specific compactor is only possible by circuit simulation, as shown in Chapter 5.

Many methods (one's count, transition count, parity check, syndrome testing, etc.) [2] can be used for producing a signature. Two well known structures for producing signatures are described in the following subsections: linear feedback shift register and cellular automata register.

2.4 Linear Feedback Shift Register

2.4.1 Definition

A shift register is a collection of binary storage elements connected so that the state of each element is shifted to the next element at each clock pulse. In a feedback shift register, a feedback function is present, such that the states of selected cells are fed back to previous cells as part of the computations of the next state. If the feedback function is linear, it is called a linear feedback shift register or LFSR.

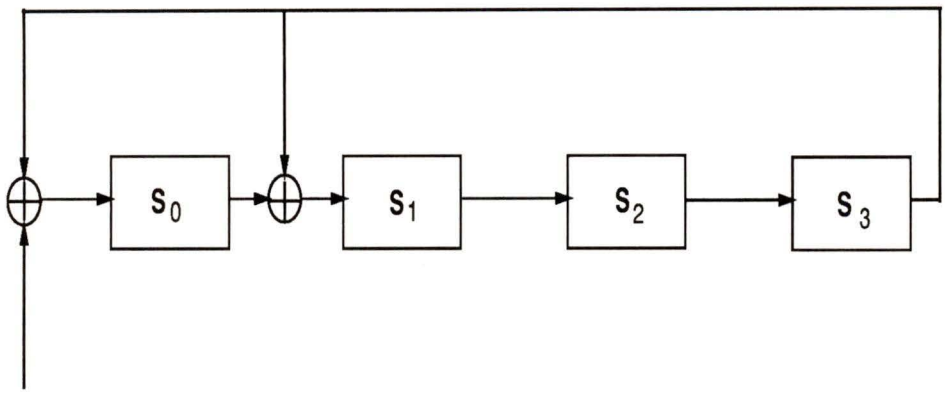


Figure 2.4: A four stage linear feedback shift register.

An example of a 4-cell LFSR is shown in figure 2.4, where the state of the cell S_3 is fed back to the cells S_0 and S_1. If we use S_i to denote the current state of cell i, S_i^+ to denote the next state of cell i and the symbol $\oplus$ to denote modulo 2 addition, the following next state equations hold for the example of figure 2.4:

$$\begin{aligned} S_0^+ &= S_3 \oplus input, \\ S_1^+ &= S_3 \oplus S_0, \\ S_2^+ &= S_1, \\ S_3^+ &= S_2, \end{aligned}$$

where the input bits are externally fed into the LFSR at each clock cycle. This is called a single input LFSR.

An alternate realization of the LFSR in figure 2.4 is shown in figure 2.5. Here all the feedbacks go into an external exclusive-or gate. The next-state equations for this LFSR are:

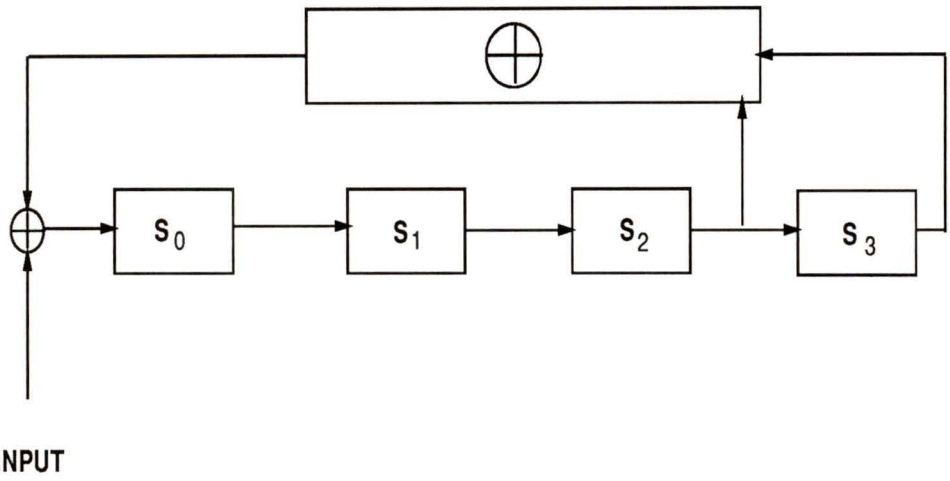


Figure 2.5: A four stage type 2 LFSR.

$$
\begin{aligned}
S_0^+ &= S_2 \oplus S_3 \oplus input, \\
S_1^+ &= S_0, \\
S_2^+ &= S_1, \\
S_3^+ &= S_2.
\end{aligned}
$$

The structure shown in figure 2.4 is called type 1 LFSR, while the structure shown in figure 2.5 is called type 2 LFSR in [25]. The two types are equivalent with respect to aliasing [23, 25].

In the LFSRs described above, there is only one external input fed into the first cell. When more than one input is present, the register is called *multiple input shift register* or MISR [2]. In section 2.4.4, the applications of LFSRs to signature analysis are explained in detail.

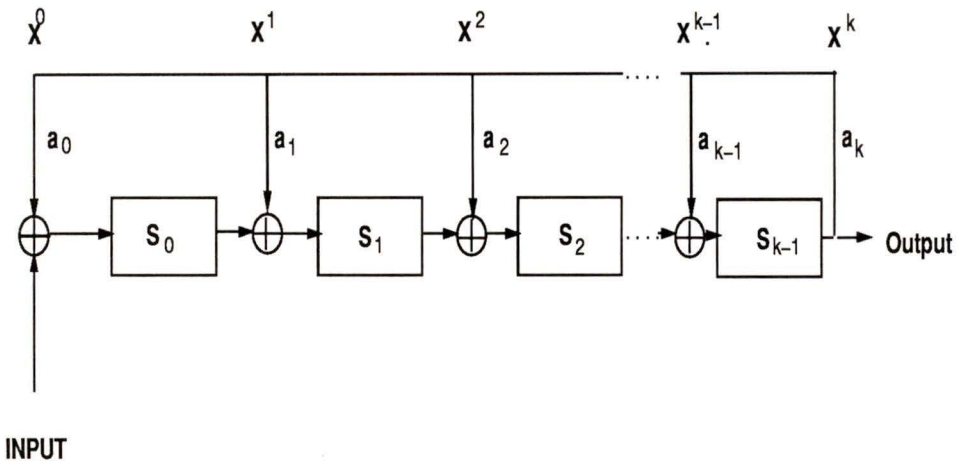


Figure 2.6: A general LFSR.

2.4.2 Characteristic Polynomial of LFSR

Usually the implementation structure of a LFSR is described by its 'feedback polynomial'. For the general type 1 LFSR in figure 2.6, the feedback polynomial is

$a_k x^k + a_{k-1}x^{k-1} + \ldots + a_2 x^2 + a_1 x^1 + a_0$, [25]

where $a_i = 1$ if the corresponding feedback is present in the LFSR, $a_i = 0$ otherwise. The terms a_k and a_0 are usually 1. The feedback polynomial of the type 1 LFSR shown in figure 2.4 is $x^4 + x + 1$. Note that the fewer the non-zero terms in the polynomial, the lower the hardware overhead (exclusive-or gate) and implementation cost for the LFSR.

Since the relationship between current state and next state of the LFSR is given by a linear function, the next state linear equations of the LFSR in figure 2.4 can be expressed in matrix form as:

$$\begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} S_0 \\ S_1 \\ S_2 \\ S_3 \end{pmatrix} + \begin{pmatrix} input \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} S_0^+ \\ S_1^+ \\ S_2^+ \\ S_3^+ \end{pmatrix}.$$

The matrix postmultiplied by the current state vector is called the *transition matrix.*

Given a $n \times n$ matrix A, one can compute the characteristic polynomial of A by calculating det(λ I - A), where I is the identity matrix of order $n \times n$. For applications in binary circuits, the elements of A and the coefficients of the characteristic polynomials are in GF(2) [1]. For our example, the characteristic polynomial for the transition matrix of the LFSR in figure 2.4 is $\lambda^4 + \lambda + 1$. Note that the characteristic polynomial is the same as the feedback polynomial used to denote the structure of the type 1 LFSR. Similarly, if one writes the transition matrix B for the type 2 LFSR in figure 2.5, the characteristic polynomial of B is also the feedback polynomial used to describe the LFSR in the first place. The similarity between the two types of LFSRs is discussed in section 2.6.

2.4.3 Polynomial Division By LFSR

Any binary stream can be represented by a polynomial whose coefficients are either 0 or 1. For example, the binary stream 1010001, where the leftmost digit represents the highest order coefficient, can be represented by the polynomial $1 \cdot x^6 + 0 \cdot x^5 + 1 \cdot x^4 + 0 \cdot x^3 + 0 \cdot x^2 + 0 \cdot x^1 + 1 \cdot x^0$ or $x^6 + x^4 + 1$. [18]

The implicit operation performed by a LFSR with a single external input is polynomial division over the field GF(2), where addition is performed modulo 2. The LFSR is initialized to zero. The feedback polynomial of the LFSR is considered the divisor polynomial and the input stream is considered the dividend polynomial. The input stream is fed through the input of the LFSR one bit at each clock cycle, high order coefficient first. At the end of $(s + 1)$ cycles, where s is the degree of the dividend, an output stream has been produced from the last cell of the LFSR which corresponds to the quotient. The last state of the LFSR is the remainder of the polynomial division. If a type 2 LFSR is used, the last state may not be the

[1] GF(2) is a field of two elements. Details can be found in [18, 24]

Clock	Input	S_0	S_1	S_2	S_3	output
t_0		0	0	0	0	
t_1	1	1	0	0	0	0
t_2	0	0	1	0	0	0
t_3	0	0	0	1	0	0
t_4	0	0	0	0	1	0
t_5	1	0	1	0	0	1
t_6	0	0	0	1	0	0
t_7	1	1	0	0	1	0

Table 2.1: Different states of LFSR $x^4 + x + 1$ while dividing $x^6 + x^2 + 1$.

remainder of the polynomial division.

As an example, the stages of the division of the input polynomial $x^6 + x^2 + 1$ (dividend) by the characteristic polynomial $x^4 + x + 1$ of the LFSR of figure 2.4 are shown in table 2.1. One can see that at the end of 7 cycles (the degree of the dividend is 6), the state of the LFSR is $1001(x^3 + 1)$, which is the remainder, and the output(quotient) is $0000100(x^2)$. This process is exploited in signature analysis as explained below.

2.4.4 The Testing Applications of LFSRs

LFSRs are widely used in signature analysis both as data compactors and as test pattern generators. Following the testing scheme of figure 2.3, the output stream from the CUT is fed into the LFSR, where it is considered to be the dividend polynomial, as explained above. The quotient (the output from the LFSR) is discarded and the last state of the LFSR, i.e., the remainder, is used as a signature of the CUT. If there is a detectable fault in the circuit then the output response of the

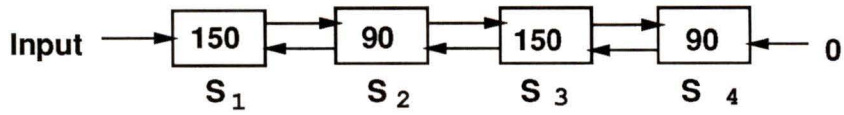


Figure 2.7: An example of linear one-dimensional cellular automata.

faulty circuit is different from the fault-free response. This in turn should produce a different signature as it should produce a different remainder after division. Aliasing occurs when the remainders for both the faulty and the fault-free response are the same. Aliasing, or its probability, can be minimized by choosing a long enough LFSR and possibly by making an appropriate selection of the feedback polynomial . The latter question is addressed in this thesis. More detailed discussion on aliasing follows in section 2.8.

2.5 Cellular Automata Registers

2.5.1 Definition

A linear one-dimensional cellular automata register (CAR) is a structure of interconnected cells in an array. The types of CAR we are interested in are one-dimensional nearest neighbor CAR, where each cell can communicate only with its nearest neighbors. Each cell can be in state 1 or 0 and the next states are determined by applying specific linear rules on the current states of each cell.

An example of a 4-cell CAR is shown in figure 2.7. Here one of the external inputs to the boundary cells is assumed to be 0 (the "null boundary condition"). If we denote the current state of cell i by S_i and next state by S_i^+, then the two most useful [2] linear rules for calculating next states are [23]:

[2]The other rules simplify to simple shifts between cells.

$$
\begin{aligned}
rule\ 150: \ S_i^+ &= S_{i-1} \oplus S_i \oplus S_{i+1}, \\
rule\ 90: \ S_i^+ &= S_{i-1} \oplus S_{i+1},
\end{aligned}
$$

where k is the number of cells, $1 \leq i \leq k$. S_0 is the input to the CAR and S_{k+1} is the null boundary input.

Using the above rules, one can derive the following linear equations for the next states for the example shown in figure 2.7.

$$
\begin{aligned}
S_1^+ &= input \oplus S_1 \oplus S_2, \\
S_2^+ &= S_1 \oplus S_3, \\
S_3^+ &= S_2 \oplus S_3 \oplus S_4, \\
S_4^+ &= S_3.
\end{aligned}
$$

Since the next state functions are linear, the transition between states can be represented by matrix for the above example as follows:

$$
\begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} S_1 \\ S_2 \\ S_3 \\ S_4 \end{pmatrix} + \begin{pmatrix} input \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} S_1^+ \\ S_2^+ \\ S_3^+ \\ S_4^+ \end{pmatrix}.
$$

Similarly to the LFSR, the characteristic polynomial $\lambda^4 + \lambda + 1$ of the above transition matrix is called the characteristic polynomial of the CAR. The similarity between a CAR and a LFSR is explained in section 2.6.

2.5.2 Testing Application

Similarly to the LFSRs, one can also use CARs both as data compactors and test pattern generators. As data compactors, the responses from the CUT are fed through

one of the boundary cells of the CAR, one bit at a time. The next state for each cell is calculated by applying the linear rules on the current states. After s clock cycles, where s is the number of bits in the output response of the circuit under test, the state of the CAR is accepted as a signature of the CUT. LFSRs and CARs have similar behavior in signature analysis. A detailed explanation on similarity is presented in the next section.

2.6 Similarity Between LFSRs and CARs

In the previous sections, three different possible structures for signature analysis are discussed — type 1 LFSR, type 2 LFSR, and CAR. All of them are linear finite state machines. It is explained below, how these three machines are similar to each other.

If two matrices A and B have the same irreducible [3] characteristic polynomial, then A is similar [4] to B [22]. It is shown in [23] that two different finite state machine with similar transition matrices, have the same aliasing, although different signature, when used as a signature analyzer. It is also shown in the same literature that given an irreducible polynomial, it appears to be possible to construct the above three structures such that, all of them have the same characteristic polynomial and thus are equivalent. The characteristic polynomial for the type 1 LFSR of figure 2.4, the type 2 LFSR of figure 2.5, and the CAR of figure 2.7 is $\lambda^4 + \lambda + 1$ and all of them are equivalent.

As we can see, all the three structures — type 1 LFSR, type 2 LFSR, and CAR — can be represented by polynomials. Thus, to predict the effect of a particular CAR or LFSR on signature analysis, one needs to examine only the corresponding

[3] A polynomial p(x) of degree n which is not divisible by any polynomial of degree k, where $0 < k < n$, is called irreducible.

[4] Two matrices A and B are similar if there exists a non-singular matrix P such that $A = PBP^{-1}$

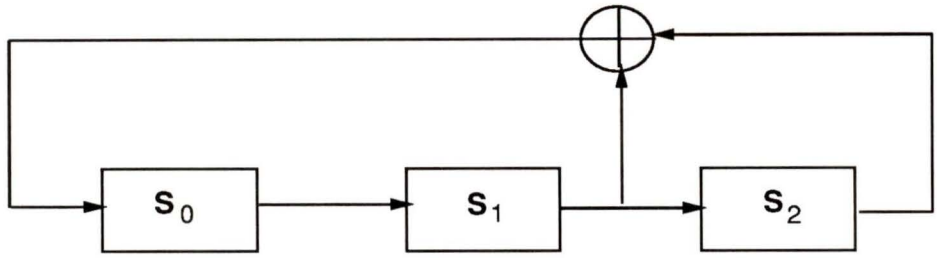


Figure 2.8: An ALFSR implementing the polynomial $x^3 + x^2 + 1$.

polynomial, given an appropriate error model for the output stream from the circuit. A special kind of polynomial, called a *primitive polynomial*, is of great interest in both test pattern generation and signature analysis and is explained in the following section.

2.7 Primitive Polynomials

An irreducible polynomial of degree m over GF(q) is called *primitive* if and only if it divides $X^n - 1$ for no n less than $q^m - 1$ [18]. A very useful behavior of a primitive polynomial, when used as a feedback polynomial for a LFSR, is described below.

If there is no external input to the LFSR, it is called an *autonomous* LFSR (ALFSR). An ALFSR representing the polynomial $x^3 + x^2 + 1$ is shown in figure 2.8. The next state equations of this ALFSR are as follows:

$$\begin{aligned} S_0^+ &= S_2 \oplus S_1, \\ S_1^+ &= S_0, \\ S_2^+ &= S_1. \end{aligned}$$

An ALFSR is a linear finite state machine. Each state is uniquely determined from the previous state by the feedback connections. Thus, if a state repeats after some number of clock cycles, all the following states will repeat, and the sequence of states

Clock	States		
	S_0	S_1	S_2
t_0	0	0	1
t_1	1	0	0
t_2	0	1	0
t_3	1	0	1
t_4	1	1	0
t_5	1	1	1
t_6	0	1	1
t_7	0	0	1

Table 2.2: Sequence of states for ALFSR $x^3 + x^2 + 1$.

is periodic. The state transitions for the ALFSR in figure 2.8 with the initial state 001 is given in table 2.2.

Given a k-stage ALFSR, at most 2^k states are possible. However, if an ALFSR is initialized to all-zero, it remains in that state. Thus, excluding the all-zero state, the sequence starting at one of the non-zero states is periodic with period no greater than $2^k - 1$. A k-stage ALFSR with period $2^k - 1$ is called a *maximum length* LFSR. This can happen if and only if the characteristic polynomial is primitive [18].

Since primitive polynomials can generate maximum length sequences, they are very useful for pseudo-random pattern generation. While there is no definitive proof that primitive polynomials are better than other polynomials as data compactors, it is concluded in recent research [4, 28] that, in general, the LFSRs implementing primitive polynomials are much better than those implementing non-primitive polynomials with respect to aliasing. In this thesis, the performance of primitive polynomial as data compactors is investigated.

2.8 Probability of Aliasing

Since the signature used in testing is r bits long, where r is very small compared to s, the test length or the length of the output response from the CUT, there is a considerable loss of information during data compaction, which is the cause of aliasing. Given the test length and the feedback polynomial of the signature register, it is desirable to know how often this aliasing may occur. One direct and accurate method to evaluate aliasing for a specific fault is to produce the signature by applying the test vectors to the simulated circuit with the fault present and compare the signature with the correct reference. Aliasing occurs if both signatures are identical. This process can be repeated for all the possible faults and we accurately determine how many fault can cause aliasing. For large circuits, however, the number of faults is large and this process is very costly.

To deal with this problem, an error model, called an *independent error model*, is introduced [28]. In this model, it is assumed that every output bit of the CUT is in error (different from the fault-free output) with probability p and successive bits are statistically independent. Thus, one is not concerned with any specific fault or fault model, assuming we consider only non-sequential faults. The length of the output stream (test length) from the CUT and the polynomial of the signature register are sufficient to calculate the probability of aliasing. In this section, it is explained how the probability of aliasing is calculated for the independent error model by simulating a Markov process [10, 26, 27, 28]. Calculation of exact aliasing by real circuit simulation is the topic of Chapter 5.

2.8.1 Markov Process

We can define a Markov process as a collection of different states of some system with *transition probabilities* associated with moving from one state to another. The

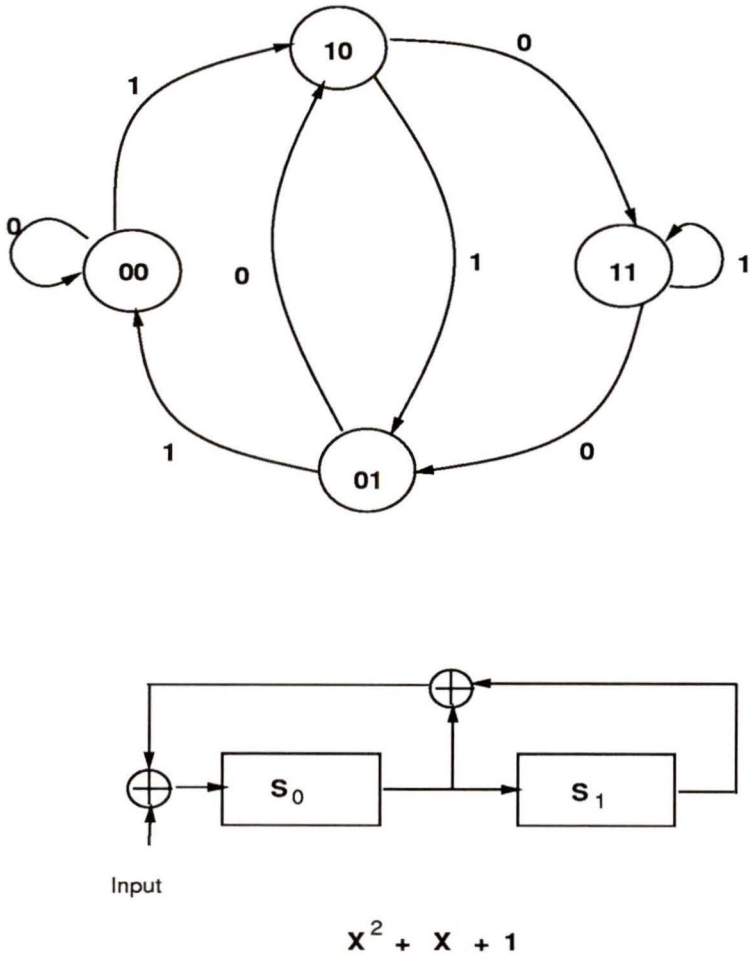


Figure 2.9: State transition diagram for the LFSR $x^2 + x + 1$.

transition probability, that is, the probability of making a transition from one state (state i) to another (state j), is not dependent on how the state i is reached, that is, on the previous history of the system. This is called a *homogeneous* Markov process [3]. More details on Markov processes can be found in [3, 17, 20].

Figure 2.9 shows the transition diagram of the LFSR represented by the polynomial $x^2 + x + 1$, where a circle shows a state of the LFSR (4 states are possible for a degree 2 LFSR) and an arrow shows a transition from one state to another. A binary number is attached to each arrow which shows the input value to the LFSR required for that transition.

When evaluating the transitions between states, one does not know, without direct simulation, what the next input is going to be. Hence a probability is attached to the input, which is considered to be random. Here we assume probability p of being a 1 and $(1-p)$ of being 0. With these changes we obtain the Markov diagram of figure 2.10.

If we assume that the probability of transition from state i to state j is always the same no matter when the state i is reached, then the LFSR can indeed be described by a homogeneous Markov process. From now on homogeneous Markov process will be assumed whenever we use the term Markov process.

The equations of the Markov process of the LFSR in figure 2.9 can be written easily from the state diagram shown in figure 2.10 [12]. The probability of being in any state S at time $(t+1)$ depends on

(i) the probability that the system was in some state S' at time t, and

(ii) the probability of a transition from S' to S.

For example, the probability that the LFSR of figure 2.10 is in state 00 at time $(t+1)$ depends on

(a) the probability that the LFSR was in state 00 at time t and the probability of the transition from state 00 back to state 00 $= (1-p)$,

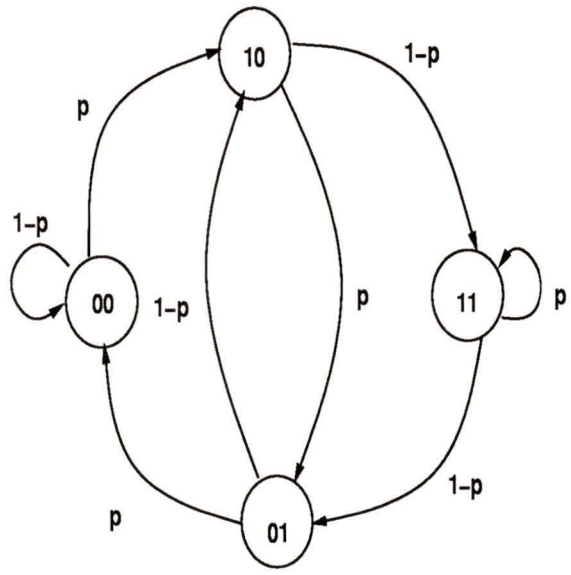


Figure 2.10: Transition diagram of Markov process for the LFSR $x^2 + x + 1$.

(b) the probability that the LFSR was in state 01 at time t and the probability of the transition from state 01 to state 00 $= (p)$.

Combining (a) and (b), we have

$$\Pi_{00}(t+1) = (1-p)\Pi_{00}(t) + p\Pi_{01}(t),$$

where $\Pi_{00}(t+1)$ is the probability of being in state 00 at time $t+1$; $\Pi_{00}(t)$ is the probability of being in state 00 at time t and $\Pi_{01}(t)$ is the probability of being in state 01 at time t.

Similarly we have the equations for the other 3 states as follows:

$$\begin{aligned}
\Pi_{01}(t+1) &= p\Pi_{10}(t) + (1-p)\Pi_{11}(t), \\
\Pi_{10}(t+1) &= p\Pi_{00}(t) + (1-p)\Pi_{01}(t), \\
\Pi_{11}(t+1) &= (1-p)\Pi_{10}(t) + p\Pi_{11}(t).
\end{aligned}$$

The above equations can be written in matrix form as

$$\begin{pmatrix} \Pi_{00}(t+1) \\ \Pi_{01}(t+1) \\ \Pi_{10}(t+1) \\ \Pi_{11}(t+1) \end{pmatrix} = \begin{pmatrix} 1-p & p & 0 & 0 \\ 0 & 0 & p & 1-p \\ p & 1-p & 0 & 0 \\ 0 & 0 & 1-p & p \end{pmatrix} \begin{pmatrix} \Pi_{00}(t) \\ \Pi_{01}(t) \\ \Pi_{10}(t) \\ \Pi_{11}(t) \end{pmatrix}.$$

The resulting matrix equation can be written in a condensed form as

$$\mathbf{\Pi}(t+1) = \mathbf{P}\mathbf{\Pi}(t),$$

where

$$\mathbf{\Pi}(t+1) = \begin{pmatrix} \Pi_{00}(t+1) \\ \Pi_{01}(t+1) \\ \Pi_{10}(t+1) \\ \Pi_{11}(t+1) \end{pmatrix},$$

$$\mathbf{P} = \begin{pmatrix} 1-p & p & 0 & 0 \\ 0 & 0 & p & 1-p \\ p & 1-p & 0 & 0 \\ 0 & 0 & 1-p & p \end{pmatrix},$$

$$\mathbf{\Pi}(t) = \begin{pmatrix} \Pi_{00}(t) \\ \Pi_{01}(t) \\ \Pi_{10}(t) \\ \Pi_{11}(t) \end{pmatrix}.$$

$\mathbf{\Pi}(t)$ is the probability state vector at time t, $\mathbf{\Pi}(t+1)$ is the probability state vector at time $t+1$ and $\mathbf{P}$ is the transition matrix.

If we denote the initial probability vector at time 0 as $\mathbf{\Pi}(0)$, we get

$$\mathbf{\Pi}(1) = \mathbf{P}\mathbf{\Pi}(0).$$

Similarly,

$$\begin{aligned} \mathbf{\Pi}(2) &= \mathbf{P}\mathbf{\Pi}(1), \\ &= \mathbf{P^2}\mathbf{\Pi}(0). \end{aligned}$$

In general,

$$\mathbf{\Pi}(t) = \mathbf{P^t}\mathbf{\Pi}(0).$$

If we denote the lth bit of the output response of the circuit under test by $CUT(l)$ and the lth bit of the fault-free response by $FF(l)$ then the following equation gives the lth bit of the *error sequence* of the circuit, called $E(l)$:

$$E(l) = FF(l) \oplus CUT(l),$$

where $\oplus$ is modulo 2 addition. It follows that one or more bits are 1 in the error sequence if the CUT is faulty, otherwise the error sequence is all-zero. It is sufficient to consider only the error sequence for calculating aliasing [9, 28]. Aliasing occurs when a non all-zero error sequence has the same result as a all-zero error sequence. It is explained below how this phenomenon can be detected.

If a LFSR is initialized to all-zero state and a stream of all zero is shifted into it, the LFSR never leaves the all-zero state. If the input stream, however, contains one or more 1s, the LFSR leaves the all-zero state and it may or may not return to all-zero state. Thus, the probability of aliasing, P_{al}, is defined as the probability of the LFSR starting in the all-zero state, leaving this state and returning to it at the end of the entire sequence[10, 26, 27, 28].

P_{al} can be calculated using the Markov process described earlier. The initial probability state vector $\mathbf{\Pi}(0)$ can be determined by setting the probability of the

all-zero state to 1 and all others to 0. The probability state vector at time t can be calculated as $\mathbf{\Pi}(t) = P^t\mathbf{\Pi}(0)$, where P is the transition matrix. The probability of being in all-zero state at time t defined by $\Pi_0(t)$ is an element of $\mathbf{\Pi}(t)$. However, aliasing requires that the situation where the LFSR always remains in the all-zero state be excluded. This situation happens when all the bits in the error sequence of length t are 0 and the probability of this situation to occur is $(1-p)^t$. Thus [10, 26, 27, 28]

$$P_{al} = \Pi_0(t) - (1-p)^t.$$

2.8.2 Asymptotic Value of the Probability of Aliasing

Aliasing for signature analysis is studied under different error models. One earlier model assumes all possible error sequences to be equally likely. According to this model, the probability of every output bit from the CUT to be in error is 1/2. It is shown in [25] that under this model, the aliasing probability is approximately 2^{-k} for long error sequence, where k is the length of the signature register. The *independent error model*, described in the beginning of this section, is more realistic in that the constant probability p of an output bit from the CUT to be in error can take any value between 0 and 1 ($0 < p < 1$). Williams et al. [28] have shown that the asymptotic value of 2^{-k} for the aliasing probability also holds for the *independent error* model for all polynomials with non-zero highest and lowest coefficients and for all values of p, $0 < p < 1$. The proof is explained below.

For a LFSR with k cells, 2^k states are possible and the order of the transition matrix $\mathbf{P}$ is $2^k \times 2^k$. The number of non-zero elements in the ith row of $\mathbf{P}$ represents the number of possible immediate predecessors of state i. Similarly, the number of non-zero elements in the jth column of $\mathbf{P}$ is the number of possible immediate successors of state j.

Under the assumption that the state of the rightmost cell of the LFSR is always

fed back to the input (the lowest and the highest order coefficients are non-zero), each state can be reached from exactly two different states. This is easily observed in figure 2.10, where there are exactly two incoming arrows to each state. If state S can be reached from state i and state j and an input 1 to the LFSR makes a transition from state i to state S, then an input 0 to the LFSR makes a transition from state j to state S. State i and state j, two predecessors of state S, differ only in the last bit. Thus each row of $\mathbf{P}$ has two non-zero elements — one is p and the other is $(1-p)$. If P_{ij} denotes the element in the ith row and jth column of the transition matrix $\mathbf{P}$, we have

$$\sum_{j=1}^{2^k} P_{ij} = 1 \quad i = 1, 2, \ldots, 2^k.$$

So, the transition matrix is *stochastic* [5].

Each state has exactly two different immediate successors. One of them is reached from the current state by applying a 1 to the input of the LFSR, while the other is reached by applying a 0. The two successors differ only in the first bit. Exactly two outgoing arrows from each state in figure 2.10 illustrates this. Since the number of non-zero elements in column j of $\mathbf{P}$ represents the number of immediate successors of state j, each column of $\mathbf{P}$ has two non-zero elements — p and $(1-p)$. We have

$$\sum_{i=1}^{2^k} P_{ij} = 1 \quad j = 1, 2, \ldots, 2^k.$$

So, the transition matrix $\mathbf{P}$ is *doubly stochastic*, i.e. both rows (necessary) and columns sum to 1. If $p \neq 1$ and $p \neq 0$, it is always possible to get from any state to any other state after some number of transitions. Thus the system is *ergodic*[6][13]

[5] Rows of a stochastic matrix consist of non-negative real numbers that sum to 1.

[6] An ergodic Markov process with transition matrix $\mathbf{P}$ has some matrix $\mathbf{P}^*$ such that $\lim_{m\to\infty} \mathbf{P^m} = \mathbf{P}^*$

and it follows that sooner or later the system will come to stability. In other words, after sufficient time t

$$\mathbf{\Pi}(t-1) = \mathbf{P^{t-1}\Pi}(0) = \mathbf{P^{*}\Pi}(0),$$

$$\mathbf{\Pi}(t) = \mathbf{P^{t}\Pi}(0) = \mathbf{P^{*}\Pi}(0) = \mathbf{\Pi}(t-1),$$

where $\lim_{t\to\infty} \mathbf{P^t} = \mathbf{P^*}$.

$\mathbf{\Pi}(t)$ can be written as

$$\mathbf{\Pi}(t) = \mathbf{P\Pi}(t-1).$$

Substituting $\mathbf{\Pi}(t-1)$ in the right hand side by $\mathbf{\Pi}(t)$,

$$\mathbf{\Pi}(t) = \mathbf{P\Pi}(t).$$

Thus,

$$\sum_{j=1}^{2^k} P_{ij} \cdot \Pi_j = \Pi_i \quad i = 1, 2, \ldots, 2^k.$$

The solution for Π_j is unique and since $\mathbf{P}$ is doubly stochastic, it follows that $\Pi_i = \Pi_j$ [28] for all $i, j = 1, 2, \ldots, 2^k$. Furthermore, since the LFSR has to be in some state at any time it follows,

$$\sum_{i=1}^{2^k} \Pi_i = 1.$$

Thus, the probability of being in any state is equal to the probability of being in any other state and all the probabilities sum to 1. Therefore, $\Pi_i = \frac{1}{2^k}$ for $i = 1, 2, \ldots, 2^k$. In other words, given p (the probability of a bit in the error sequence being 1, and $0 < p < 1$) the probability of being in any of the 2^k states of a k-stage LFSR is 2^{-k} irrespective of the value of p when the length of the error sequence tends to infinity.

This is true for any LFSR where the output of the rightmost stage is fed back to the input (the lowest and the highest order coefficients are non-zero). Thus, the probability of being in all-zero state and hence the probability of aliasing reaches asymptotically to 2^{-k} as the length of the input stream to the LFSR (length of the error sequence) tends to infinity.

2.8.3 An Algorithm to Calculate Probability of Aliasing

It is shown in section 2.8.1, that the probability state vector $\mathbf{\Pi}(t)$ at time t can be calculated from the probability state vector $\mathbf{\Pi}(t-1)$ at time $(t-1)$ by the formula $\mathbf{\Pi}(t) = \mathbf{P}\mathbf{\Pi}(t-1)$, where $\mathbf{P}$ is the transition matrix of the Markov process of the LFSR. The probability of aliasing, P_{al}, for the test length t, is calculated as $P_{al} = \Pi_0(t) - (1-t)^n$, where $\Pi_0(t)$ is the first element of the vector $\mathbf{\Pi}(t)$ and denotes the probability of the LFSR being in all-zero state after applying t inputs to the LFSR.

It is, therefore, possible to calculate the aliasing probability incrementally for each test length. At each stage, a $2^k \times 2^k$ matrix $\mathbf{P}$ is multiplied by a vector $\mathbf{\Pi}$ of 2^k elements, where k is the length of the signature register. The complexity for this multiplication is $O(2^k \times 2^k)$ or $O(4^k)$. This process becomes infeasible for large k. It is important to note that there are only two non-zero elements in each row of $\mathbf{P}$. Thus storing and processing the whole matrix is unnecessary and inefficient for both time and storage. This is utilized to develop and implement an algorithm [10, 15] which stores only the non-zero elements in $\mathbf{P}$ and makes the multiplication process much simpler and storage efficient. This algorithm has been used for the experiments in this work to calculate the aliasing probability for a given polynomial. A detail description of this algorithm follows.

In the example of section 2.8.1, the probability state vector $\mathbf{\Pi}(t)$ is calculated as

$$
\begin{aligned}
\Pi_0(t) &= (1-p)\Pi_0(t-1) + p\Pi_1(t-1), \\
\Pi_1(t) &= p\Pi_2(t-1) + (1-p)\Pi_3(t-1), \\
\Pi_2(t) &= p\Pi_0(t-1) + (1-p)\Pi_1(t-1), \\
\Pi_3(t) &= (1-p)\Pi_2(t-1) + p\Pi_3(t-1).
\end{aligned}
$$

Since each state has exactly two successors, the probability of each state at time $(t-1)$ contributes to the calculation of the probabilities of two states at time t. In the above example, each of $\Pi_0(t-1)$ and $\Pi_1(t-1)$ contributes to the calculation of $\Pi_0(t)$ and $\Pi_2(t)$, where states 0 and 2 are the two successors of each of states 0 and 1. Given the LFSR structure, the two successors for each state can be calculated from the next state equations. Let us keep the next states of each of 2^k states calculated with an input 1 to the LFSR in an array NEXT1 and the next states calculated with an input 0 to the LFSR in an array NEXT0. If the state probability vector $\mathbf{\Pi}(t)$ is denoted by the array NEWPR and $\mathbf{\Pi}(t-1)$ by the array OLDPR, the following procedure calculates the aliasing probability, P_{al}, for each test length, 1 to t.

```
Begin
/* INITIALIZATION PHASE */
   Calculate the next states for each state and initialize NEXT1 and NEXT2;
   OLDPR=(1,0,...,0) (Initially the LFSR is in state 0, which is the all-zero state);
   NEWPR = 0;
   /* CALCULATE ALIASING PROBABILITY */
   /* outer loop for every test length, 1 to t */
   While(i ≤ t) do
   Begin
```

$j = 0$;

/* inner loop for all the 2^k states */

While($j \leq 2^k - 1$) do

Begin

NEWPR[NEXT1[j]] = NEWPR[NEXT1[j]] + p*OLDPR[j];

NEWPR[NEXT0[j]] = NEWPR[NEXT0[j]] + $(1-p)$*OLDPR[j];

$j = j + 1$;

End;

P_{al} = NEWPR[all-zero state]$-(1-p)^i$;

OLDPR = NEWPR;

NEWPR = 0;

$i = i + 1$;

End;

End.

The time complexity of the initialization phase and the inner loop of the calculation phase is $O(2^k)$. If the outer loop in the calculation phase is run for l times, the time complexity of the whole procedure is $O(l2^k)$.

Each of NEXT1, NEXT0, NEWPR and OLDPR requires 2^k storage. The whole process, therefore, requires 4×2^k storage and the storage complexity of this algorithm is $O(2^k)$.

The original iterative algorithm was developed in [10]. Similar implementations can be found in [15] and [21].

Chapter 3

The Behavior of Signature Analyzers

As defined earlier, LFSRs and CARs are finite state machines, where each state is defined uniquely by the previous state and the input. Because of the linearity of the feedback function, a state transition matrix can be formed from the finite state machine equations. As shown in chapter 2, the transition matrix has a significant influence on the calculation of the aliasing probability (AP). The behavior of the finite state machine is determined by the minimum [1] polynomial of the transition matrix. The characteristic polynomial of the transition matrix is a multiple of the minimum polynomial of that matrix [22]. It follows that if the characteristic polynomial is irreducible, it is equal to the minimum polynomial. As the feedback polynomial of the LFSR is the same as the characteristic polynomial of the transition matrix, one can presume that the polynomial chosen for a signature analyzer, is a major factor in determining the chances of aliasing. Some authors [4, 11, 21, 28] have published results on selecting polynomials that minimize aliasing. In this chapter, a

[1] The minimum polynomial of a matrix A is defined to be the monic (the highest order coefficient is 1) polynomial of lowest degree which annihilates A [14].

summary is given of the relevant research and the problem of predicting the behavior of a polynomial is approached.

3.1 Previous Work

It is shown in section 2.8.2, that the aliasing probability (AP) for any signature analyzer, reaches an asymptotic value of 2^{-k}, where k is the number of stages in the compactor, as the test length tends to infinity. If the AP is plotted against the test length, we obtain graphs such as those shown in figure 3.1 and figure 3.2. The aliasing probability is said to have reached *steady state* when it is very near to the asymptotic value of 2^{-k} and shows no significant changes. The region before the steady state is called the *dynamic region*, since the AP changes significantly in this region. The APs for some polynomials behave monotonically in the dynamic region (figure 3.1), while others are non-monotonic and have peaks (figure 3.2). The number of test vectors needed for the AP to reach the steady state, is called the length of the dynamic region. In the example of figure 3.1, the length of the dynamic region is approximately 100 vectors.

All the polynomials can be characterized as *primitive* and *non-primitive*. In 1986, Williams et al. showed in [28], by analyzing the Markov process, that primitive polynomials reach steady state faster than non-primitive polynomials and concluded that, in general, primitive polynomials have lower aliasing probability than non-primitive ones.

Later Damiani [4] analyzed the Markov process by a more detailed mathematical analysis and confirmed the earlier results of Williams. He further suggested that primitive polynomials of the same degree are equally good with respect to aliasing and those with minimum implementation cost for the signature register are best suited for the on-chip realization of signature analysis techniques.

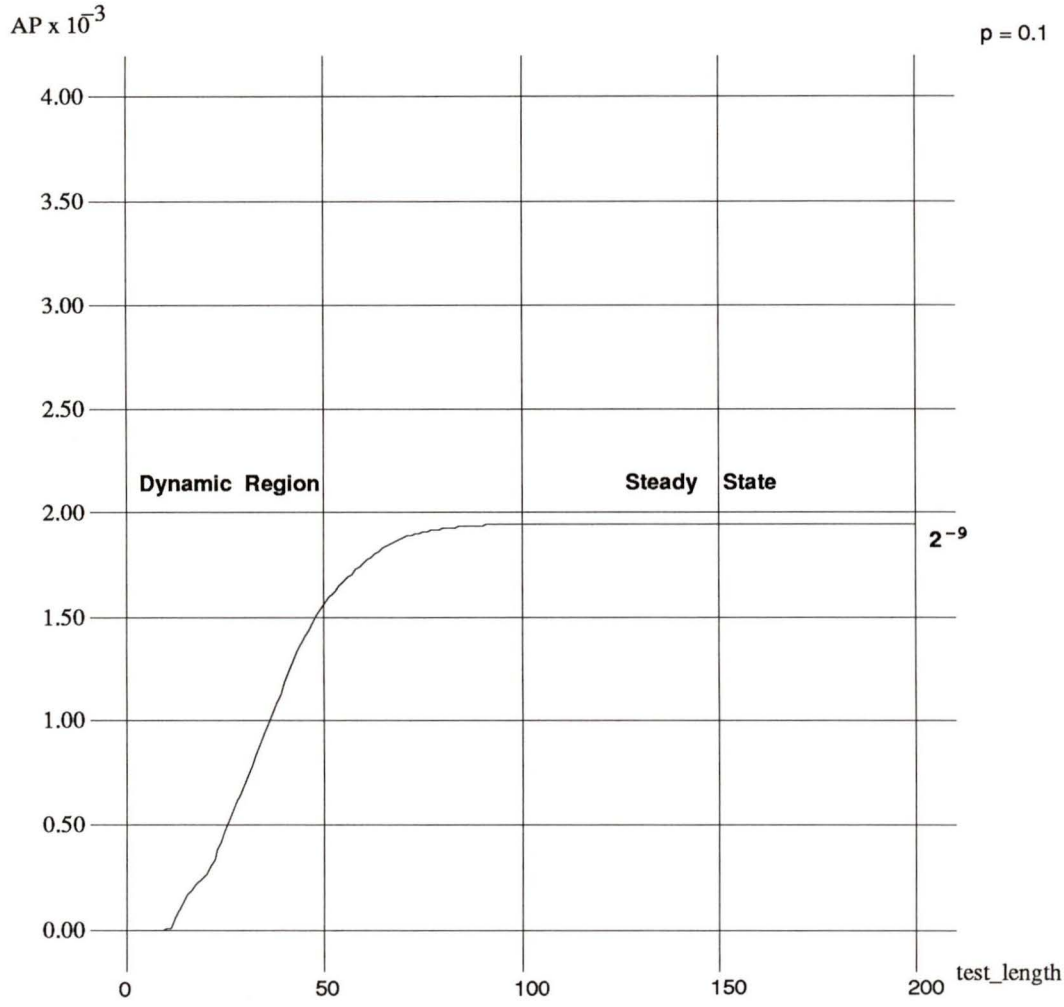


Figure 3.1: Aliasing probability against test length for the primitive polynomial $x^9 + x^7 + x^4 + x^2 + 1$.

Both Damiani and Williams characterized the polynomials only as primitive and non-primitive. In 1990, Robinson [21] proposed a different approach in which he classified the polynomials according to the number of non-zero coefficients. He suggested that polynomials with fewer peaks in the dynamic region are preferable and concluded that polynomials with half of their coefficients non-zero are the best.

In a recent work, Iwasaki and Yamaguchi [11] present a new insight into calculating the AP. The aliasing probability in signature analysis with G(x) as the feedback polynomial is the same as the probability of an error being undetected in the cyclic code with G(x) as the generator polynomial [8]. If the generator polynomial is primitive, the generated code has distance three [11]. It is shown in [11], that the generator polynomials which generate codes of distance greater than three, have lower undetected error probability or aliasing probability for short test lengths.

According to the *independent error model*, as explained in Chapter 2, all the polynomials have the same aliasing probability, asymptotically. The APs for different polynomials, however, behave differently in the dynamic region. Therefore, all the previous research, described above, concentrates on the behavior of the dynamic region. In this exposition, we also investigate the dynamic region to predict the efficiency of a polynomial for practical testing. In the following section, a new approach to determining low aliasing polynomials is presented. Later, in Chapter 5, results from some real circuits simulations are shown to determine the significance of this approach.

3.2 Statistical Approach to The Identification of Low Aliasing Polynomials

It appears from past results that although the primitive polynomials are 'generally' better, there are some cases when they perform worse than non-primitive polyno-

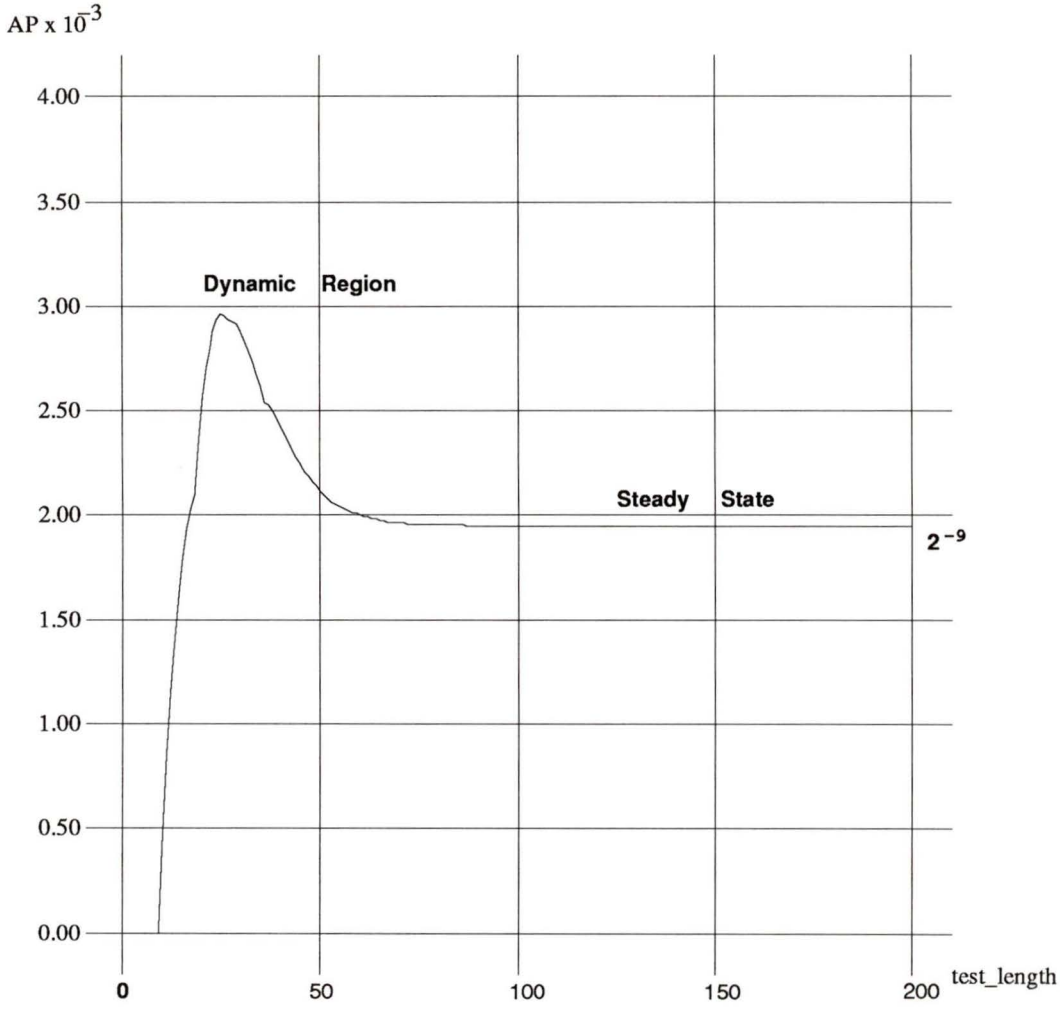


Figure 3.2: Aliasing probability against test length for the *bad* primitive polynomial $x^9 + x^4 + 1$.

mials. This is observed in [1] by some real circuit simulation. By examining the dynamic regions of different primitive polynomials, one can see that they are not all the same. Both figure 3.1 and 3.2 show AP graphs for primitive polynomials for $p = 0.1$. In figure 3.1, AP goes smoothly from the dynamic region to the steady state without ever going above the asymptotic value, while the dynamic region in figure 3.2 has a peak exceeding the asymptotic value.

In this thesis, the polynomials whose aliasing probabilities never go above 2^{-k} are termed as *good* polynomials, while the polynomials with aliasing probabilities greater than 2^{-k} are called *bad*. Since the aim of our approach is to characterize the *good* polynomials, it is not enough to divide the polynomials into only two groups of *primitives* and *non-primitives*. The primitive polynomials are investigated and further divided into groups based on the following criteria.

Roots: Based on the linear dependency of the roots of the polynomials, the primitive polynomials are divided into four categories — E, F, G, and H. Details are explained in Chapter 4.

Weights: Depending on the number of non-zero coefficients, the polynomials can be categorized as low, medium, and high weight polynomials. This categorization is also used by Robinson [21].

Transitions: The transition count of the binary representation of a polynomial is used to classify the polynomials as low, medium, and high count polynomials.

Clusters: The distribution of non-zero coefficients is not the same for all polynomials. The non-zero coefficients may form one or more clusters or groups. Thus, polynomials can be further divided based on the number of clusters they have.

Statistics of *good* and *bad* polynomials for each of the above categories are studied to characterize the *good* polynomials. Since it is a very time intensive operation to

run the algorithm of section 2.8.3 to check whether a polynomial is *good* or *bad*, all the lower degree ($\leq$ 16) and a subset of the higher degree (17-19) primitive polynomials are investigated. In Chapter 4, we explain the procedures used to conduct the experiments, tabulate the results, and discuss the statistical significance of the results obtained.

Chapter 4

Statistical Results

The behavior of the aliasing probability curve in the dynamic region directly depends on the value of p, where p is the probability that each output bit from the CUT is in error. The AP graphs for different values of p for the polynomial $x^5 + x^2 + 1$ are illustrated in figure 4.1. Selecting a value for p is, therefore, very important for investigating the dynamic region behavior of the AP graph. In the experiments, described in this chapter, p is selected as 0.1 for the following reasons:

(i) from the empirical data, it is observed that most polynomials have AP peaks greater than 2^{-k} for $p >> 0.1$, while the dynamic region, in general, remains smooth and below the asymptotic value of 2^{-k} for $p << 0.1$. Thus, $p = 0.1$ is a good choice for comparing two different polynomials by looking at their dynamic region,

(ii) for practical situations, $p = 0.1$ or 10% error rate seems reasonable. (Some trials with other values of p have also been done.)

The reciprocal of a primitive polynomial is also primitive [18]. It is shown in Appendix A that the reciprocal of a primitive polynomial has exactly the same AP as the original. Therefore, only the original primitive polynomials are investigated

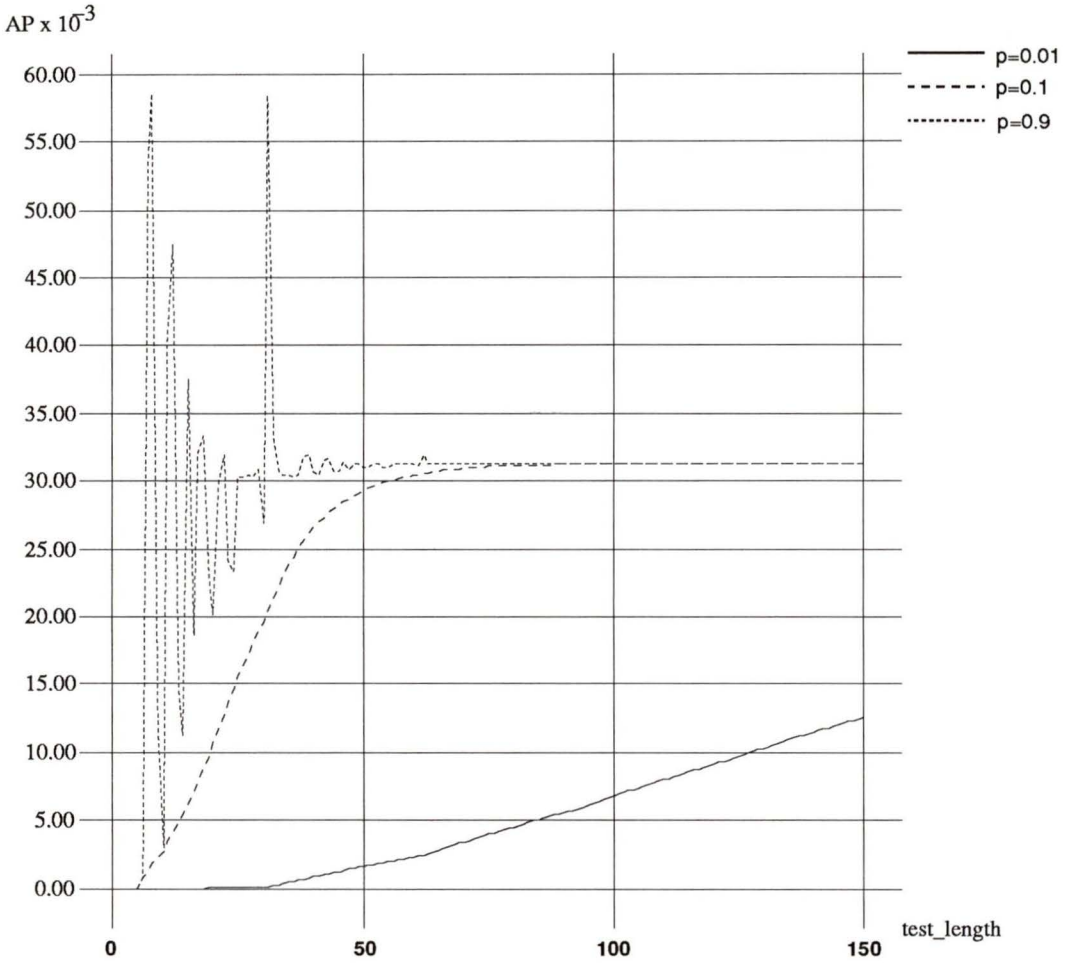


Figure 4.1: AP graphs for different values of p for the primitive polynomial x^5+x^2+1.

in this thesis.

A primitive polynomial is marked as *good* or *bad* by the following procedure:

Begin
  Mark the polynomial as *good*;
  $dynamic\_length$ = estimate the length of the dynamic region;
  $test\_length = 1$;
  While (polynomial is *good*) and ($test\_length < dynamic\_length$) do
  Begin
    Calculate AP;
    If ($AP > 2^{-k}$) then mark the polynomial as *bad*;
    $test\_length = test\_length + 1$;
  end;
end.

In the above algorithm, the length of the dynamic region, $dynamic\_length$, is estimated by plotting the AP graph against test length for a few polynomials. The length of the dynamic region has been found to be approximately the same for all the polynomials for a given p and for $p = 0.1$ it is estimated as 150 vectors. To be on the safe side, experiments are conducted for 200 test vectors.

The statistical results based on the classification of the primitive polynomials, described in Chapter 3, are discussed below. These results are used for the selection of primitive polynomials to reduce aliasing which is the topic of section 4.5. Later, in Chapter 5, a few benchmark circuits are simulated to evaluate this selection.

4.1 Statistics Based On the Degrees of the Polynomials

The percentage of *bad* polynomials (PBP) for each of the degrees 10-19 is calculated and plotted against the degrees as shown in figure 4.2. The PBP graph rises almost linearly up to degree 14, then stays between 32-34 for degrees 14, 15 and 16. The graph rises again for degrees 17, 18 and 19. While all the primitive polynomials up to degree 16 are considered, only a subset of the total number of polynomials for the higher degrees are investigated. Four hundred for each of degrees 17 and 18, and two hundred primitive polynomials for degree 19 are investigated. For those degrees we know the total number of polynomials, but not the distribution. The investigation shows that 30-40% of the primitive polynomials for degrees 16-19 do not have desired behavior for signature analysis.

4.2 Statistics Based On the Roots of the Polynomials

The primitive polynomials can be divided into four categories depending on the roots: An explanation of polynomials and their roots in the galois fields can be found in [18] and is beyond the scope of this work. The classes are:

E The roots of both the original and the reciprocal are linearly dependent.

F The roots of the original are linearly dependent but the roots of the reciprocal are linearly independent.

G The roots of the original are linearly independent but the roots of the reciprocal are linearly dependent.

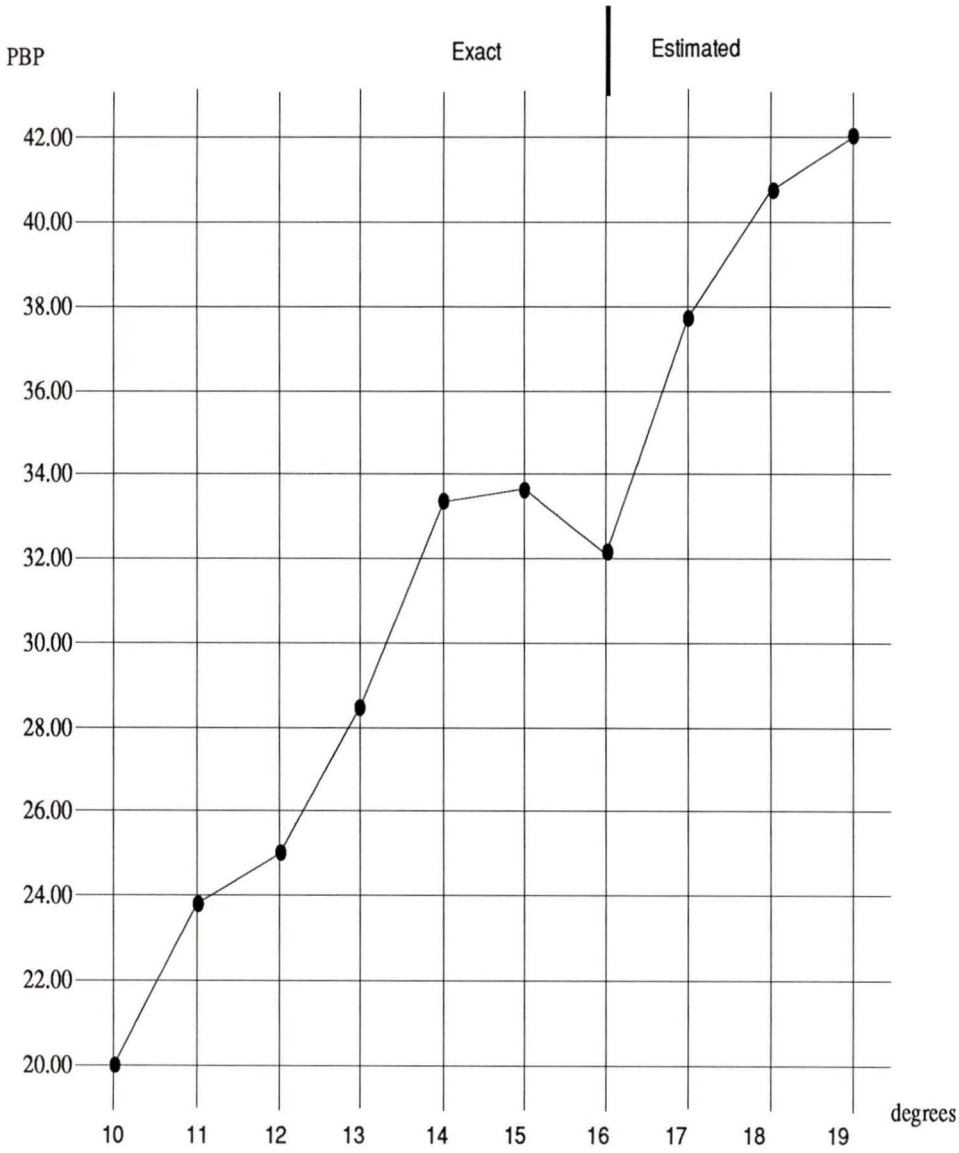


Figure 4.2: Percentage of *bad* primitive polynomials for various degrees.

Types	Number of Originals	Number of Bad Polynomials	Percentage of Bad Polynomials
E	970	332	34.23
F	729 + 650 = 1379	224 + 190 = 414	30.02
G	650 + 729 = 1379	190 + 224 = 414	30.02
H	508	153	30.12
	Total=2857	Total=899	

Table 4.1: Percentage of *bad* polynomials of up to degree 16 for different types of primitive polynomials based on their roots.

H The roots of both the original and the reciprocal are linearly independent.

The classification according to roots is useful in other applications of coding theory to produce particular types of codes with desired properties. Hence it is conjectured that these same properties may be of value in signature analysis [7].

Percentage of *bad* polynomials (PBP) for each of the above categories is calculated. Table 4.1 shows the results for all the original primitive polynomials exhaustively for degree 16 or less. From the above characterization it is clear that the reciprocals of F are in G and vice versa. Hence when examining polynomials of categories F and G, we obtain identical results as can be seen in table 4.1.

Type E polynomials have higher overall PBP than the rest. For each type, the PBP is plotted against the degree of the polynomial to obtain the graph as shown in figure 4.3. The graph for type E remains above the others for most of the degrees.

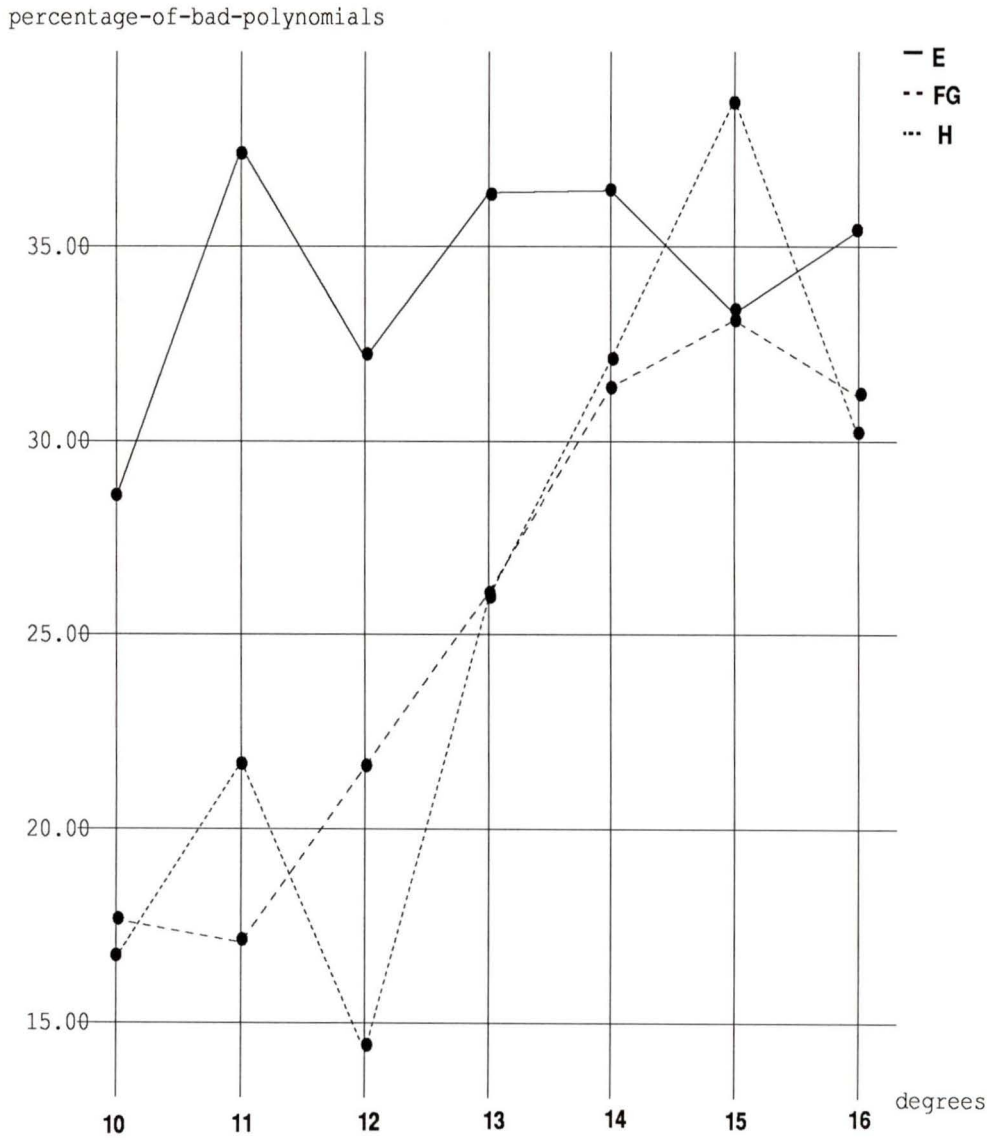


Figure 4.3: Percentage of *bad* polynomials for different types (E, F, G, H) for various degrees of polynomials.

6	7	8	9	10	11	12	13	14
3*	3*	3*	3*	3*	3*	3*	3*	3*
5**	5**	5**	5**	5*	5*	5*	5*	5*
7***	7***	7**	7**	7**	7**	7**	7**	7**
		9***	9***	9***	9***	9**	9**	9**
				11***	11***	11***	11***	11**
						13***	13***	13***
								15***

Table 4.2: Possible weights for primitive polynomials of degree 6-14.

4.3 Statistics Based On Weights

The weight, w, of a polynomial can be defined as the number of non-zero coefficients in that polynomial. For example, the polynomial x^4+x+1 (10011) has weight three. All the primitive polynomials have the following two properties regarding the non-zero coefficients [18]:

1) the highest and the lowest order coefficients are always non-zero,

2) total number of non-zero coefficients is odd.

From the above properties, it is clear that the weight of a primitive polynomial is always odd and the minimum possible weight is three. Also notice that there are $\lfloor k/2 \rfloor$ possible weights for a primitive polynomial, where k is the degree of the polynomial. For example, a primitive polynomial of degree 9 has $\lfloor 9/2 \rfloor = 4$ possible weights — 3, 5, 7, or 9. Possible weights for primitive polynomials of degree 6-14 are listed in table 4.2.

We divide the primitive polynomials into 3 classes with the following objectives:

Weights of Polynomials	Total Number of Polynomials	Number of Bad Polynomials	Percentage of Bad Polynomials
Low	357	127	35.57
Medium	2306	711	30.83
High	194	61	31.44
	Total=2857	Total=899	

Table 4.3: Percentage of *bad* polynomials up to degree 16 for different polynomials based on their weights for $p = 0.1$.

(a) the lowest and the highest class has the same number of possible weights,

(b) the difference between the numbers of possible weights for the medium class and the lowest (or the highest) class is minimum.

The following three classes are defined to satisfy the above two objectives.

Low: $w \leq (2\lfloor\frac{k+2}{6}\rfloor + 1)$,

High: $w \geq (k + 2 - 2\lfloor\frac{k+2}{6}\rfloor)$,

Medium: others.

In table 4.2, the weights marked with single, double, and triple asterisks fall into *Low*, *Medium*, and *High* classes respectively.

The PBP for each of the above categories for all the primitive polynomials of degree 16 or less, is shown in table 4.3. Medium weight polynomials have less overall PBP than others. PBP is plotted against each possible weight for the primitive polynomials of degree 14, 15, and 16 and they are shown in figures 4.4, 4.5, and 4.6 respectively. From the graphs it appears that extreme weight (very low or very high) polynomials are more likely to be *bad* than others.

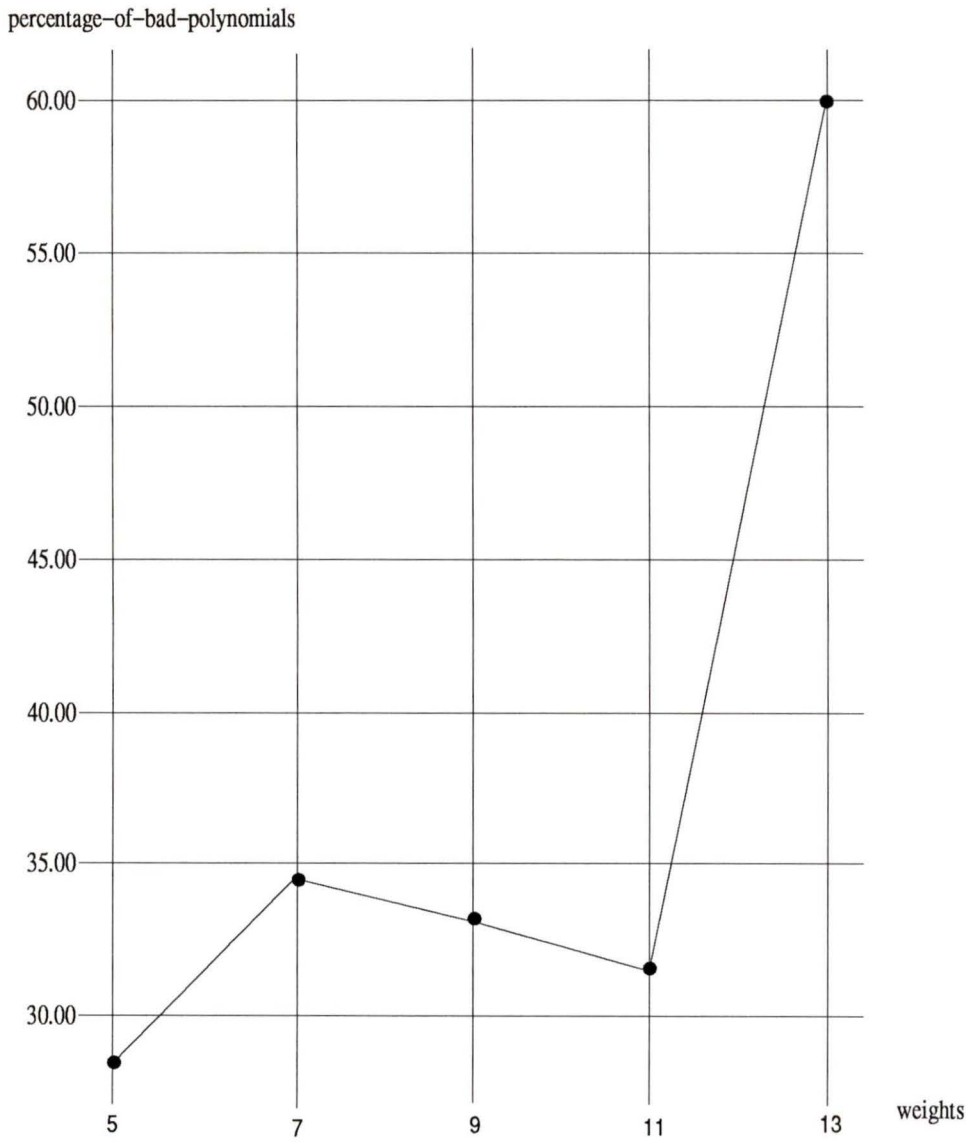


Figure 4.4: Percentage of *bad* polynomials for all possible weights for degree 14 primitive polynomials for $p = 0.1$. There are no primitive polynomial of weight 3 and 15 for degree 14.

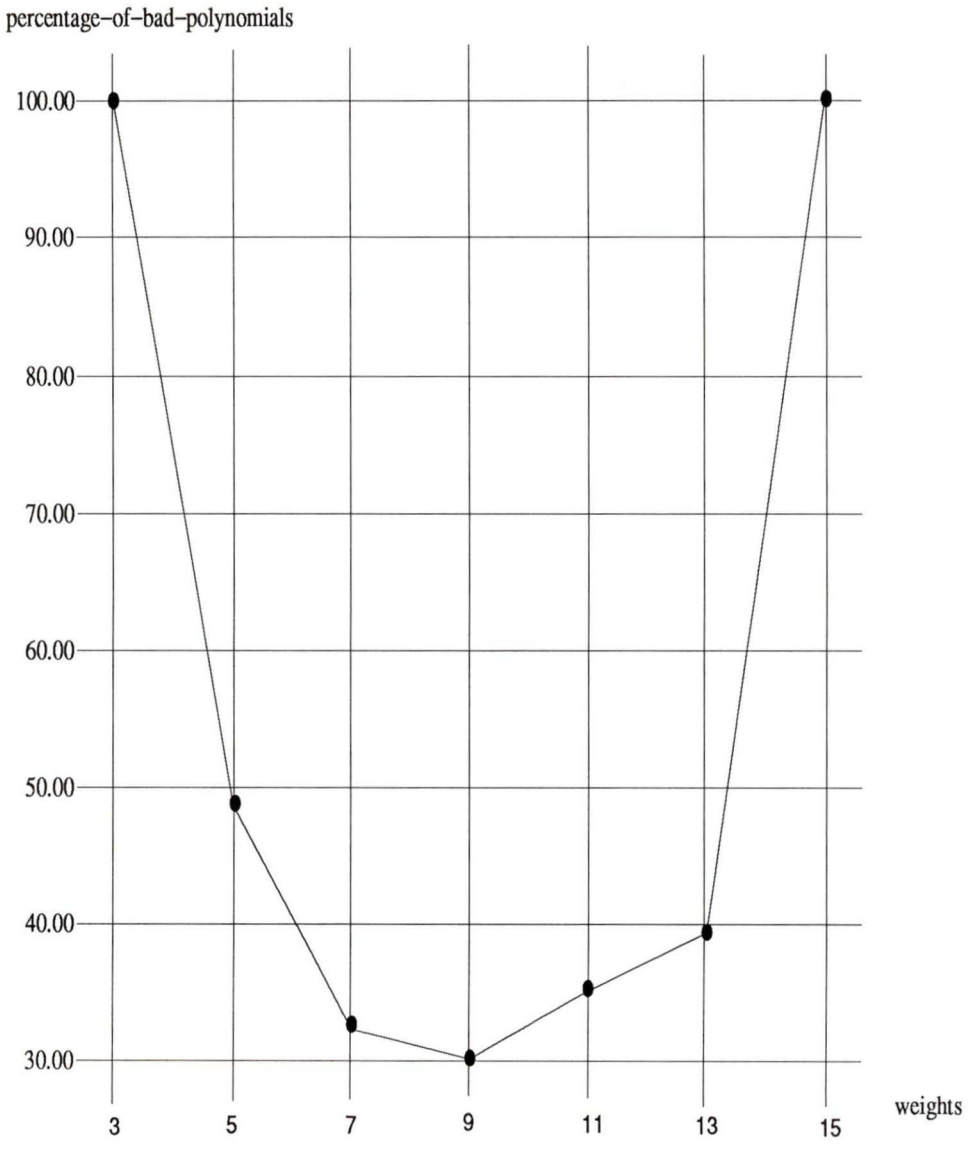


Figure 4.5: Percentage of *bad* polynomials for all possible weights for degree 15 primitive polynomials for $p = 0.1$.

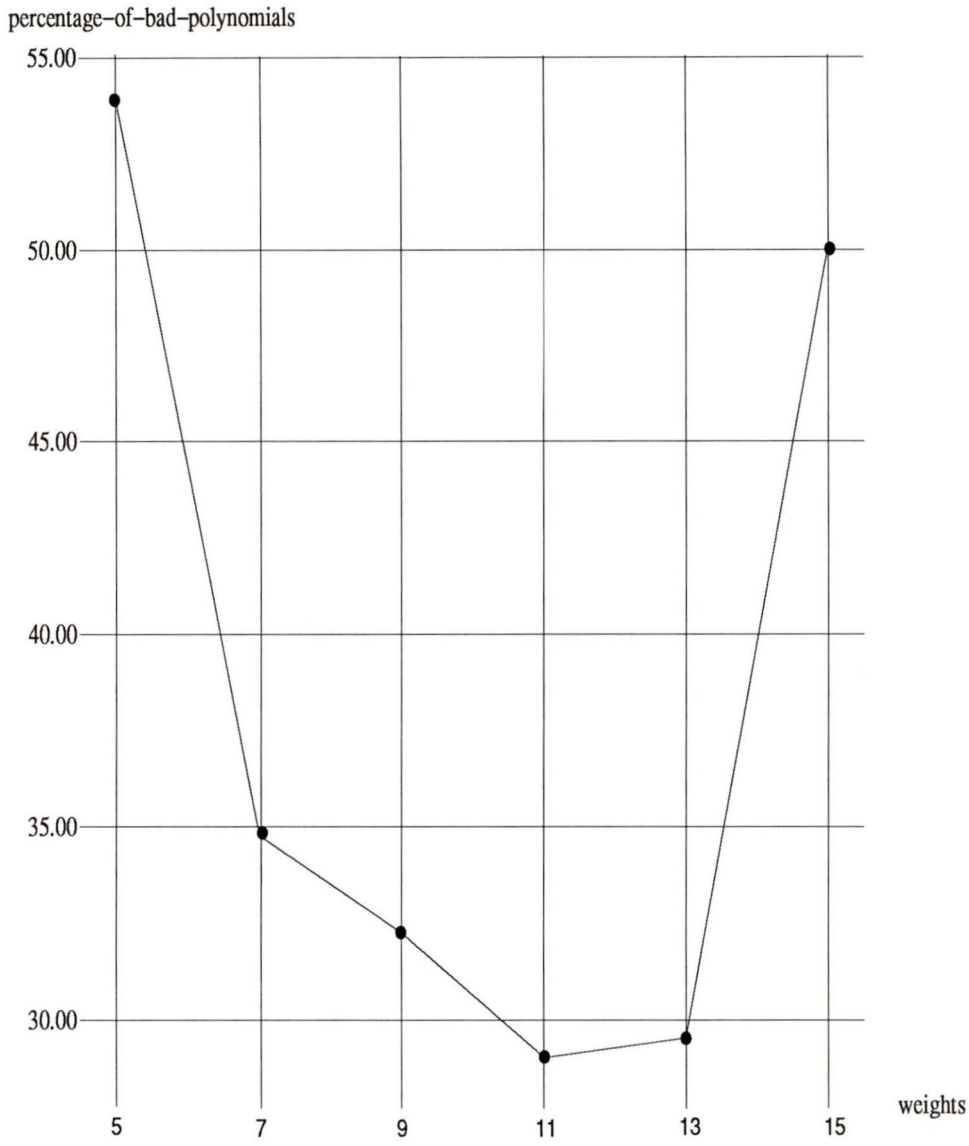


Figure 4.6: Percentage of *bad* polynomials for all possible weights for degree 16 primitive polynomials for $p = 0.1$. There are no primitive polynomials for weight 3 and 17 for degree 16.

4.4 Statistics Based On Distribution of Non-zero Coefficients

The distribution of non-zero coefficients is not the same for all the primitive polynomials. Sometimes all the non-zero coefficients form one or more clusters, sometimes they are scattered. Two different categories, transition count and cluster count, which depends on the distribution of the non-zero coefficients, are described below.

4.4.1 Transition Count

The transition count, t, of a polynomial can be defined as the number of transitions (0 to 1, or 1 to 0) in the binary representation of that polynomial. For example, the polynomial $x^7 + x^5 + x^2 + 1$ (10100101) has transition count six.

From the two properties of primitive polynomials, stated in the previous section, it follows that the transition count, t, of a primitive polynomial is always even. Unlike the weight distribution, the minimum possible t and the total number of possibilities is not the same for the primitive polynomials of odd and even degree. Possible transition counts for primitive polynomials of degree 6-14 are listed in table 4.4.

The following three classes are defined to satisfy the same two objectives stated in the previous section for weight distribution.

T_Low: $t \leq \begin{cases} 2\lfloor \frac{k+2}{6} \rfloor & \text{if } k \text{ is odd} \\ 2\lfloor \frac{k+4}{6} \rfloor - 2 & \text{if } k \text{ is even,} \end{cases}$

T_High: $t \geq \begin{cases} n+1-2\lfloor \frac{k+2}{6} \rfloor & \text{if } k \text{ is odd} \\ n+2-2\lfloor \frac{k+4}{6} \rfloor & \text{if } k \text{ is even,} \end{cases}$

T_Med: others.

6	7	8	9	10	11	12	13	14
0*	2*	0*	2*	0*	2*	0*	2*	0*
2**	4**	2*	4**	2*	4*	2*	4*	2*
4**	6***	4**	6**	4**	6**	4**	6**	4*
6***		6***	8***	6**	8***	6**	8**	6**
		8***		8***	10***	8**	10***	8**
				10***		10***	12***	10***
						12***		12***
								14***

Table 4.4: Possible transition counts for primitive polynomials of degree 6-14.

In table 4.4, the transition counts marked with single, double, and triple asterisks fall into *T_Low*, *T_Med*, and *T_High* classes respectively.

The PBP for each of the above categories for all the primitive polynomials of degree 16 or less, is shown in table 4.5. The PBP for *T_High* is higher than those of other groups and there is not much difference between the PBPs of *T_Low* and *T_Med*. PBP is plotted against each possible transition counts for the primitive polynomials of degrees 14, 15, and 16 and they are shown in figures 4.7, 4.8, and 4.9 respectively. The figures show that like the weights, the polynomials with very high and very low transition counts have more percentage of bad polynomials.

4.4.2 Clusters

The 1s in the binary representation of the polynomial may be together or they may be scattered. A *cluster* can be defined as a group of 1s, containing two or more 1s, without any 0s between them. We do not consider the highest and the lowest order non-zero coefficients, since they are present in every primitive polynomial.

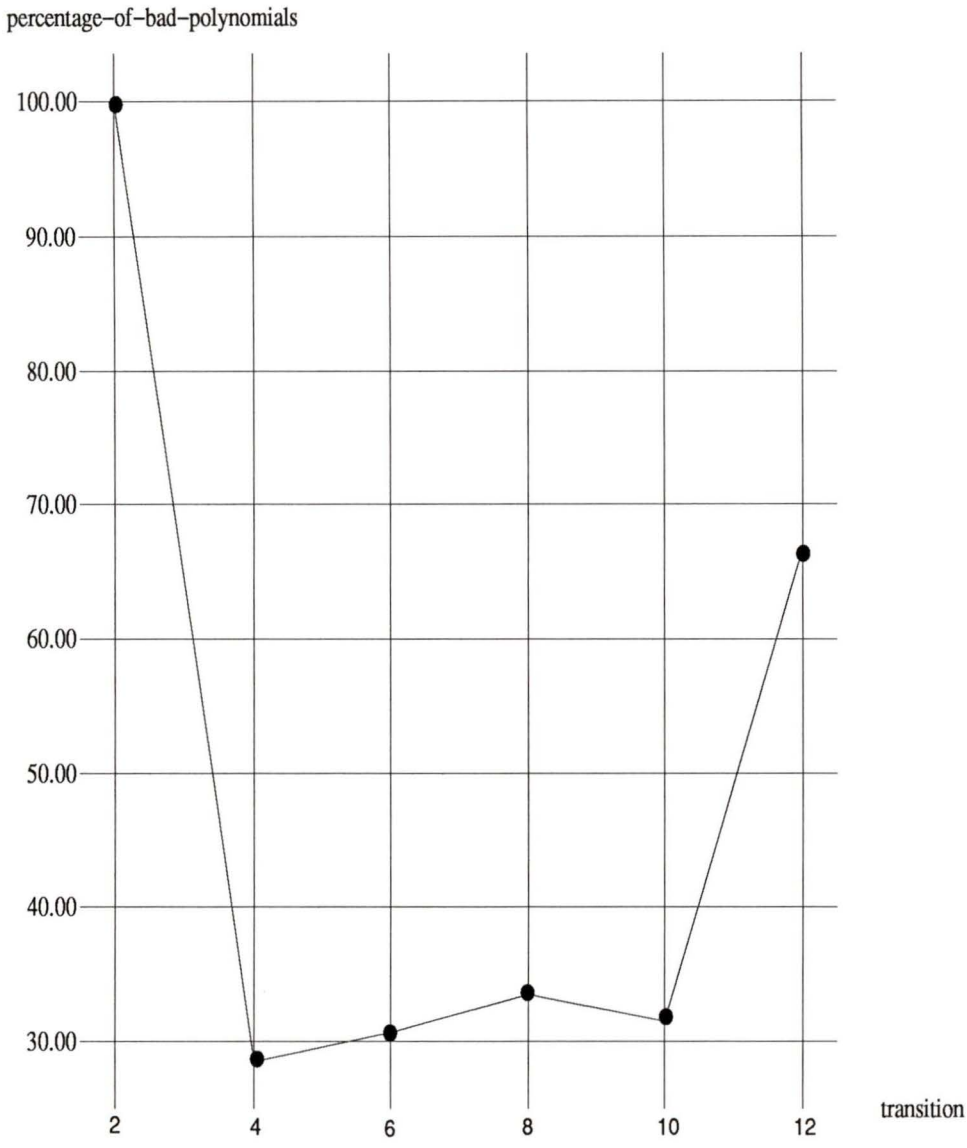


Figure 4.7: Percentage of *bad* polynomials for all possible number of transitions for degree 14 for $p = 0.1$. There are no primitive polynomials with transition count 0 for degree 14.

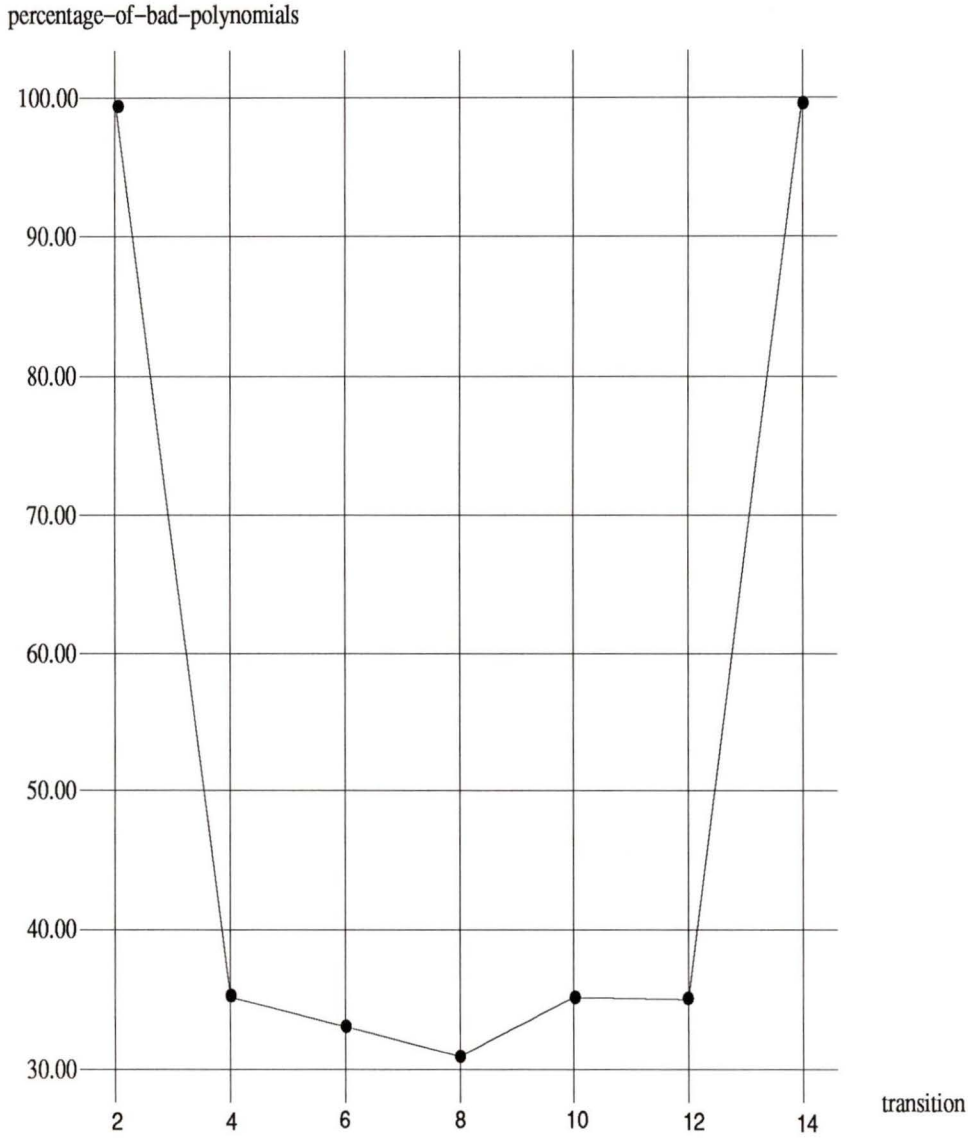


Figure 4.8: Percentage of *bad* polynomials for all possible number of transitions for degree 15 for $p = 0.1$.

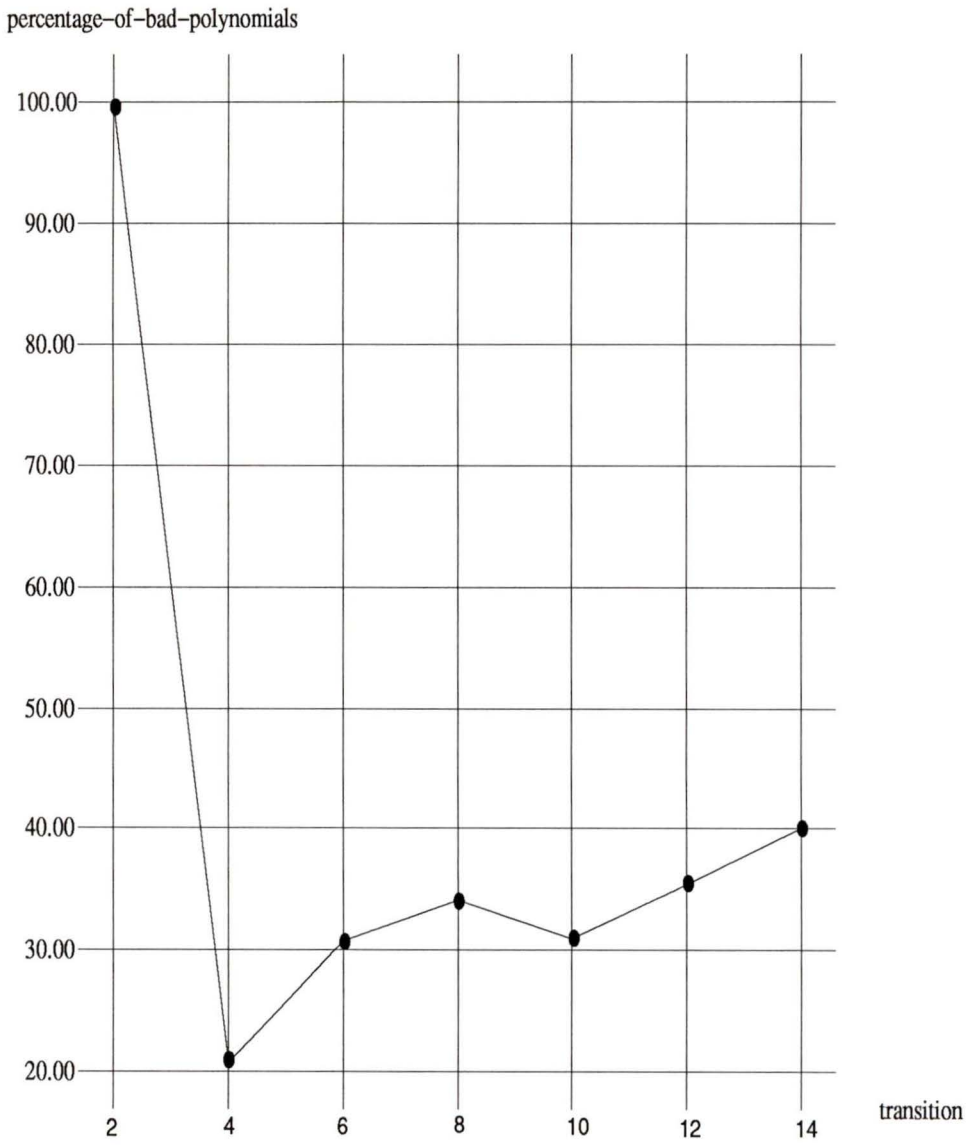


Figure 4.9: Percentage of *bad* polynomials for all possible number of transitions for degree 16 for $p = 0.1$. There are no primitive polynomials with transition count 0 for degree 16.

Transitions	Total Number of Polynomials	Number of Bad Polynomials	Percentage of Bad Polynomials
T_Low	283	83	29.33
T_Med	2378	747	31.41
T_High	195	69	35.35
	Total=2856	Total=899	

Table 4.5: Percentage of *bad* polynomials up to degree 16 for different polynomials based on their transitions for $p = 0.1$. Degree 2 primitive polynomial has no transition.

For example, the polynomial $x^{15} + x^{14} + x^{12} + x^{10} + x^9 + x^7 + x^6 + x^5 + x^2 + x + 1$ (1101011011100111) has 3 clusters. All the primitive polynomials can be divided into the following classes, based on the number of clusters the non-zero coefficients form.

1-cluster: All the non-zero coefficients, except the highest and the lowest order, are contained in exactly one cluster.

2-cluster: All the non-zero coefficients, except the highest and the lowest order, are contained in exactly two clusters.

3-cluster: All the non-zero coefficients, except the highest and the lowest order, are contained in exactly three clusters.

4-cluster: All the non-zero coefficients, except the highest and the lowest order, are contained in exactly four clusters.

Others: The rest.

Distribution of 1s	Total Number of Polynomials	Number of Bad Polynomials	Percentage of Bad Polynomials
1-cluster	22	7	31.82
2-cluster	150	49	32.67
3-cluster	126	33	26.19
4-cluster	31	5	16.13
Others	2528	805	31.84
	Total=2857	Total=899	

Table 4.6: Percentage of *bad* polynomials up to degree 16 for different polynomials based on the distribution of non-zero coefficients.

Notice that according to the above classification, a polynomial with n clusters may not be in *n-cluster* class, if there are some non-zero coefficients which do not belong to any of the n clusters. For example, the polynomial $x^{10} + x^8 + x^6 + x^5 + x^3 + x^2 + 1$ (10101101101) is not in 2-*cluster* class, although it has two clusters. There is one non-zero coefficient which do not belong to any of the two clusters, and thus, the polynomial belongs to *others*.

The PBPs for the above categories are shown in table 4.6. The PBPs for the polynomials with higher number of clusters (3_cluster and 4_cluster) are less than those of lower cluster class (1_cluster and 2_cluster).

4.5 Selection of Better Primitive Polynomials

Summarizing the above results, we conclude that a polynomial is more likely to be *good* if

(1) it is not of type E,

Selections	Total Number of Polynomials	Number of Bad Polynomials	Percentage of Bad Polynomials
choose	1301	398	30.59
avoid	1556	501	32.20
	Total=2856	Total=899	

Table 4.7: Percentage of *bad* polynomials up to degree 16 for the selected and non-selected polynomials for $p = 0.1$.

(2) it has medium weight,

(3) it has medium transition count,

(4) it is in 4_cluster class.

However, there are only 31 polynomials out of all the polynomials exhaustively up to degree 16, which are in 4_cluster. There are no polynomials of degree less than 13 which are in class 4_cluster. Therefore, we can neglect the cluster classification for the selection of better primitive polynomials. Finally, we define two classes — *choose* and *avoid* as follows.

choose: Primitive polynomials of type F, G, or H and with medium weight and medium transition count (in Medium and T_Med class),

avoid: the rest.

In table 4.7, the percentage of bad polynomials for the above groups is shown. We see that the PBP for *choose* is less than the PBP for *avoid*. For any degree, we select a primitive polynomial from class *choose* for signature analysis. Experiments are conducted on some real circuits to evaluate our selection criteria for practical cases and is discussed in Chapter 5.

4.6 Statistical Significance of the Experimental Results

The results and the observations obtained in the previous sections are all based on the primitive polynomials of degree 16 or less. The obvious question is — what is the validity of these statistical results for the polynomials of higher degrees?

There are standard statistical procedures to deal with this kind of questions. Unfortunately, the model of our experiment does not fit into those procedures, and, therefore, we can not provide any specific confidence level for our results to be applicable for higher degree polynomials. The following subsection briefly describes two standard statistical methods and explains why they can not be applied to our model.

4.6.1 Standard Estimation Methods

Since it is generally impractical to measure the behavior of all the individuals in a large population, the investigation is usually confined to a small sample of subjects from that population. The objective is to estimate the behavior of the whole population based on the results obtained from the sample. Two different kinds of estimations are — point estimation, and interval estimation [16]. In the point estimation, a specific value for a parameter of the population is estimated from the sample. On the other hand, an interval for a parameter of the population is calculated with some confidence level attached to it in the interval estimation. Both of these methods require that the sample be random. We have not found any way to estimate the behavior of the population when the sampling is not random. It is shown below, why random sampling is not possible in our experimental model.

4.6.2 Sampling Method of Our Experiment

Since it is impractical to examine a polynomial of higher degree (> 19) due to the exponential complexity of the algorithm, it is not possible to sample randomly from the infinite number of polynomials for all degrees. Thus our sample consists of all the primitive polynomials exhaustively up to degree 16 and some random samples from degree 17-19. It is, therefore, a sequential sampling and the methods of estimation described in section 4.6.1 can not be applied for the higher degrees.

Chapter 5

Simulation

In Chapter 4, the aliasing probability (AP) is calculated under the assumption that every output bit from the CUT is in error with a constant probability p (independent error model). No circuit structure and no fault model is required for this probabilistic approach. The classification of *choose* and *avoid*, as defined in chapter 4, is based on the results of this error model. However, in practice faults in a circuit may not follow the model and, consequently, the selection of Chapter 4 may not produce the desired results. The purpose of this Chapter is to simulate some benchmark circuits to evaluate the selection criteria suggested in Chapter 4.

Given the circuit description, it is possible to simulate the output response of a fault free circuit for any input vector or set of vectors. One can also inject a fault of any type (for example a stuck-at fault) and simulate the faulty response. If the fault is undetectable, the output response of the faulty circuit is exactly the same as that of the fault-free circuit for all input vectors and they both produce the same signature. We do not consider these faults for aliasing. If, however, the faulty response is different from the fault-free response and they both have the same signature, we know that the fault causes aliasing.

Given a signature analyzing register (or its equivalent polynomial), we can cal-

t	AP(t)
1	0.01
2	0.00
3	0.00
4	0.02

Table 5.1: AP(t) for different values of t.

culate exactly the aliasing probability (AP) for a given circuit and fault model for any test length and for any particular test vector sequence by simulation. If, for test length t, N is the number of faults considered, for A of them the signature is the same as in the fault-free case, and U of them are undetectable, then (N-U) is the number of detectable faults and (A-U) of them cause aliasing. Thus, we have, AP(t) = (A-U)/(N-U) for the particular sequence of test vectors applied to the circuit. If we apply a different sequence of test vectors, we may obtain a different AP(t).

To simulate a circuit, a program called 'sig' [1] is used. Given a circuit, any test length t, and the initial test pattern, this program generates t pseudo-random test vectors and calculates the AP(t) for each test length, 1 to t. Two different measures are performed on the AP(t) and are discussed in the following sections.

5.1 Average Aliasing

If we add all the individual AP(t)s for each test length, 1 to t, and take the average value, we obtain the average aliasing probability (AAP). Thus, we have,

$$AAP = \frac{\sum_{i=1}^{t} AP(i)}{t}.$$

[1] This program is written by Mr. Zhang, research associate of VLSI group in Uvic. A 'man' page for 'sig' is provided in Appendix B.

In the example of table 5.1, the AAP is $(0.01 + 0.00 + 0.00 + 0.02)/4 = 0.0075$. Different polynomials implementing signature analyzers, have different aliasing probabilities and thus different AAPs. The AAPs for the polynomials of classes *choose* and *avoid* for different circuits are calculated and discussed in section 5.3.

5.2 Percentage of Test Lengths Having Aliasing

While calculating the AP(t) for each test length, 1 to t, we notice that not all the test lengths have aliasing. In this measure, the percentage of all the test lengths (PP) having aliasing (AP(t)>0) is calculated. In the example of table 5.1, only two test lengths out of 4, AP(1) and AP(4), have any aliasing. Thus, PP is calculated as $(2 \times 100)/4 = 50\%$. The average PP (APP) for a class of n polynomials can be calculated as

$$APP = \frac{\sum_{i=1}^{n} PP_i}{n},$$

where PP_i represents the PP for the ith polynomial in the class. The APPs for classes *choose* and *avoid* for different circuits are calculated and shown in section 5.3.

5.3 Simulation Results

Three circuits are simulated for all possible single stuck-at and single delay faults for test length 512 [2] . Each of these circuits has multiple outputs. Each output is simulated separately.

Three different implementations are available for generators and data compactors — type 1 LFSR, type 2 LFSR, and CAR. It is shown in section 2.6 that all three of

[2]512 is used so that simulation time does not become prohibitive but still illustrates the average aliasing behavior

these structures are equivalent for aliasing when used as data compactors. However, they may have different results when used as pseudo-random pattern generator.

For our benchmark simulations, two different experiments under different combination of generators and compactors are used. In the first experiment, the standard circuit SN74181 is simulated. It has 14 inputs, 8 outputs, and 226 gates. The generator used is type 1 minimum cost LFSR and the compactor is type 2 LFSR. Total 466 single stuck-at and 466 single delay faults are simulated for this circuit.

In the second experiment, circuits sao2.pla and misex3c.pla (from the Berkeley PLA benchmarks) are simulated with the generator implemented by the CAR similar to the minimum cost LFSR and the compactor by type 1 LFSR. The circuit sao2.pla has 10 inputs, 4 outputs, and 58 products. Total 158 single stuck-at and 158 single delay faults are simulated for this circuit. The circuit misex3c.pla has 14 inputs, 14 outputs, and 305 products. Total 668 single stuck-at and 668 single delay faults are simulated for this circuit.

For both of the above experiments, all the original primitive polynomials exhaustively up to degree 16 are considered as signature analyzers. For each compactor (polynomial) one sequence of 512 test vectors, generated with the initial pattern $100\ldots0$, is applied and it is the same for all the polynomials.

The aliasing probability is directly related to the size of the signature register or the degree of the polynomial representing the register. The higher the degree, the lower the average aliasing probability (AAP) and the percentage of patterns having aliasing (PP). This is shown in figures 5.1 and 5.2. In figure 5.1, for each degree, 2 to 16, an average AAP is calculated and plotted against the degree. Three different graphs, representing three different circuits, are shown. All of them show higher AAPs for lower degrees. Similarly, in figure 5.2, it is shown that lower degrees have higher average PP (APP). Since the distribution of different classes of polynomials are not uniform among the degrees, it is necessary to calculate statistics for each

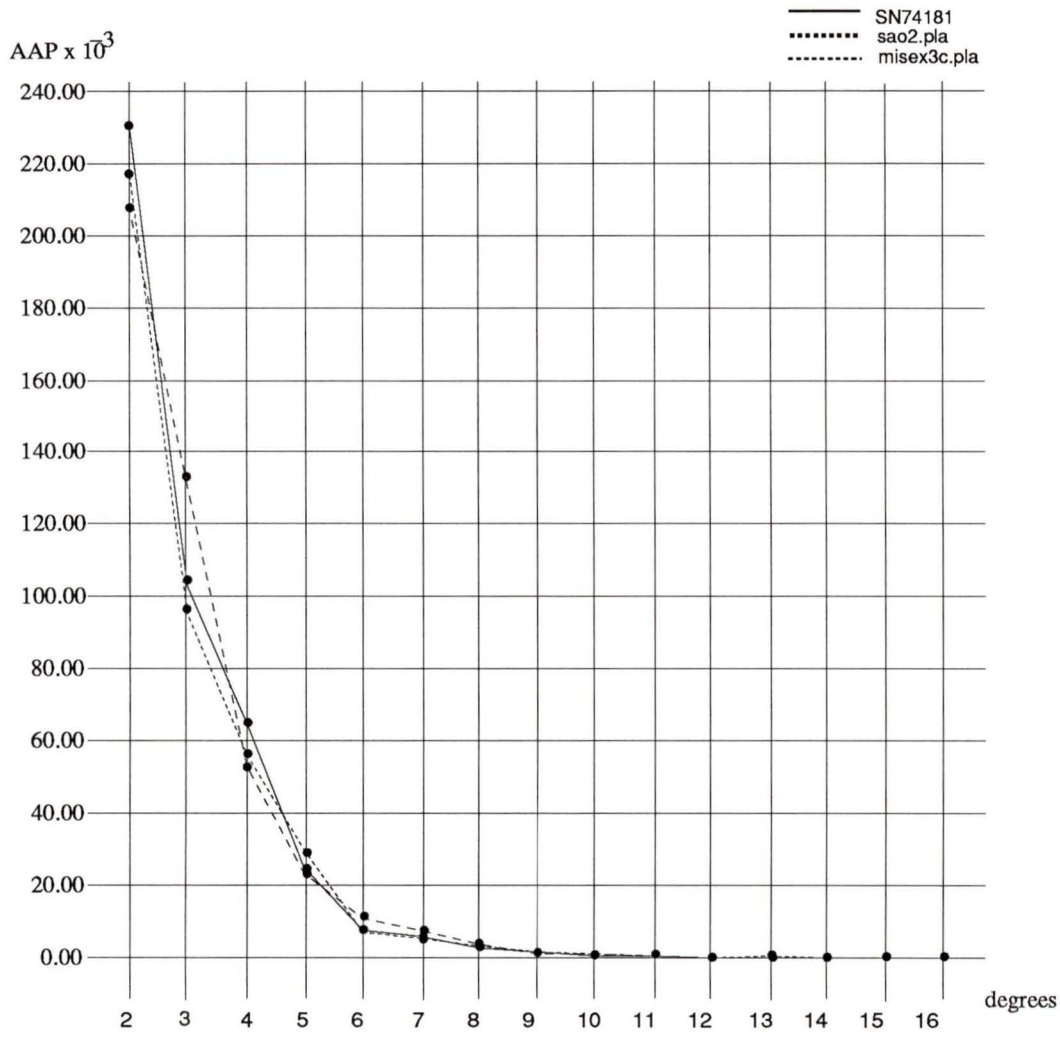


Figure 5.1: AAPs for various degrees of primitive polynomials for output 1 of the circuits SN74181, sao2.pla, and misex3c.pla for stuck-at faults.

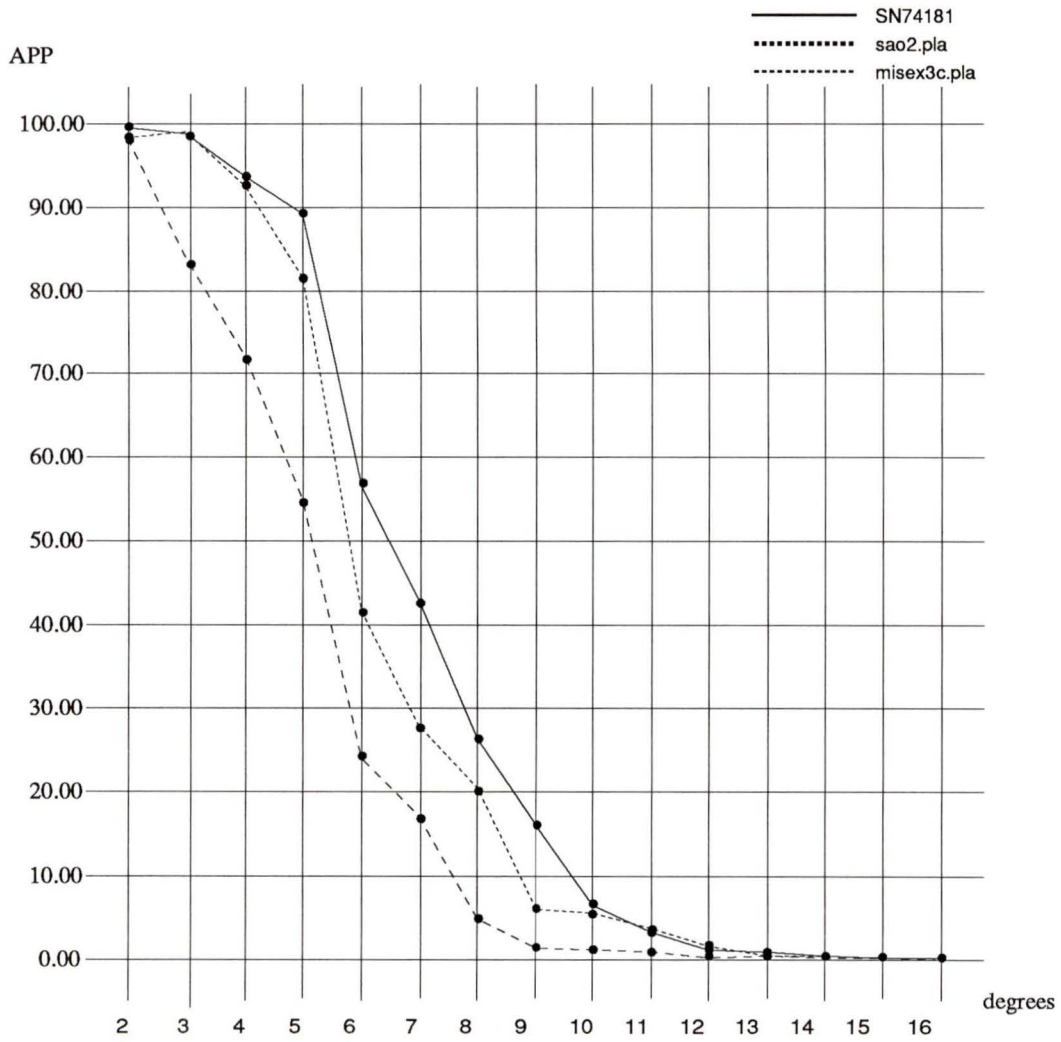


Figure 5.2: APPs for various degrees of primitive polynomials for output 1 of the circuits SN74181, sao2.pla, and misex3c.pla for stuck-at faults.

degree separately.

The AAPs for all the polynomials in classes *choose* and *avoid* are added up and the average AAP is calculated for each class. Similarly, the APP for each class is calculated. The results for degrees 14-16 are tabulated in tables 5.2-5.4. In these tables, the results of individual outputs as well as the average results for each circuit are shown. For example, the 5th row of circuit SN74181 in table 5.2 shows that the AAPs for the *choose* and the *avoid* class for output 5 is 4.70×10^{-5} and 5.90×10^{-5} respectively for the stuck-at faults. The average AAPs for the above classes for all the 8 outputs are 4.06×10^{-5} and 5.41×10^{-5} respectively. This shows that the selection actually reduces aliasing for the circuit SN74181 for the stuck-at faults. Similarly, in table 5.2, all the outputs of sao2.pla, except output 1, show lower AAPs for the stuck-at faults for the selected polynomials and in the average the AAPs are 3.50×10^{-5} and $4.98 \times 10{-}5$ for the classes *choose* and *avoid* respectively for degree 14 polynomials.

5.4 Significance of the Results

Most of the outputs of the circuit SN74181 show lower aliasing for the selected polynomials and the average aliasing for the selection is always lower for degrees 14, 15 and 16. The average aliasing for the circuit sao2.pla is always lower except for two cases. For degree 15 and delay faults, both the AAP and the APP of the *choose* class are greater than those of *avoid* class. For degree 16 and stuck-at faults, the AAP for the *choose* is greater than that of *avoid* class.

The selection does not reduce aliasing for most of the cases for circuit misex3c.pla. The possible error patterns in the outputs of this circuit may not be close to the error model on which the theory behind our selection is based.

We obtain the above results by applying one of the many possible sequences of

Circuits	Outputs	Single stuck-at faults				Single delay faults			
		choose (Total=180)		*avoid* (Total=198)		*choose* (Total=180)		*avoid* (Total =198)	
		AAP $\times 10^{-5}$	APP $\times 10^{-1}$	AAP $\times 10^{-5}$	APP $\times 10^{-1}$	AAP $\times 10^{-5}$	APP $\times 10^{-1}$	AAP $\times 10^{-5}$	APP $\times 10^{-1}$
SN74181	output 1	3.50	4.49	5.10	5.41	6.30	6.13	2.70	3.68
	output 2	4.30	3.92	5.00	5.65	2.60	3.75	2.90	4.77
	output 3	2.80	1.87	2.60	2.01	3.80	2.92	4.70	3.11
	output 4	4.20	5.70	5.90	6.48	4.20	5.51	6.40	8.44
	output 5	4.70	5.27	5.90	7.05	5.00	8.00	5.70	5.49
	output 6	3.60	7.31	3.80	8.42	3.20	7.83	4.00	9.28
	output 7	6.30	3.07	6.80	3.46	4.20	4.19	4.80	3.95
	output 8	3.10	1.36	8.20	2.56	6.70	2.64	6.20	2.67
	average	4.06	4.12	5.41	5.13	4.50	5.12	4.68	5.17
Sao2.pla	output 1	3.50	0.91	4.20	0.42	3.00	1.64	8.50	4.65
	output 2	1.40	0.31	6.50	0.51	8.00	4.84	7.50	4.23
	output 3	1.20	0.03	5.60	1.62	0.00	0.00	0.00	0.00
	output 4	7.90	2.66	3.60	1.59	0.00	0.00	0.00	0.00
	average	3.5	0.98	4.98	1.04	2.75	1.62	4.00	2.22
Misex3c.pla	output 1	6.40	6.23	0.90	1.80	4.10	7.83	1.40	2.58
	output 2	3.90	0.09	2.10	0.04	0.00	0.00	0.00	0.00
	output 3	1.90	0.50	2.90	0.84	0.90	0.47	0.60	0.30
	output 4	2.10	0.62	5.20	0.85	7.60	2.77	1.30	0.54
	average	3.58	1.86	2.78	0.88	3.15	2.77	0.83	3.42

Table 5.2: Average AAP and APP for all the primitive polynomials of degree 14.

Circuits	Outputs	Single stuck-at faults				Single delay faults			
		choose (Total=396)		*avoid* (Total=504)		*choose* (Total=396)		*avoid* (Total =504)	
		AAP $\times 10^{-5}$	APP $\times 10^{-1}$	AAP $\times 10^{-5}$	APP $\times 10^{-1}$	AAP $\times 10^{-5}$	APP $\times 10^{-1}$	AAP $\times 10^{-5}$	APP $\times 10^{-1}$
SN74181	output 1	1.80	1.90	1.50	1.12	1.00	1.62	0.80	1.05
	output 2	1.90	2.89	2.60	3.09	2.20	2.75	3.30	5.25
	output 3	1.20	1.15	2.20	1.12	1.50	1.29	1.20	1.00
	output 4	2.30	3.22	2.00	2.79	2.90	3.95	1.50	2.60
	output 5	1.50	1.74	3.10	3.47	1.20	2.07	2.80	3.19
	output 6	2.80	5.81	2.20	4.91	1.60	4.23	2.00	4.83
	output 7	1.80	1.36	2.00	1.60	2.20	1.55	2.50	2.10
	output 8	1.40	0.85	3.10	0.98	2.20	1.21	3.00	1.62
	average	1.84	2.37	2.34	2.39	1.85	2.33	2.14	2.71
Sao2.pla	output 1	3.00	1.30	2.80	0.91	1.60	0.83	0.40	0.24
	output 2	0.80	0.12	1.90	0.95	0.50	0.27	0.60	0.38
	output 3	0.70	0.02	2.20	0.22	0.00	0.00	0.00	0.00
	output 4	0.20	0.01	0.80	0.05	0.00	0.00	0.00	0.00
	average	1.18	0.36	1.93	0.53	0.53	0.28	0.25	0.16
Misex3c.pla	output 1	2.50	2.50	1.70	0.65	1.40	2.79	1.50	2.91
	output 2	1.00	0.02	0.90	0.02	0.00	0.00	0.00	0.00
	output 3	2.30	0.74	2.20	0.74	2.30	0.81	2.30	1.19
	output 4	1.60	0.33	4.50	1.49	2.80	1.11	4.50	1.15
	average	1.85	0.90	2.33	0.73	1.63	1.18	2.08	1.31

Table 5.3: Average AAP and APP for all the primitive polynomials of degree 15.

Circuits	Outputs	Single stuck-at faults				Single delay faults			
		choose (Total=620)		*avoid* (Total=404)		*choose* (Total=620)		*avoid* (Total=404)	
		AAP $\times 10^{-5}$	APP $\times 10^{-1}$	AAP $\times 10^{-5}$	APP $\times 10^{-1}$	AAP $\times 10^{-5}$	APP $\times 10^{-1}$	AAP $\times 10^{-5}$	APP $\times 10^{-1}$
SN74181	output 1	0.80	1.10	0.90	1.20	0.50	1.01	0.90	1.07
	output 2	1.40	0.96	1.90	1.75	1.00	2.12	0.70	1.41
	output 3	1.60	0.83	2.00	1.23	1.50	1.17	1.40	1.24
	output 4	0.90	1.39	2.50	1.56	0.50	1.05	2.20	2.51
	output 5	1.00	1.16	1.30	0.85	0.90	1.11	0.90	1.03
	output 6	1.10	2.19	2.00	1.92	1.60	3.39	2.60	4.51
	output 7	1.30	0.96	1.10	0.69	0.80	0.75	0.90	0.75
	output 8	1.60	0.62	0.90	0.46	1.20	0.47	2.00	0.55
	average	1.21	1.15	1.58	1.21	1.00	1.38	1.45	1.63
Sao2.pla	output 1	1.20	0.08	0.80	0.29	0.90	0.50	0.50	0.33
	output 2	0.60	0.10	0.30	0.05	0.50	0.25	1.30	0.90
	output 3	0.60	0.09	0.60	0.12	0.00	0.00	0.00	0.00
	output 4	0.70	0.19	0.50	0.02	0.00	0.00	0.00	0.00
	average	0.78	0.12	0.55	0.12	0.35	0.19	0.45	0.31
Misex3c.pla	output 1	1.90	1.02	0.60	0.55	1.00	1.91	0.60	1.15
	output 2	0.50	0.01	2.30	0.67	0.00	0.00	0.00	0.00
	output 3	0.80	0.31	0.70	0.12	0.30	0.14	0.20	0.08
	0utput 4	0.50	0.20	0.20	0.05	0.90	0.32	0.80	0.28
	average	0.93	0.39	0.95	0.35	0.55	0.59	0.40	0.38

Table 5.4: Average AAP and APP for all the primitive polynomials of degree 16.

512 test vectors which could be produced by the generator with different initial pattern. Different sequence of test vectors might produce different results for individual test lengths, but our interest is in the average value. It is thought to be a reasonable measure even though based on a small sample size.

For circuit SN74181, the compactor is chosen as type 1 LFSR while for other circuits the similar CAR to type 1 LFSR is used for compaction. As explained in section 2.6, similar CAR to LFSR should have the same aliasing effect. However, different structures for pattern generators may produce different sequence and the randomness of different sequences may vary. The effects of randomness of the input vector sequences on the aliasing may not be known.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

A new approach to the problem of selecting polynomials to reduce aliasing in signature analysis is introduced in this thesis. Primitive polynomials are classified among several classes. All primitive polynomials exhaustively up to degree 16 are analyzed by a Markov process. The polynomials having aliasing probability greater than the asymptotic value are termed as *bad*, and the statistics of the *bad* polynomials for each class are calculated. Certain primitive polynomials with medium weight and medium transition count are found to have the best theoretical behavior. The results suggest that although the lowest weight polynomials have the smallest implementation cost, they do not necessarily have the best performance with regard to aliasing.

Since the results of the above experiments are based on the independent error model, where it is assumed that each output bit from the circuit under test (CUT) is in error with a constant probability p, it is hard to judge the effectiveness of the selection criteria obtained from such results. For any particular circuit, the selection may not work well.

An attempt is made in chapter 5 to evaluate the effectiveness of our selection criteria by simulating a few benchmark circuits. Each circuit is simulated with a single permutation of 512 pseudo-random input test vectors for all the original primitive polynomials, used as data compactors, up to degree 16. The average aliasing probability (AAP) and the average percentage of test patterns (out of 512 patterns) causing aliasing are calculated for the selected and the non-selected primitive polynomials. All the 8 outputs of circuit SN74181 have lower AAP and APP for stuck-at faults for the selected polynomials. For the other circuits, however, the selected polynomials do not have noticeably better behavior than the others.

The selection proposed in this thesis should, in general, reduce aliasing. However, in specific cases we may not obtain desired results.

While the selection criteria are developed by considering polynomials up to degree 16, it is thought that the results are likely to carry over to higher degree.

6.2 Future Work

In the benchmark simulation, we used only one permutation of 512 test vectors for each polynomial. We might get different results for different permutations. A better approach would be to apply w different permutations to each circuit for each polynomial and consider the average value for aliasing.

While defining *bad* primitive polynomials in section 3.2, we considered only the polynomials having aliasing probability greater than the asymptotic value of 2^{-k} in the dynamic region. A more detailed distinction would be to consider the number of peaks above the asymptotic value. Another way to define bad polynomials is to consider the average aliasing probability in the dynamic region.

In all the experiments in this thesis only single input LFSRs are considered. In the future, multiple input LFSRs or MISRs should be investigated to determine if

they produce the same results.

In the theoretical analysis of the aliasing effect of the polynomials, we consider the *independent error model.* In this error model the probability of output bit from the CUT does not change with time. A more practical model, called *exponential error model* [21], may be used, where the bit error probability changes with time. The *correlated error model*, where the error patterns have erroneous bits separated by a span b [25], may also be considered to obtain more accurate selection criteria for the data compaction polynomials.

In recent research [11], the proposal has been made to use polynomials, even other than primitive, which generate some particular codes (in [11], generator of BCH code is examined). It should be possible to examine statistically various other types of polynomial whose theoretical usefulness comes from other domains, and then try to infer further criteria within the context of testing.

Bibliography

[1] A. Ahmad, N.K. Nanda, and K. Garg. Are primitive polynomials always best in signature analysis. *IEEE Design and Test of Computers*, pages 36–38, August 1990.

[2] P. H. Bardell, W. H. McAnney, and J. Savir. *Built-In Self Test for VLSI.* John Wiley and Sons, New York, 1987.

[3] D. R. Barr and P. W. Zehna. *Probability Modeling Uncertainty.* Adison-Wesley, 1983.

[4] M. Damiani, P. Olivio, M. Favalli, and B. Ricco. An analytical model for the aliasing probability in signature analysis testing. *IEEE Transactions on Computers*, 8(11):1133–1144, November 1989.

[5] R.A. Frohwerk. Signature analysis: a new digital field services method. *Hewlett Packard J.*, pages 2–8, May 1977.

[6] P. Goel. An implicit enumeration algorithm to generate tests for combinational logic circuits. *IEEE Transaction on Computer*, 30(3):215–222, 1981.

[7] A. Gulliver, M. Serra, and V. K. Bhargava. Primitive polynomials with independent roots and their applications. *Canadian Conference on Electrical and Computer Engineering*, pages 818–822, November 1988.

[8] S. K. Gupta and D. K. Pradhan. A new framework for designing and analyzing bist techniques: Computation of exact aliasing probability. *International Test Conference*, pages 329–342, 1988.

[9] A. Ivanov and V.K. Agarwal. On a fast method to monitor the behaviour of signature analysis registers. *International Test Conference*, pages 645–655, September 1987.

[10] A. Ivanov, C. W. Starke, V. K. Agarwal, W. Daehn, M. Gruetzner, and T W. Williams. Iterative algorithms for computing aliasing probabilities. *IEEE Transactions on CAD*, 10(2):260–265, February 1991.

[11] Iwasaki and N. Yamaguchi. Design of signature circuits based on weight distributions of error-correcting codes. *International Test Conference*, pages 779–785, 1990.

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

[13] J.G. Kemeny and J.L. Snell. *Finite Markov Chains.* Princeton, N.J., 1960.

[14] S. Lipschutz. *Linear Algebra.* McGraw-Hill, New York, 1968.

[15] D.M. Miller. Review of built-in self test methodologies. *Canadian Conference on Electrical and Computer Engineering*, pages 375–378, November 1988.

[16] J. L. Myers. *Fundamentals of Experimental Design.* Allyn and Bacon, Inc., 1979.

[17] L. B. Page. *Probability for Engineering with Applications to Reliability.* Computer Science Press, Inc., 1989.

[18] W. W. Peterson and E.J. Weldon. *Error Correcting Codes.* MIT Press, Cambridge, MA, 1972.

[19] D.K. Pradhan, editor. *Fault Tolerant Computing*, volume 1. Prentice Hall, 1986.

[20] M. M. Rao. *Probability Theory with Applications.* Academic Press, 1984.

[21] J. P. Robinson. Aliasing probabilities for feedback signature compression of test data. not yet published.

[22] J. E. Scroggs. *Linear Algebra.* Wadsworth Publishing Company, Inc., California, 1970.

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

[24] L. Shu and J. Daniel. *Error Control Coding — Fundamentals and Applications.* Prentice Hall, Englewood Cliffs, New Jersey, 1983.

[25] J.E. Smith. Measures of the effectiveness of fault signature analysis. *IEEE Transactions on Computer*, C-29:510–514, June 1980.

[26] T.W. Williams, W. Daehn, M. Gruetzner, and C.W. Starke. Aliasing errors in signature analysis registers. *IEEE Design and Test*, pages 39–45, April 1987.

[27] T.W. Williams, W. Daehn, M. Gruetzner, and C.W. Starke. Bounds and analysis of aliasing errors in linear feedback shift registers. *IEEE Transactions on Computers*, 7(1):75–83, January 1988.

[28] T.W. Williams, W. Daehn, M. Gruetznmer, and C.W. Starke. Comparison of aliasing errors for primitive and non-primitive polynomials. *International Test Conference*, pages 282–288, 1986.

[29] S. Zhang and D. M. Miller. A comparison of LFSR and cellular automata BIST. *Canadian Conference on VLSI*, pages 8.4.1–8.4.6, October 1990.

Appendix A

Primitive Polynomials and Their Reciprocals

In this appendix a more formal explanation is given of the relationship between a polynomial and its reciprocal. Due to this relationship, only the polynomials themselves, and not their reciprocals, need to be considered in the statistical analysis.

Let $P(\lambda)$ be the characteristic polynomial of a matrix A [1].

$$P(\lambda) = \sum_{i=0}^{n} a_i \lambda^i = a_0 + a_1\lambda + a_2\lambda^2 + \ldots + a_{n-1}\lambda^{n-1} + a_n\lambda^n$$

The reciprocal polynomial of $P(\lambda)$ is defined as:

$$\begin{aligned} P(\tfrac{1}{\lambda}) &= \textstyle\sum_{i=0}^{n} a_i(\tfrac{1}{\lambda})^i \\ &= a_0 + \tfrac{a_1}{\lambda} + \tfrac{a_2}{\lambda^2} + \ldots + \tfrac{a_{n-1}}{\lambda^{n-1}} + \tfrac{a_n}{\lambda^n} \\ &= \tfrac{1}{\lambda^n}[a_0\lambda^n + a_1\lambda^{n-1} + a_2\lambda^{n-2} + \ldots + a_{n-1}\lambda + a_n] \\ &= \tfrac{1}{\lambda^n} \cdot \textstyle\sum_{i=0}^{n} a_{n-i}\lambda^i \\ &= \tfrac{1}{\lambda^n} \cdot Q(\lambda) \end{aligned}$$

Hence:

[1] For all our cases A is nonsingular and $P(\lambda)$ is irreducible and/or primitive.

$$\lambda^n P(\frac{1}{\lambda}) = Q(\lambda)$$

If $\lambda \neq 0$ (A is nonsingular), then

$$Q(\lambda) = 0 \text{ if and only if } P(\frac{1}{\lambda}) = 0$$

This means that the zeroes of $Q(\lambda)$ are the reciprocals of the zeroes of $P(\lambda)$. The zeroes of the polynomials correspond to the eigenvalues of the respective matrices. As the eigenvalues of A^{-1} are the reciprocals of the eigenvalues of A, $P(\frac{1}{\lambda})$ is the characteristic polynomial of A^{-1}. Note that in equation form:

$$Ax = \lambda x \text{ if and only if } A^{-1}x = \frac{1}{\lambda}x$$

If A is the transition matrix representing a linear transformation T in a vector space, then A^{-1} represents the inverse transformation. The cycle structure is the same, but states are visited in reverse order.

Appendix B

Manual Page for 'sig'

NAME

'sig' is a signature aliasing probability analysis program on the basis of the CA, LFSR, HLFSR, CA-LFSR and Binary-Counter test pattern generators, and the MICA, MISR, MICSR, HMISR and CA-MISR data compactors.

SYNOPSIS

sig[-GHIZNCUXRAJKYTBSD] [-Pn] [-l length] [-i seed] [-g genpoly] [-p compoly] [-c genca] [-a compca] [-o sigfile] *circuitfile*

DESCRIPTION

Read in a description of a circuit from *circuitfile*, simulate fault-free and single stuck-at faults or fault collapsing on the set of single suck-at faults or single delay faults by using pseudorandom vector sequence produced by the CA, LFSR, HLFSR, CA-LFSR or Binary-Counter test pattern generator, and evaluate exact fault coverage and signature aliasing probabilities based on the MICA, MISR, MICSR, HMISR and CA-MISR data compactors. The coverage is given by ((N-U)/N)*100 where N

is the number of the faults of the type being considered and U is the number left uncovered. The aliasing probability is computed as (A-U)/(N-U) where N and U are as above and A is the number of the signatures which are the same as a signature produced by fault-free simulation.

CA, MICA, HLFSR, HMISR, CA-LFSR and CA-MISR have maximum length cycle for degree up to 150, and have irreducible property for degree over 150.

CA and MICA: Cellular Automata and Multiple Input Cellular Automata with minimal rule 150 cells (at most two).

LFSR and MISR: Linear Feedback Shift Register and Multiple Input Linear Feedback Shift Register based on primitive polynomials in [2].

HLFSR and HMISR: LFSR and MISR with corresponding polynomials which have half taps evenly distributed.

CA-LFSR and CA-MISR: LFSR and MISR with corresponding polynomials being the same as CA's.

MICSR: MISR with the simple polynomial $x^n + 1$.

OPTIONS

-G Test patterns are generated by the LFSR. By default of -GHIN, -c *genca* and -g *genpoly*, the patterns are generated by the CA.

-H Test patterns are generated by the HLFSR. By default of -GHIN, -c *genca* and -g *genpoly*, the patterns are generated by the CA.

-I Test patterns are generated by the CA-LFSR. By default of -GHIN, -c *genca* and -g *genpoly*, the patterns are generated by the CA.

-Z Test patterns are produced by the LFSRs with their XORs between the cells on using option -G, -H or -I. By default, test patterns are produced with their XORs on the feedback chain.

-N Test patterns are generated by the Binary-Counter. By default of -GHIN, -c *genca* and -g *genpoly*, the patterns are generated by the CA.

-C Evaluate exact fault coverage.

-U Output the number of uncovered faults instead of fault coverage and aliasing probability.

-X Signatures are produced by the MISR.

-R Signatures are produced by the MICSR.

-A Signatures are produced by the MICA.

-J Signatures are produced by the HMISR.

-K Signatures are produced by the CA-MISR.

-Y Signatures are produced by the MISRs with their XORs between the cells on using option(s) -X, -J, and/or -K. By default, the Signatures are produced with their XORs on the feedback chain.

-T Output a result at each output of the circuit. By default, output a result at each 256 outputs of the circuit.

-B Output a result at each output of the circuit within 256 patterns, then at each 256 outputs.

-S Simulate single stuck-at faults. By default of -S and -D, simulate fault collapsing on the set of single stuck-at faults.

-D Simulate single delay faults.

-P *n* *n* is the number of bits in the signature. By default, the number of bits in the signature is equal to the number of outputs in the circuit.

-l *length* *length* is the length of pseudorandom vector sequence. By default, the length of sequence is 2^{m-1} for the CA and LFSR generators, 2^m for the Binary-Counter, where m is the number of inputs in the circuit. If $m > 31$, this option must be given.

-i *seed* *seed* is a positive number for producing initial value of the CA, LFSR, HLFSR, CA-LFSR or Binary-Counter test pattern Generator.

-g *genpoly* *genpoly* is a polynomial described as binary digits, being as an LFSR generator.

-p *compoly* *compoly* is a polynomial described as binary digits, being as a MISR compactor.

-c *genca* *genca* is a CA described as binary digits, being as a CA generator.

-a *compca* *compca* is a CA described as binary digits, being as a MICA compactor.

-o *sigfile* Send the output produced to file *sigfile*. By default, the output is written to *stdout*.

EXAMPLES

sig -A -l 4096 -o C432.sig C432.isc

Read circuit description from *C432.isc* and evaluate signature aliasing probability produced by the MICA date compactor for stuck-at fault collapsing simulation using the CA test pattern generator. The output is sent to the file C432.sig.

sig -CGXAD -l 4096 -o C432.sig C432.isc

Read circuit description from *C432.isc*, evaluate exact fault coverage, and signature aliasing probabilities produced by the MICA and MISR data compactors for single delay fault simulation based on the LFSR test pattern generator. The output is sent to the file C432.sig.

DIAGNOSTICS

All error messages are sent to *stderr* and are intended to be self-explanatory, and assume the circuit file has no error.

NOTES

sig is available in subdirectory "~ szhang/bin" on shannon.

VITA

Surname: Hassan — Given Names: Mahbub

Place of Birth: Chapainawabganj, Bangladesh — Date of Birth: Oct 01, 1965

Educational Institutions Attended:

Middle East Technical University — 1985 to 1989

Degrees Awarded:

B.Sc.(High Honor) Middle East Technical University — 1989

Honours and Awards:

University of Victoria Teaching Award — 1990-1991

PARTIAL COPYRIGHT LICENSE

Title of Thesis:

The Selection of Better Polynomials to Reduce Aliasing in Signature Analysis

Author: 

Mahbub Hassan

June 10, 1991