THE PARTITIONING BEHAVIOR OF LINEAR FINITE STATE MACHINES AND VLSI APPLICATIONS

by

Evaggelia Kontopidi
Honors in Computer Engineering and Informatics
University of Patras, 1989

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

MASTER OF SCIENCE

in the Department of Computer Science

ACCEPTED
FACULTY OF GRADUATE STUDIES
[redacted]
DEAN
DATE 92/04/28

We accept this thesis as conforming
to the required standard

[redacted]
Dr. J.C. Muzio, Supervisor (Dept. of Computer Science)

[redacted]
Dr. M. Serra, Departmental Member (Dept. of Computer Science)

[redacted]
Dr. N. Dimopoulos, Outside Member (Dept. of Elect. & Comp. Eng.)

[redacted]
Dr. F. El-Guibaly, External Examiner (Dept. of Elect. & Comp. Eng.)

Supervisor: Dr. J. C. Muzio

ABSTRACT

Linear Feedback Shift Registers (LFSRs) and Linear Cellular Automata Registers (LCARs) are linear finite state machines which are used in built-in self test as test pattern generators and signature analyzers. We consider the problem of partitioning maximal length machines into a number of smaller maximal length submachines. Our research has shown that only a small percentage of all the primitive machines can be partitioned in such a way. Generally, LCARs have better partitioning behavior than LFSRs. It is suggested that almost all maximal length machines can be partitioned into bit slices which preserve the required property if we allow minimum modifications to the registers. The extra hardware required for partitioning and modifications is estimated using the OASIS Design System and the Magic Layout Editor. The experimental and theoretical results are presented and discussed as well as application areas of our research.

Examiners:

[redacted]

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

[redacted]

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

[redacted]

Dr. N. Dimopoulos, Outside Member (Dept. of Elect. & Comp. Eng.)

[redacted]

Dr. F. El-Guibaly, External Examiner (Dept. of Elect. & Comp. Eng.)

Table of Contents

List of Figures

List of Tables

Acknowledgment

I am grateful to my supervisor, Dr J C Muzio of the Department of Computer Science, University of Victoria, for his invaluable encouragement, guidance and support during my time as a graduate student I would also like to thank the two other professors of the VLSI Design and Test group of Computer Science Department, Dr M Miller and Dr M Serra who taught me courses on VLSI design and testing In addition, they were extremely supportive during our cooperation in the project on design for testability techniques for the Canadian Microelectronics Corporation (CMC)

A sincere thanks goes to all seventeen graduate students of the same group whose hospitality and kindness made my studies and living in Victoria easier I would also like to thank Kevin Cattell and Panagiotis Rondogiannis for numerous discussions and useful suggestions during the preparation of this thesis I am also thankful to all the people who carefully read an earlier version of this thesis and provided comments These are Dr Mike Miller, Kevin Cattell, Grant Marven and Xiaoling Sun

Finally and most importantly, I would like to express my gratitude to my parents, my grandfather and my two sisters for their moral support, love and encouragement throughout my studies

Στους γονείς μου,
Δημήτρη και Ειρήνη

Chapter 1

Introduction

The purpose of this thesis is to convey a better understanding of the behavior of linear machines used in a widely accepted approach to built-in self-test of combinational logic. In this approach, random testing is used for supplying test vectors to the unit being tested and signature analysis is performed to evaluate the responses to these vectors. Typically, the machines which are used in this test structure are Linear Feedback Shift Registers and the recently proposed alternative Linear Cellular Automata Registers [24, 33].

A Linear Feedback Shift Register (LFSR) is a finite state machine, which can be implemented in digital hardware using storage elements (for example, flip-flops), and XOR gates which perform addition modulo 2, chained together and controlled by a synchronous clock. The linear finite state machine, which defines the LFSR, performs polynomial division over the binary field $GF(2)$. The LFSR represents the divisor, a binary polynomial $p(x)$ of degree n where n is the number of memory elements. The serial stream of

inputs of length k represents a binary polynomial $m(x)$ of degree $k-1$ which is the dividend polynomial. The serial output stream gives the quotient, while the last state of the LFSR describes the remainder polynomial.

In testing applications, the input stream k is the output from a circuit under test. The output from the LFSR is discarded, while the remainder (last state) provides the signature. When the LFSR is configured as an autonomous machine, that is without any external input, it can be used as a pseudorandom pattern generator over the binary field.

The LFSR has been used in many other applications. Example of these are shift register counters [18], cyclic code encoders and decoders [50] etc.

On the other hand, a Linear Cellular Automata Register (LCAR) is a finite state machine which has recently been proposed as an alternative to LFSR [24, 33], although their study was initiated by John von Neuman in the 1950's. The properties of LCARs have been studied by Wolfram [55, 56] and later by Pries *et al.* [36]. Three major factors have resulted in the revival of interest in the behavior of cellular systems. First, the development of powerful computers and microprocessors have made possible the rapid simulation of LCARs in a serial, parallel, and/or cellular mode of operation. Second, the use of LCARs to simulate a variety of physical systems (for example, crystal growth) has generated much interest in the scientific community. Third, the advent of VLSI as an implementation medium has focused attention on the communication requirements of successful hardware algorithms.

Ongoing research in the area of VLSI testing is studying the effectiveness of LFSRs and LCARs as stimulus generators and response analyzers both empirically and theoretically. In the former case, the randomness properties of autonomous LFSRs and LCARs are investigated and the anticipated fault

coverage is measured [5, 6, 10, 17, 23, 24, 57, 58] This indicates the degree to which the test pattern generator exercises the potential faults in the circuit In the latter case, the aliasing behavior of LFSRs and LCARs is investigated Aliasing is the condition where the use of data compaction to reduce the test responses to a signature causes a faulty circuit to produce the valid signature thereby causing the built-in self-test to fail to detect the fault The problem of aliasing is of considerable practical importance and it has been studied by a number of authors using either simulation or various error models which are abstractions of the behavior of faulty circuits [4, 13, 14, 26, 53]

This thesis approaches the study of LFSRs and LCARs in a different way We address the problem of partitioning maximal length cycle machines into a number of smaller maximal length cycle submachines To illustrate the practicality of this study we discuss the applications of it in VLSI architectures and several examples are presented with emphasis on boundary scan designs Our approach serves the following broader goals

1 It studies the partitioning properties of two extensively used finite state machines, namely LFSRs and LCARs In this sense, this thesis not only presents the first results in the above topic but also increases our understanding on the behavior of these machines

2 It provides an efficient way for improving the partitioning performance of both LFSRs and LCARs

3 It presents a probabilistic treatment of the partitioning of LCARs which can be used as a basis for developing an analogous theory for LFSRs

4 It suggests VLSI applications where the partitioning of linear finite state machines allows the use of the same machine for different purposes

thereby allowing built-in self-test to be less expensive

The rest of this introduction gives a chapterwise summary of the thesis contents. Chapter 2 is an introduction to digital circuit testing. It explores the objectives of built-in self-test (BIST) techniques, looks at available alternatives and discusses the main problems which may arise in the use of BIST.

Chapter 3 presents some mathematical material from the theory of semigroups and finite fields preparing a background for the rest of the thesis. It examines the structure, the characteristics, properties and the behavior of LFSRs and LCARs, and discusses several recent results in their study.

Chapter 4 is dedicated to VLSI applications where the idea of partitioning linear finite state machines allows more options to the system designer with great economy in hardware. These options include concurrent checking using off line testing resources and testing conventional logic using boundary scan devices.

Chapter 5 examines whether irreducible or primitive LFSRs and LCARs can be partitioned into irreducible or primitive bit slices, respectively. Tables are included containing the results for partitioning machines of length 4 up to 16. Then it describes a bit-sliced implementation of LFSRs and LCARs and compares their partitioning performance.

Chapter 6 suggests that better partitioning performance for LFSRs and LCARs can be accomplished by allowing a single hardware modification. Experimental results are demonstrated to substantiate this claim. It also presents layout studies of various LFSRs and LCARs which estimate the extra hardware required for partitioning and modifications.

Chapter 7 determines the probability that a randomly chosen LCAR will

have the property to be partitioned into primitive or irreducible bit slices It concludes that in particular cases only a good approximation of the calculated probability is possible because of the random distribution of the machines which have the desired property

Chapter 8 completes the thesis It summarizes the main contributions of our research and discusses the most interesting questions that require further investigation

Chapter 2

Built-In Self-Test

In this chapter, the problems of testing digital circuits are introduced. We explore built-in self-test as a solution to these problems and we present different approaches that have been developed for test pattern generation and signature analysis. Finally, the two main problems which can arise in the use of self-test are discussed.

2.1 Design For Test

As the number of transistors integrated into one piece of silicon approaches one million and as these circuits appear in many applications of our daily lives, the demand that they be adequately tested becomes intense. The objectives in testing a design are twofold [37]:

1. to ensure that, before fabrication, the circuit behavior satisfies the intent of the design, that is, it is free from functional or logical design errors, and

2. to detect faulty devices, after fabrication, either on the wafer, as packaged ICs, or as a system component in the field.

Testing of a circuit requires the application of an appropriate set of test stimuli and the comparison of the actual circuit response with the correct response. General purpose testers, though commonly used for this purpose, are very expensive, one million dollars for a tester is not uncommon. There are also problems with the length of time required to generate and apply the test patterns as well as with the large volume of data needed to be handled by the external tester. This results in a test cost contribution to product cost of 35 percent to 55 percent (or more) depending upon product size, technology, and complexity [45].

Several design techniques have been proposed to increase testability and to make testing of the resulting product economical. These techniques are collectively grouped under the name "design for testability (DFT)". DFT techniques include *ad hoc techniques* such as partitioning, initialization, employing test points etc, and *structured design techniques* such as scan techniques, built-in self-test techniques etc. In the rest of this chapter, we discuss various aspects of built-in self-test. Comprehensive introductions to other design for testability methods can be found in [1, 2, 29, 30, 54].

The inclusion of on-chip circuitry to facilitate testing is called *built-in self-test* (BIST). McCluskey [31] presents the argument that "this is the only economical method to reduce the cost of testing". Any built-in self-test method

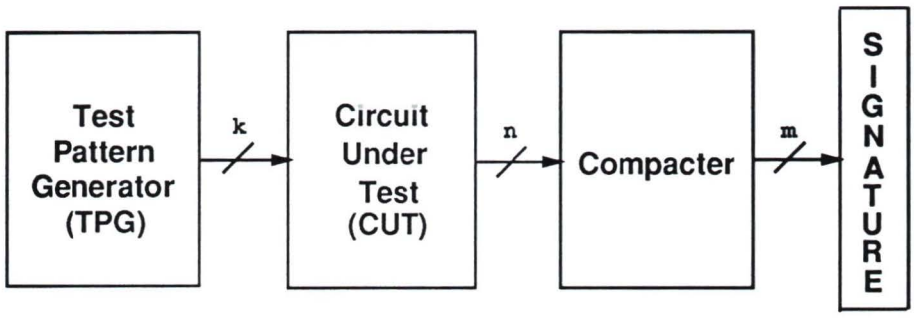


Figure 2.1 Typical BIST architecture

must consists of (1) a strategy for generating the inputs to be applied, (2) a strategy for evaluating the output responses, and (3) the implementation mechanisms.

A typical BIST architecture is shown in figure 2.1. During the self-testing mode a number of test patterns generated by the test pattern generator (TPG) appears on the inputs of the circuit under test (CUT). The circuit output responses are passed through a circuit called a compacter. The aim is to reduce the number of bits that must be examined in order to determine whether the CUT is faulty. At the end of the testing procedure a *signature* of the test response is obtained. Fault detection occurs if the signature realized by the CUT differs from the signature of a fault free version of the circuit.

Many different techniques and approaches have been developed for test pattern generation and data compaction. In the next two sections, we present briefly various such methods, followed by the two main problems which can arise in the use of BIST.

2.2 Test Pattern Generation

In theory, it would be possible to generate test vectors (manually or by a test pattern generation program) and store them in an on-chip ROM. However, this scheme does not reduce the cost of test pattern generation and it requires a very large ROM. One basic approach for test pattern generation is *exhaustive testing* which uses all possible input combinations as test patterns. For example any binary counter can be used to exhaustively generate these patterns. Although exhaustive testing eliminates the need for a fault model and fault simulation, and guarantees excellent fault coverage, for a circuit with a large number of inputs this technique may require too much time. Some form of circuit partitioning may be used to reduce the number of inputs in this case [31].

Random testing is another test pattern generation approach which uses a set of randomly generated patterns as test patterns. Simple circuits, like an *autonomous linear feedback shift register* (ALFSR) can be used in random testing of chips without RAMs[1]. However, the patterns generated by an ALFSR are called pseudorandom patterns[2] to distinguish them from truly random ones. For test pattern generation, pseudorandom sequences are better than random ones since the pseudorandom sequences can be reproduced for simulation. In [7], the sequences produced by an ALFSR are analyzed and their characteristics and properties are discussed in detail. An alternative

[1] Due to pattern sensitivity of RAMs specially generated patterns must be used to test them.

[2] This is because they appear to be random in the local sense, but they are in some way repeatable, hence only pseudorandom.

structure for pseudorandom pattern generation is *linear cellular automata registers* (LCARs). Although ALFSRs and LCARs generated by the same characteristic polynomial have been shown to be similar [40], classes of linear LCARs have been found to have very good randomness properties, and be superior to ALFSRs as pattern generators [10, 23, 57, 58].

In addition to these three best known approaches to the TPG problem, hybrid approaches have also been proposed. These attempt to exploit the best features of each of these individual techniques [3].

2.3 Data Compaction

Various data compaction techniques have been proposed in order to reduce the volume of output data and obtain the signature of the CUT. All these techniques require that the fault free signature for the CUT be known, for example simulation of the design can be used to find the fault free signature.

Compaction techniques include a) **ones counting** (the signature is the sum of the number of 1's appearing on the circuit output) [21, 34], b) **transition count** (the signature is the number of logical transitions, from 0 to 1 and vice versa, in the output data stream) [22], c) **parity compression** (the signature is one bit signifying odd or even parity of the output data stream) [46], d) **syndrome testing** (the signature is the normalized number of 1's in the output response stream under exhaustive application of all possible input patterns) [38] and, e) **compression using Walsh spectra** (the signature is all the Walsh coefficients or a carefully chosen subset of them) [44].

However, the common implementations of BIST using compaction rely on

LFSR techniques These techniques use an LFSR as a compaction circuit an which is fed by the output data stream An LFSR performs a polynomial division over GF(2) where the input stream is the dividend polynomial and the LFSR itself implements the divisor polynomial At the end of the division, the last stage of the LFSR describes the remainder polynomial or the signature of the CUT The output of the LFSR which represents the coefficients of the quotient polynomial is normally discarded

For a multiple-output circuit under test the overhead of a single-input signature analyzer on every output would be high Of course the single-input analyzer could be multiplexed to the various circuit outputs, one at a time, and the test sequence repeated for each output However, a preferred structure for signature analysis of multiple-output circuits is a *multiple-input signature register* (MISR) Network outputs are connected to the LFSR through XOR gates added to the shift lines between stages This is faster, but requires more added circuitry than the serial signature analyzer

LCARs can also be used to obtain the signature of the CUT in a similar manner and it has been found that classes of the LCARs are superior to LFSRs as signature analyzers [14]

2.4 Two Main Issues in BIST

Although BIST appears to be the only method to reduce the cost of testing there exist two main problems concerning the test pattern generation and the signature analysis Both of them are presented in the following two paragraphs In our discussion, we assume that both the test pattern generator and the signature analyzer are implemented using an LFSR However, similar

results follow if LCARs are used

Fault coverage Perhaps the most important feature of a testing technique is its coverage the fraction of the possible failures that the test technique can detect [31] Usually, less than 100% fault coverage is obtained and it becomes increasingly expensive to generate tests for the last untested faults However, in the literature there is no proposed test pattern generation technique for BIST which always guarantees 100% fault coverage Furthermore, when pseudorandom patterns generated by an LFSR are used as test stimuli, there is always a concern about the linear dependencies within the sequence of patterns It is possible for these linear dependencies to preclude *a priori* a specific test pattern from being present in the sequence of applied patterns If this is the case and the particular test pattern is in the test set[3] for the CUT, then a 100% fault coverage is no longer possible

Although the linear dependencies within a sequence of patterns and ways to calculate their effects have been studied in [5, 7], the problem of finding a method for test pattern generation which could always guarantee that the test set is included in the generated pattern set and hence 100% fault coverage is still open In the next paragraph, the main problem with signature analysis is introduced

Aliasing The main problem with signature analysis is that due to the loss of information during compaction, the output response from a faulty circuit may produce a signature that is identical to the signature of a fault free

[3]By the term *test set* we mean the set of test patterns by which we can determine whether the CUT is faulty or fault free

circuit. This phenomenon is called *aliasing*. As a result it is possible for a fault to not be detected even though the output response differs from the fault free response and the CUT to be declared fault free even though it is actually faulty. The probability of aliasing is strictly a function of the machine length, i.e., it approaches 2^{-m} where m is the length of the machine used in signature analysis.

The aliasing characteristics of a MISR are quite similar to these of a single-input signature analyzer [7]. However, an MISR has an additional type of aliasing called *error cancellation* which is independent of the feedback polynomial. For example suppose an error occurs on an output at test cycle i. This error is canceled if an error occurs on the next output at the next test cycle $i+1$. It is shown in [7] that error cancellation is highly unlikely, and hence, the aliasing characteristics of multiple-input signature analyzers are very nearly as good as these of single-input analyzers.

A number of techniques have been proposed in the literature to reduce aliasing in signature analysis [4]. Here, it is worth briefly describing a compaction technique presented in [19], called *periodic quotient compression*, which guarantees zero aliasing. Since the problem of aliasing occurs because of loss of information contained in the discarded quotient bits, this technique looks at both the remainder and the quotient, assuming that the good circuit response is known a priori during the design of the LFSR. An algorithm is given in order to construct an LFSR in such a way that the quotient is periodic with a period T and hence can be checked using $O(T)$ hardware. Although the technique is applicable to all CUTs, the hardware overhead can be large since for an N bit output response, zero aliasing can be achieved using an LFSR of length $\lceil N/2 \rceil$ in the worst case, which is too costly to be

used in practice, especially in BIST environments

Chapter 3

Linear Finite State Machines

A widely accepted approach to the built-in self-test of combinational logic requires a pseudo random test vector generator and a data compactor. Linear feedback shift registers (LFSRs) form the basis for these two units, as their random sequencing and inexpensive implementation makes them suitable for hardware testing. Recently, [24, 33], one-dimensional linear cellular automata registers (LCARs) have been proposed as an alternative to LFSRs for test vector generation and data compaction.

In this chapter, we discuss the structure, the characteristics, properties and the operation of an LFSR. Then, a similar discussion is provided for an LCAR. Finally, we present some recent results in the study of LFSRs and LCARs.

3.1 Algebraic Concepts

In this section, we discuss some appropriate mathematical material from the theory of semigroups relative to linear finite state machines. We also examine aspects from the theory of finite fields, preparing a background for the following sections. The main sources for this section are [7, 35, 41].

Definition 3.1 [41, page 208] A *finite state machine* is an algebraic structure $< S, I, Y, M, \delta >$, where S, I, and Y are finite sets of states, inputs, and outputs, respectively, M (the next state function) is a mapping from $S \times I$ into S, and δ (the output function) is a mapping from S into Y.

We specify the behavior of a finite state machine by specifying its next state and output functions. The next state function of a machine can be described graphically using a *state graph*.

One important aspect in the design of finite-state machines is that of designing the least expensive machine that will realize a specific behavior. This can be obtained by reducing the number of states in a sequential machine since the cost of constructing a real machine increases with the number of states in the machine. In [41, page 220], an algorithm for state reduction is discussed.

Another concept that arises from the study of finite-state machines is the concept of *machine homomorphisms*. Here, we avoid a formal definition of a machine homomorphism (the reader can find one in [41, page 226]), by saying that if a machine M_1 is a homomorphic image of a machine M_2, then the next-state function of M_1 has some (and possibly all) of the structure of the next-state function of M_2. If this is the case the machine M_2 can simulate its homomorphic image M_1.

In preparation for our study of linear finite state machines, we introduce several related algebraic structures.

Definition 3.2 A *field* F is a set of elements with two operations + (called addition) and · (called multiplication) such that

(a) it is an Abelian group [35] under + with identity 0,

(b) the non-zero elements form an Abelian group under · with identity 1 $(1 \neq 0)$,

(c) the distributive law holds – that is,

$x \cdot (y+z) = (x \cdot y) + (x \cdot z)$ and $(y+z) \cdot x = (y \cdot x) + (z \cdot x)$

for all x, y, z in F.

For example $< Q, +, \cdot >$ and $< R, +, \cdot >$ are fields. Also $< Z_p, +_p, \cdot_p >$ is a field if and only if p is prime[1]. Here, Z_p, is the set of integers modulo p, and $+_p$ and $\cdot_p$ denote addition and multiplication modulo p. This field is denoted as $GF(p)$, the *Galois field of order p*.

Definition 3.3 A *finite field* is a field which has a finite number of elements. The number of elements in the field is called the order of the field.

Definition 3.4 Let $f(x)$ and $g(x)$ be two polynomials in the vector space of polynomials of degree less than n over a field K. Then $f(x) \cdot g(x)$ is defined to be $[f(x)g(x)]$ mod $(x^n - 1)$, where $f(x)g(x)$ denotes the product obtained by using ordinary polynomial multiplication with coefficients computed in the field K.

[1]Recall, Q is the set of rational numbers, R is the set of real numbers, and Z is the set of integer numbers.

Definition 3.5 A polynomial $p(x)$ of degree $n > 0$ over a field K is *irreducible* over K if and only if there do not exist polynomials $f(x)$ and $g(x)$ of degree greater than 0 over K such that $f(x)g(x) = p(x)$, where multiplication is ordinary polynomial multiplication with coefficient operation in K.

The reader can easily verify that the polynomials x, $x+1$, and x^2+x+1 are all irreducible polynomials over $GF(2)$.

Definition 3.6 A polynomial $p(x)$ of degree n is a *monic* polynomial if the coefficient of x^n is unity.

Definition 3.7 [41, page 274] The *characteristic* of any field (finite or infinite) is the order of 1 in the additive group of the field.

Theorem 3.1 [41, page 277] Let $f(x)$ be a polynomial of degree n over any field K. Then $f(x)$ has at most n distinct roots in K.

It is proven in [41, page 274] that the characteristic of any field is either prime or infinite. Our discussion is concentrated now on mathematical aspects for finite fields.

Definition 3.8 Let α be an element of a field K of characteristic p. The *minimum polynomial* of α, denoted by $M_\alpha(x)$, is the monic polynomial of least degree over $GF(p)$ that has α as a root.

Theorem 3.2 [41, page 283] Let $f(x)$ be an irreducible polynomial of degree d over $GF(p)$, and let a be a root of $f(x)$. Then the d distinct roots of $f(x)$ are a^{p^i}, where $0 \leq i \leq d-1$.

Definition 3.9 An element α of the field $GF(p)$ is called a *primitive element* if it generates the multiplicative group of the field.

Definition 3.10 The minimum polynomial of a primitive element is called a *primitive polynomial* and all its roots are primitive elements.

It is proved in [35, page 161] that an irreducible polynomial of degree m over $GF(q)$ is primitive if, and only if, it divides $x^n - 1$ for no n less than $q^m - 1$.

Several listings of such polynomials exist in the literature. A complete listing of irreducible polynomials over $GF(2)$ up to degree 16 (with primitive polynomials identified) is given in Peterson and Weblon (1972). In the same source, many polynomials of higher degree up to 34 are also listed. Also in [7, pages 336–338] a primitive polynomial for each degree up to 300 is given. The polynomials listed are these having the smallest number of terms for each degree.

Before the introduction to linear feedback shift registers and linear cellular automata registers, two basic definitions from linear algebra are presented.

Definition 3.11 A function $f: V \rightarrow V'$ is a *linear function* from a vector space V into a vector space V' over the same scalar field K as V if, for all c_1 and c_2 in K and u_1 and u_2 in V,

$$f(c_1u_1 + c_2u_2) = c_1f(u_1) + c_2f(u_2)$$

Definition 3.12 A machine M is a *linear finite state machine* if

1. The state space S_M of M, the input space I_M, and the output space Y_M are each vector spaces over a finite field K (we let the dimensions of spaces be n, p, and r, respectively), and

2. Let the vector s_i denote the state of the machine, the vector u_i denote the input to the machine, and the vector y_i denote the output of the machine at time i. The next-state s_i^+ and the output of M are defined, respectively by

$$M(s_i, u_i) = s_i^+ = A \cdot s_i + B \cdot u_i$$
$$\delta(s_i) = y_i = C \cdot s_i + Q \cdot u_i$$

where A, B, C, and Q are matrices over K, s_i, u_i, and y_i are column vectors, and the matrix operations are the usual matrix operations with arithmetic performed in the field K.

Here K is $GF(2)$ and the $+$ operation is addition modulo 2 (XOR). As one can see from the definition, the relationship between the state of the machine at time i and the state at time $i+1$ is linear.

If the machine has no input, it is called *autonomous*. The next state function of an autonomous linear machine is defined by $s_i^+ = M(s_i) = A \cdot s_i$ and the output is defined by $y_i = C \cdot s_i$.

In preparation for the analysis of the cycle structure associated with an autonomous linear machine, a few definitions from linear algebra are given below.

Definition 3.13 An $n \times n$ matrix A over a field K is said to be *nonsingular* if $det(A) \neq 0$, where the determinant is calculated in the arithmetic of the

field K

The state graph of an autonomous linear machine that has a nonsingular next-state matrix is a collection of disjoint cycles The difference between a singular and a nonsingular machine is the presence of transient states in the state graph of the singular machine [41, page 304]

Definition 3.14 If A is any $n \times n$ matrix over a field K, and if Q is a nonsingular $n \times n$ matrix over the same field, then the matrix $A' = Q\ A\ Q^{-1}$ is said to be *similar* to A

Definition 3.15 The *characteristic polynomial* of a square matrix A is defined as $\Delta_A(\lambda) = det(\lambda I - A)$ Δ_A is a degree n polynomial in the indeterminate λ

The next theorem shows that similarity transformations preserve cycle structure

Theorem 3.3 [41, page 307] If matrices A and A' are similar nonsingular state-transition matrices, then their state graphs have identical cycle structures and differ only in the labeling of the states

From the preceding theorem, we see that the problem of finding the cycle structure induced by a matrix A reduces to the problem of finding the cycle structure for some matrix similar to A The theorem give us the freedom to select the machine of least cost in an equivalence class This property suggests that we should look for some standard machine in each equivalence class that is likely to be the least costly machine to synthesize For this purpose we choose feedback shift registers

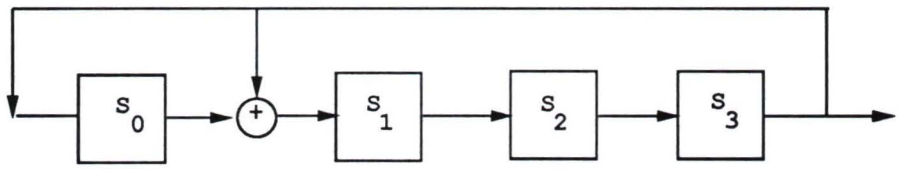


Figure 3.1 Type 1 LFSR

3.2 Linear Feedback Shift Registers

A *Linear Feedback Shift Register* (LFSR) is a finite state machine defined as a collection of storage elements (for example, flip-flops) and XOR gates which perform addition modulo 2, chained together and controlled by a synchronous clock. In the analysis of LFSRs, all operations are done modulo 2[2]. Such a circuit is considered to be linear since it preserves the principle of superposition, i.e., its response to a linear combination of stimuli is the linear combination of the responses of the circuit to the individual stimuli. Two possible realizations of a four-stage shift register are shown in figure 3.1 and figure 3.2, where the symbol $\oplus$ represents XOR gate, and each box represents a memory cell which holds one $GF(2)$ field element. In this network, the state of the network is the four-tuple of field elements stored in the register.

For any LFSR one can write the set of finite state machine equations which describe the transition to the next state and also the corresponding state transition matrix. For example, the equations for the LFSR of figure

[2]This is the same as ordinary arithmetic with one exception $1 + 1 = 0$

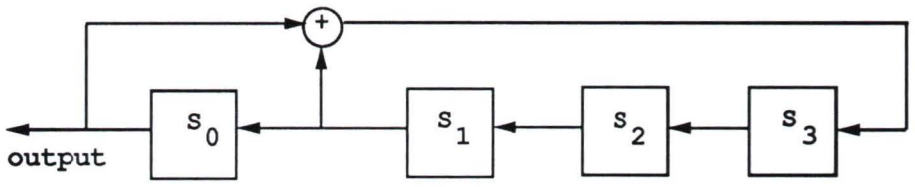


Figure 3.2 Type 2 LFSR

3.1 are

$$\begin{aligned} s_0^+ &= s_3 \\ s_1^+ &= s_0 \oplus s_3 \\ s_2^+ &= s_1 \\ s_3^+ &= s_2 \end{aligned}$$

and the corresponding state transition matrix is

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

The generating function $f(x)$, corresponding to an LFSR is called the *characteristic polynomial*. This is the characteristic polynomial of the tran-

sition matrix A. The LFSRs in figure 3.1 and 3.2 have $f(x) = x^4 + x + 1$ as their characteristic polynomial. Note that the output of stage 1 represents x or x^1, stage-2's output is x^2, and so forth. Here a nonzero polynomial coefficient implies that a connection exists, while a zero polynomial coefficient implies that no connection exists.

Given a characteristic polynomial, it is easy to implement an LFSR to realize it. The last column of the transition matrix is the set of coefficients of the characteristic polynomial. Conversely, given an LFSR, it is a simple process to determine its corresponding characteristic polynomial.

Consider an autonomous linear feedback shift register, i.e., an LFSR which has no inputs except for clocks. Such a circuit is cyclic in the sense that when clocked repeatedly, it goes through a fixed sequence of states. Given an n-stage shift register there are at most 2^n possible states. If the feedback is linear, the successor of the all zero states is itself, since the output of the feedback circuitry is 0 when all of its inputs are 0. Hence, a linear feedback shift register, starting from a nonzero state can reach at most $2^n - 1$ different states. Then the sequence generated by one of the stages of the register is periodic with period no greater than $2^n - 1$.

The relation between the characteristic polynomial of an LFSR and the period of the resulting sequence is determined from the following theorem.

Theorem 3.4 [7, page 69] Given an LFSR initialized such that the first $n-1$ stages contain 0 and the nth stage contains a 1, then the LFSR sequence $\{a_n\}$ is periodic with a period which is the smallest integer k for which $f(x)$ divides $1 - x^k$.

Another interesting property concerning the periodicity of shift register sequences is that, when $f(x)$ is irreducible, the period is independent of the

initial conditions (except for the all 0's case)

Definition 3.16 If the sequence generated by an n-stage LFSR has period $2^n - 1$, it is called a *maximum-length sequence* or *m-sequence*.

The following definition is equivalent to definition 3.10.

Definition 3.17 [7, page 77] The characteristic polynomial associated with a maximum length LFSR is called a *primitive polynomial*.

On some occasions it is desirable to extend the period of a sequence from $2^n - 1$ to 2^n. In this case, nonlinear feedback functions are required to implement these sequences of period 2^n, called de Bruijn sequences [7, page 74].

Sequences generated by LFSRs are called pseudorandom sequences, since they are periodic and deterministic but they have many properties like these of random sequences[3]. Some of these properties [1, page 441] are listed next.

In the following, any sequence of $2^n - 1$ consecutive outputs is referred to as an $m-$sequence. Furthermore, a maximal contiguous grouping of symbols is called a "run".

Property 3.1 The number of 1s in an $m-$sequence differs from the number of 0s by one.

Property 3.2 An $m-$sequence produces an equal number of runs of 1s and 0s.

[3]For random test pattern generation it is not so important that the numbers used be strictly random, since they are not being used for statistical purposes.

Property 3.3 In every m-sequence, one half the runs have length 1, one fourth have length 2, one eighth have length 3, and so forth, as long as the fractions result in integral numbers of runs.

These randomness properties make it feasible to use LFSRs as test sequence generators in BIST circuitry. When an LFSR is used as a signature analyzer the following holds [1, page 444].

Theorem 3.5 An LFSR signature analyzer based on any polynomial with two or more nonzero coefficients detects all single bit errors.

Definition 3.18 A (k, k) *burst error* is one where all erroneous bits are within k consecutive bit positions, and at most k bits are in error.

Theorem 3.6 If the characteristic polynomial $f(x)$ is of degree n and the coefficient of x^0 is 1, then all (k, k) burst errors are detected as long as $n \geq k$.

In chapter 2, we saw that the probability of aliasing, i.e., the probability that a length n LFSR does not detect an error, is approximately 2^{-n}. However, this bound is not too useful since it is based on unrealistic assumptions. In general, signature analysis gives excellent results. Results are sensitive to $f(x)$ and improve as n increases.

Implementation Issues The structure of an LFSR like the one shown in figure 3.1 is typically inconvenient to implement because XOR gates have to be included between certain stages. For this reason, the LFSR shown in figure 3.2 is easier to layout. This alternative structure does calculate the actual remainder, however its final state has certain properties and is still usable in the signature analysis technique. In fact, this alternative form of LFSR is the one that is typically implemented.

3.3 One-Dimensional Linear Cellular Automata Registers

In this section, we present the basic theory and the operation of *linear cellular automata registers* (LCARs). Recently LCARs have been proposed as an alternative to LFSRs for test vector generation and data compaction [24, 33].

LCARs are finite state machines, defined as uniform arrays of identical cells in an n-dimensional space. Cellular automata registers can be characterized by the following four properties.

1. The cellular geometry.
2. The neighborhood specification. Cells are restricted to local neighborhood interaction and have no global communication.
3. The number of states per cell.
4. The algorithm to compute the successor state. Cells use an algorithm to compute their successor state, called its *computation rule*, based on the information received from its nearest neighbors.

There are 256 possible distinct cellular automata rules in one dimension with a three-site neighborhood. However, only the combination of linear rules[4] 90 and 150 (these are defined later in this section), yields maximal length cycles LCARs. This is a *hybrid* LCAR since all sites do not use the same rule.

[4]This means that the next state of the machine can be computed from the previous state (and perhaps inputs) using only linear operations (XOR operators).

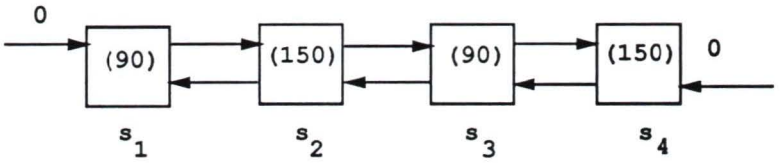


Figure 3.3: A linear cellular automaton register.

For the rest of this thesis, whenever LCARs are mentioned, the reference is to one-dimensional LCARs with computation rules 90 and 150. Figure 3.3 is an example of the type of machine being described. Each site, labeled s_i, $1 \leq i \leq k$, can hold either 0 or 1, and at every clock cycle, it receives an input from its nearest neighbors, s_{i-1} and s_{i+1}. The sites at the boundary of the array always receive a 0[5].

The computation rules 90 and 150 are defined as follows:

$$\textit{Rule } 90 \quad : \quad s_i^+ = s_{i-1} \oplus s_{i+1}$$

$$\textit{Rule } 150 \quad : \quad s_i^+ = s_{i-1} \oplus s_i \oplus s_{i+1}$$

According to rule 90, the value of a particular site i is simply the sum modulo 2 of the values of its two neighboring sites on the previous time step t. Rule 150 also includes the value of site i at time step t. If we use 1 to denote a rule 150 cell and 0 to denote a rule 90 cell, then an LCAR can be represented by a binary vector. For the example of figure 3.3 the binary vector has the form (0101).

[5]There are alternative possible boundary conditions that can be also considered (see [36])

For any LCAR, one can write the set of finite next state equations and also the corresponding state transition matrix. The characteristic polynomial of the state transition matrix is the characteristic polynomial of the LCAR. For the example of figure 3.3 the next state equations are

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

and the corresponding state transition matrix is A'

$$A' = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix}$$

with characteristic polynomial $f(x) = x^4 + x + 1$. This is primitive and hence the LCAR of figure 3.3 produces a maximum length cycle.

3.4 Discussion

In the previous two sections, the characteristics and properties of two linear finite state machines, LFSRs and LCARs, were discussed. Both of these machines can be used as pseudo-random pattern generators and signature analyzers. The next paragraph summarizes some recent results in the study of LFSRs and LCARs.

- Empirical results showed that an LCAR has better randomness distribution and hence better fault coverage than an LFSR [23, 57, 58].
- An LCAR and an LFSR with the same irreducible (or primitive) characteristic polynomial are isomorphic. This implies that they have the same cycle structure and differ only in the labeling of the states [40].
- An LCAR and an LFSR which are based on the same irreducible (primitive) polynomial, and thus are similar to each other, have the same aliasing for a single input stream [40].

P. D. Hortensius *et al.* [23] suggest that if a designer requires a pseudorandom number generator for built-in self-test with primary concerns of area, speed, and test coverage she/he should use the hybrid LCAR rather than the LFSR with its inherent correlation.

This concludes our introduction to the linear finite state machines which are used in BIST for test pattern generation and signature analysis. The use of LCAR as compaction testers must be evaluated on performance and implementation criteria. In particular, the extensibility to longer lengths, without the introduction of very long feedback loops (causing delays), is a major consideration.

Chapter 4

VLSI Applications

In chapter 3, we present some background on the theory and operation of LFSRs and LCARs. These machines can be used in BIST designs as sources of pseudorandom binary test sequences and as means to carry out response compaction. This approach has some associated cost although it allows testing to be accomplished without the aid of expensive external hardware. Since the self-test circuitry (either an LFSR or an LCAR) uses chip area, there is an area and pin overhead which in turn implies a decrease both in the yield and in the reliability. However, the cost in silicon can be reduced significantly if the same linear finite state machine can be used for other purposes. For example, this can be carried out if a machine can be partitioned into smaller submachines which preserve certain properties.

In this chapter, we present VLSI designs where the idea of partitioning long machines allows efficient use of the built-in self-test hardware. The designs which are considered here comply with the new ANSI/IEEE Std

1149 1, IEEE Standard Test Access Port and Boundary Scan Architecture Initially, an introduction to this standard is presented Then, it is demonstrated how concurrent checking using off line testing resources, and testing conventional logic using boundary scan devices, is accomplished by partitioning machines which already exist in the design to serve other functions

4.1 Introduction

Efficient techniques for the testing and diagnosis of digital systems and modules are becoming of critical importance as the density of integrated circuits is increasing However, as designs grow more complex, established testing techniques become less cost-effective while at the same time, the physical access for the bed-of-nails, on which many techniques depend, has become very limited [28] One solution to this complexity is to turn to more advanced methods, such as ANSI/IEEE std 1149 1 [25] - the *Standard Test Access Port and Boundary Scan Architecture* Boundary scan equips a device with shift register elements that are logically, and often physically, adjacent to the I/O pins of every chip on the board These elements are connected to form a shift register path around the periphery of the IC The shift register is used to shift, apply, or capture test data and can thereby test, not only individual chips, but also the board interconnect

Although the standard supports a number of different test modes (either mandatory or optional), it does not explicitly address concurrent checking, but it provides for establishing a framework that would merge boundary scan and concurrent checking

4.2 Concurrent Checking

Built-in self-test techniques are only capable of detecting design faults and fabrication defects, they cannot detect temporary errors. Typical temporary faults include a) *intermittent faults* due to environmental conditions, b) *transient faults* due to electromagnetic interference and, c) *soft errors* caused by the interaction of charged particles with the silicon substrate. The transient faults tend to be random faults which cause no damage to the circuit in which they occur. Intermittent faults are usually recurring faults which occur randomly in time and always appear in the same part of a circuit. It is predicted in [37] that temporary faults are likely to increase and become more dominant in integrated circuits as device geometries decrease in size.

The inability of built-in test techniques to detect temporary errors arises from the fact that these are used only once during initial testing or possibly to diagnose a system after it has failed. If the fault causing the error is permanent then it can be detected, however, if it is temporary then it may go undetected. Another problem which arises with these techniques is that the testing is carried out while the tested circuit is not in use (*off-line testing*). However, reliability and continuous operation are very important issues, especially in real-time applications.

Techniques which allow digital circuit testing without suspending the normal operation (*on-line testing*), include hardware and software redundancy. *Hardware redundant techniques* range from self-checking circuits at the gate level [49] through duplication at the module level to replicated computers at the system level. *Information redundant approaches* include coding schemes

which promise adequate fault detection without the large area overheads, in this thesis, the term *concurrent checking* is used to refer to information redundant methods.

In concurrent checking, extra bits are added to the data used in the system so that the integrity of the system can be checked. This is achieved through the use of error detecting codes. Possible coding schemes which can be used for that purpose are: **a)** *parity check codes* (basically, one more bit is added to the original data such that the resulting code word has even or odd parity), **b)** *m-out-of-n codes* (they require m out of the n information bits be ones), **c)** *residue codes* (code words are created by appending the residue), **d)** *AN codes* (code words are created by multiplying the original data by a constant A), **e)** *berger code* (code words are created by appending the binary representation of the number of 1s in the original data). All these coding schemes require specially designed checkers for error detection.

However, in order to take advantage of boundary scan design and use the shift register elements that surround the periphery of a CUT, another coding scheme should be employed. In [43], a separable cyclic coding scheme based on LFSR/LCAR implementation is proposed for this purpose. (This scheme is defined later in this chapter.)

Figure 4.1 shows the general structure of concurrent checking. During the concurrent checking mode a number of test patterns used in normal operation appears in the inputs of the circuit under test (CUT). The circuit coded output responses are monitored by the checker which signals the detection of a fault by decoding. Although concurrent checking allows detection of temporary faults during system normal operation, its use is limited in VLSI environments. This is because the fault coverage is less than that of explicit

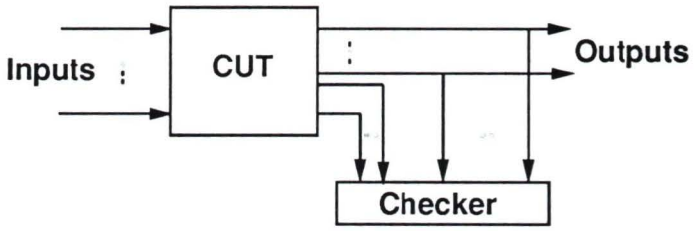


Figure 4.1 The general structure of concurrent testing

testing as well as because extra hardware is required. In the literature, no firm results have been established for the overhead expected for concurrent checking. However, experiments with PLAs have shown that a 30% hardware increase can be expected on average, without counting the expense of the checker [39].

However by merging boundary scan and concurrent checking, hardware resources that already exist in the design can be used to perform on-line testing. In section 4.4, we discuss a framework for that purpose. First, an overview of the principal features defined by the ANSI/IEEE Std 1149.1 is provided. Emphasis is given in the different test modes proposed by the standard.

4.3 Boundary Scan

The development of boundary scan architecture originated from design engineers representing major European electronics companies. Their concern was the rising cost of testing complex PCBs by probing them physically. As a result, early in 1986, the Joint European Test Action Group (JETAG, later

to be known as JTAG) was established, the efforts of which have resulted in the new standard with the formal name ANSI/IEEE Std 1149.1, *IEEE Standard Test Access Port and Boundary Scan Architecture*. The standard as defined in [25], provides a means for convenient access to a unit under test. Boundary scan can be used at different hierarchical levels of testing like component testing, system testing etc.

Figure 4.2 shows a high level view of a board with boundary scan parts, while figure 4.3 shows a chip level view of boundary scan. Each device on the board has a 4 or 5 wire interface called the *Test Access Port* (TAP) that includes two control pins, TMS (test mode select) and TCK (test clock), two scan pins, TDI (test data in) and TDO (test data out), and optionally, the TRST (test reset input) pin which provides a supplementary reset mechanism. The test logic in the boundary scan architecture also includes many other features such as a TAP controller and various registers.

The *TAP controller* allows one to keep the number of pins in the TAP to a minimum. This is very important since many ICs are pin - (rather than silicon) - limited. The TAP controller is a 16-state finite state machine (however, in eight states, no operation of the test logic occurs) which generates the clocks and control signals required for the correct operation of the other parts of the test logic.

ANSI/IEEE Std 1149.1 specifies the design of various registers: Instruction, Bypass, Identification, Boundary Scan, and User-Specific. The mandatory registers are the Instruction, Bypass and Boundary Scan registers. In general, a register element is capable of three fundamental actions: capture, shift and update. *"Capture"* and *"update"* mean to transfer data to or from the parallel input or output ports of the register. *"Shift"* means to transfer

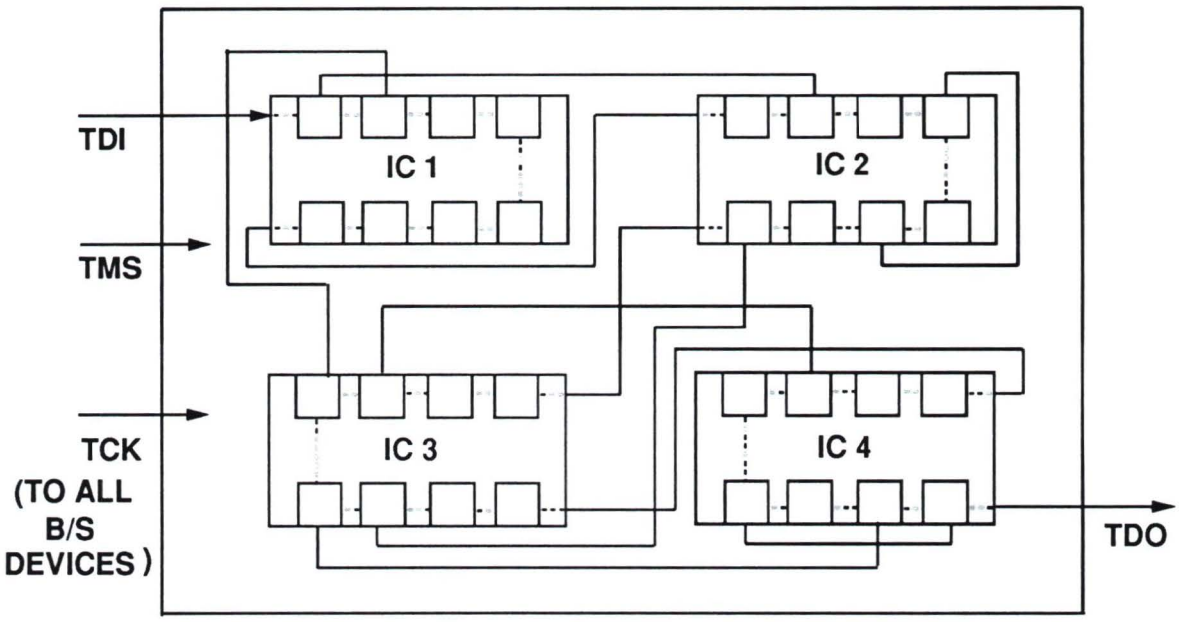


Figure 4.2 High level view of boundary scan

register data serially through the elements of the register. Not all register elements contain all these features, for example, the identification register does not contain an update function. But, all registers must contain a core shift function. Furthermore, only one register is connected from TDI to TDO at any one time. As a result, the shift elements can be considered to be a shared resource, thereby reducing the final number of memory elements needed to implement the various registers [8]. Which register is connected from TDI to TDO depends on the selected test mode.

The standard supports the following test modes:

1. *Bypass*. In this mode, a minimum length serial path is setup to bypass an IC. This allows access to the chip of interest in the minimum possible time, increasing test throughput.

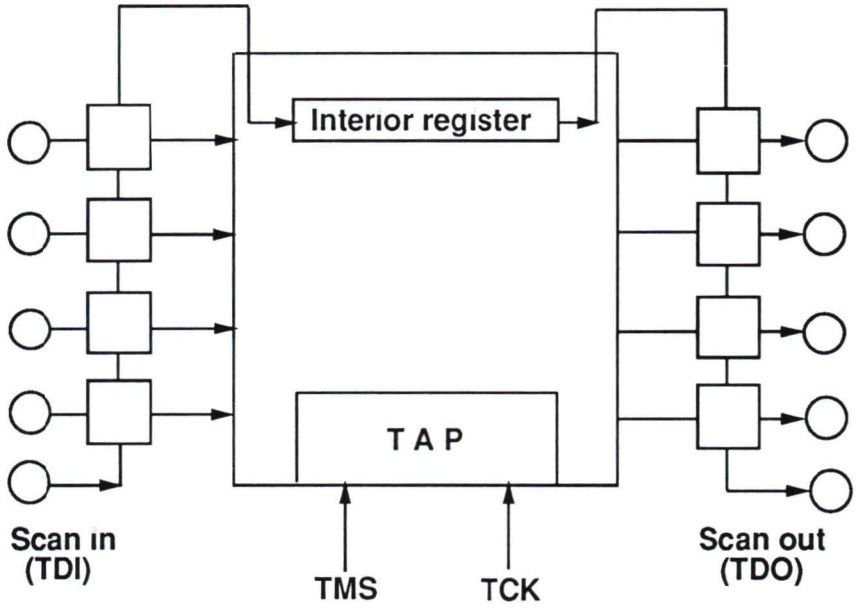


Figure 4.3 Chip level view of boundary scan

2 *Idcode/Usercode test* This allows a binary data pattern to be read from the chip that identifies the manufacturer, the part number, the variant, and (where appropriate) the programmed state, the identification register is used for this purpose. This information might be used for example to verify that the correct IC has been mounted in each board location.

3 *Sample test* This allows a snapshot of the normal operation of the component to be taken and examined. Data is latched in both the input and output registers. During this test only the boundary scan register is connected between TDI and TDO.

4 *External test* This mode is used to test the interconnect of the printed

circuit board. The boundary scan register is the one and only register that is to be connected between TDI and TDO for data scanning purposes. Boundary scan register cells at output pins are used to apply test stimuli to the board, while these at input pins capture the test results flowing from another chip via the board. Captured results are scanned out of the serially linked boundary scan registers of a board while the next set of test input values is scanned in.

5. *Internal test*. This is one of the two optional test modes defined by the standard which provides the means to test the internal logic of the design. During internal test, test stimuli are shifted in through the boundary scan input register, one at a time and applied to the on-chip system logic. The corresponding responses are captured in the IC's boundary scan output register and are examined by subsequent shifting. Internal test mode must select the boundary scan register to be the one and only one register connected between TDI and TDO for shifting access, i.e., each pattern and response must be shifted through the boundary scan register.

6. *Built-in self-test*. This is the second optional test mode, however the standard recommends its implementation wherever possible. The use of BIST mode allows a component user to determine the health of a component without the need to load complex data patterns. While the BIST mode is selected, the boundary scan register may act as a pattern generator and/or signature compactor. The resulting signature is shifted out and is checked to ensure proper circuit operation. When BIST is the current test mode, the test data register into which the

results of the self-test(s) will be loaded, must be connected for the serial access between TDI and TDO. (This register can be the boundary scan register.)

When the circuit in not in one of the test modes, it performs it's normal function.

All the different test modes described here except the sample mode allow off-line testing of a circuit. However, with the sample test mode the test engineer can only take a snapshot of the circuit in time. The standard does not support a test mode by which a continuous monitoring (i.e., concurrent checking) of the CUT is possible. In [43], X. Sun and M. Serra present a scheme which allows merging of concurrent checking and off-line testing in PLA's. This scheme can be easily extended to boundary scan architectures where more resources are available. The idea of incorporating concurrent checking with boundary scan is discussed in the next section. The description which follows makes clear that this is possible only by partitioning maximal length cycle machines into smaller maximal length submachines.

4.4 Boundary Scan and Concurrent Checking

In this section, we demonstrate how the concepts of boundary scan and concurrent checking are merged by partitioning hardware resources. These resources are used by test modes proposed by the standard. The purpose is to increase system reliability while introducing minimum overhead. This is possible only when the appropriate coding scheme is used. Such a scheme is proposed in [43, page 10] and is defined next.

LFSR/LCAR Based Coding Scheme Let $v(x) = v_{n-1}x^{n-1} + v_{n-2}x^{n-2} + \cdots + v_1x + v_0$ be the polynomial over $GF(2)$ represented by a binary vector $v = (v_{n-1}, v_{n-2}, \cdots, v_1, v_0)$, and let $G(x)$ be a *generator polynomial* then,

Definition 4.1 A *separable cyclic encoding* of a vector v is the vector $v \circ r$ where, $\circ$ denotes the concatenation of two vectors and, r is the binary vector represented by the polynomial $r(x) = v(x) \ mod \ G(x)$

In other words, the output vector of a multiple output CUT is taken as a dividend polynomial which is divided by a generator polynomial. The remainder of this division is appended to the output vector and the code word is created.

For example, consider the output vector $v = (1001011)$ which corresponds to the input stimuli 111 of a given CUT and, the generator polynomial $G(x) = x^3 + x + 1$. The binary vector $v = (1001011)$ represents the polynomial $v(x) = x^6 + x^3 + x + 1$. Then, $r(x) = v(x) \ mod \ G(x) = (x^6 + x^3 + x + 1) \ mod \ (x^3 + x + 1) = x^2 + 1$. The polynomial $r(x)$ is represented by the binary vector $r = (101)$ and, the encoded output word is 1001011101. This coding scheme is capable of detecting both single and multiple errors.

Perhaps the most interesting aspect of the separable cyclic codes is the manner in which they can be generated [27]. The generation circuit is an LFSR or a LCAR which implements the generator polynomial. For the generator polynomial $G(x) = x^3 + x + 1$, figures 4.4 and 4.5 show the corresponding LFSR and LCAR, respectively, while figure 4.6 shows the general structure of the concurrent checking scheme. During the concurrent checking test mode, data patterns used in normal operation also serve as

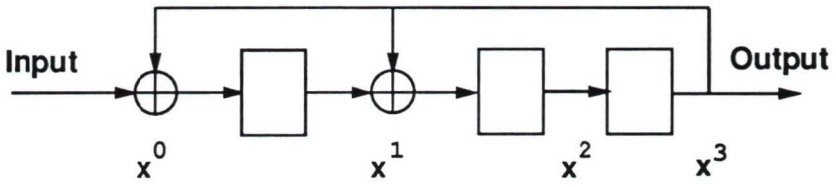


Figure 4.4 An LFSR based on the polynomial $G(x) = x^3 + x + 1$

Figure 4.5 An LCAR based on the polynomial $G(x) = x^3 + x + 1$

test patterns and encoded output words of m-bits are produced. An output word consists of r coding bits and $(m - r)$ data bits. The encoded bits are held in a register while the information bits are shifted into the concurrent data compactor (LFSR/LCAR). After $(m - r)$ clock cycles, the coded bits are computed and compared with the ones previously stored in the register. Any difference in the two compared parts is signaled as the presence of a fault.

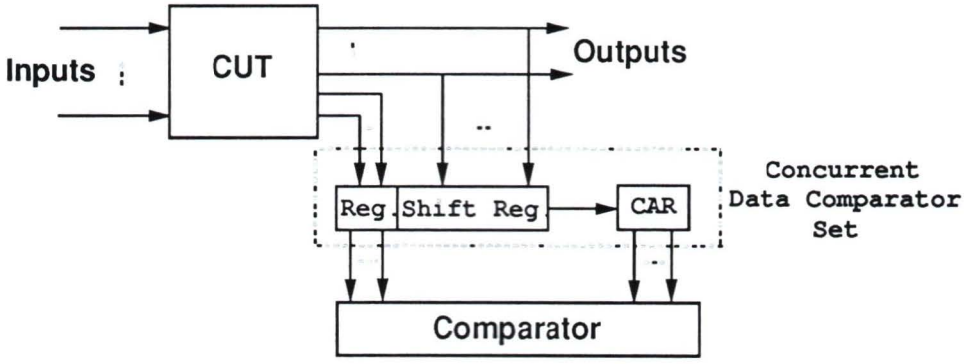


Figure 4.6 A general structure of the concurrent checking scheme

As it can be seen, a data compactor (LFSR/LCAR) is implemented using shift-register cells. Hence, a concurrent checking scheme based on separable cyclic encoding, can share resources that already exist in a boundary scan design thereby introducing less hardware overhead.

Implementation Issues. The proposed [43] concurrent checking scheme requires the following hardware resources:

1. An r-bit register.
2. An $(m-r)$-bit shift register.
3. An r-bit signature analyzer.
4. An r-bit comparator.

In order to implement concurrent checking with low cost, it is suggested the sharing of resources used by the test modes proposed by the ANSI/IEEE Std 1149.1. Recall that the standard defines an input and an output register in the periphery of the CUT. These two registers operate in the same way except that in the BIST mode the input register may generate patterns while, the output register may function as the signature analyzer. Details on how these registers are reconfigured during the various test modes (see figure 4.7) and the requirements which they must meet are discussed in [16, 17].

For the concurrent checking mode, the input and output registers must meet two additional requirements in order to make the sharing of resources with off-line testing feasible:

- The input register must be able to be reconfigured to an r-bit data compactor.

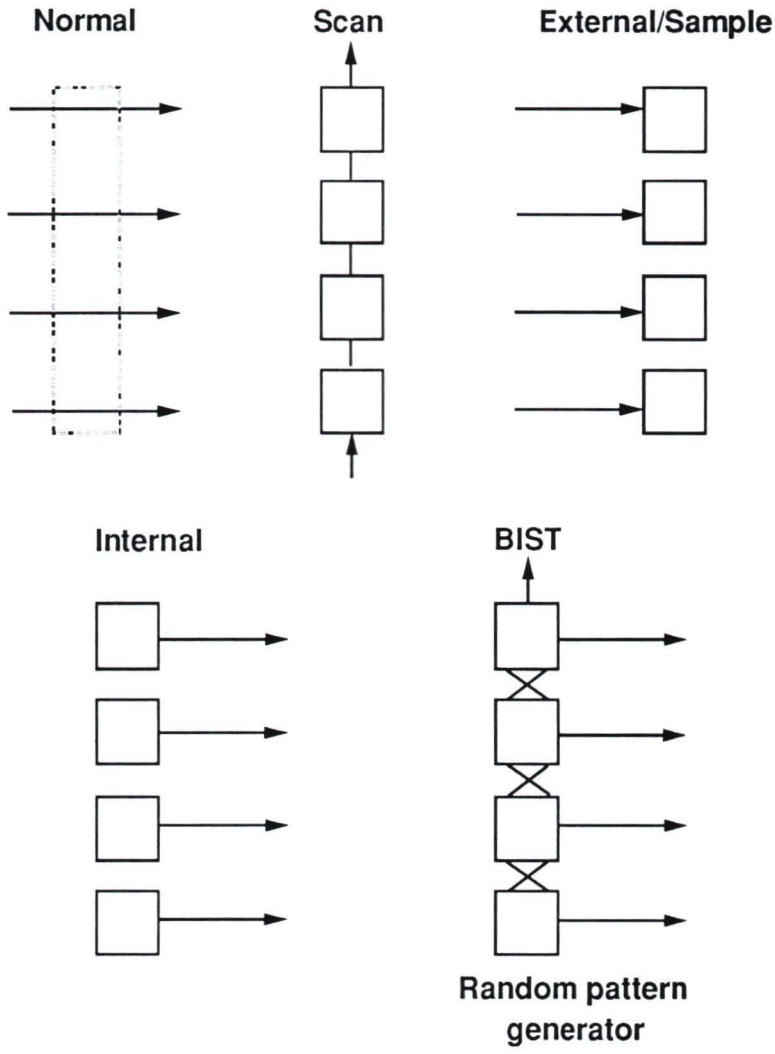


Figure 4.7: Input register modes (four-bit example)

- The output register must function as an r-bit register and $(m-r)$-bit shift register

In [17], it is proposed that the boundary scan cells should incorporate cellular automata principles for the BIST mode. We propose the same idea here for the concurrent checking mode, because LCARs have been found to be superior to LFSRs as test pattern generators, and as is shown in chapters 5 and 6 of this thesis, they exhibit better partitioning behavior than LFSRs.

Note that in order to minimize the hardware overhead the input register of the CUT is reconfigured to an r-bit data compactor during the concurrent checking mode. This idea can be applied in the chip-level, however in the board level we propose one more alternative. More specifically, in this level the input register of the nearest neighboring IC can be used for that purpose if this eliminates some routing problems which may be introduced.

In the following paragraphs, we briefly discuss the performance impact, area overhead and fault coverage of the scheme which merges boundary scan with concurrent checking.

Performance Impact The major impact of the proposed scheme occurs mainly at the dynamic reconfiguration of the resources for concurrent checking. Recall, that during concurrent checking mode the input register is reconfigured to an r-bit data compactor while the output register is reconfigured to an r-bit register and $(m-r)$-bit shift register.

Moreover, if serial data compaction is used, a signature is obtained after a delay of $(m-r)$ clock cycles, which results in a concurrent monitoring only for output words i $(m-r)$, i.e., it misses $(m-r-1)$ words for each one that is checked, $i = 0, 1, 2, \ldots$, so called *semi-concurrent checking*. In [43],

improved structures, namely *output partitioning* and, *output multiplexing*, are presented attempting to reduce the delay and the time intervals of semi-concurrent checking

Area Overhead In determining the overhead involved in adding boundary scan with concurrent checking to an existing design, three aspects are of concern the augmented CUT, the boundary scan logic and, the concurrent checking test logic

Clearly, the area overhead of the augmented CUTs depends on the coding scheme In [43], it is shown that this overhead for the case of PLAs is usually better with LCAR based codes than with residue codes, and always better than Berger encoding

The area overhead introduced when boundary scan is included in the design is estimated in [15] It is found to range from 4% to 13% However, since the proposed scheme shares resources with off-line test (BIST) we are also interested in the boundary scan with BIST overhead This consists of some additional control logic and larger boundary scan cells The estimated cost for a device which includes both boundary scan and BIST ranges from 19% to 34% [15][1], the lowest cost value was obtained when some of the boundary scan logic was put into the I/O buffer ring In [17] where the boundary scan cells were placed adjacent to the pins of the design the estimated chip area overhead ranged from 6 70% to 16 96%

The last aspect of the hardware overhead to be examined concerns the concurrent checking test logic This consists of some additional control logic

[1]In [15], the boundary scan cells were connected during BIST as an LFSR

required to support the dynamic reconfiguration of the resources during test mode and the testing resources themselves. The former is evaluated in chapter 6 of this thesis using the OASIS Design System and the Magic Layout Editor. The results which presented there show that the extra hardware to control the partitioning of a linear machine is only a small percentage of the size of the machine. On the other hand different coding schemes may need very large checkers. For example, a self-checking checker for Berger codes may reach the size of the CUT giving 100% overhead for the testing resources alone.

Fault Coverage. Theoretically, as we have already mentioned, concurrent checking has the ability to detect temporary faults which remain undetected with off-line testing. Moreover the fault coverage of concurrent checking depends on the type of error detection code employed. The quality of the coding scheme can be measured by its error masking (aliasing) probability, given some standard error distribution on its input data stream. For example, a residue code of n-bits has a general aliasing probability of $1/(2^n - 1)$. For an LFSR/LCAR code, this is asymptotic to $1/2^n$. For large values of n, the difference between $1/2^n$ and $1/(2^n - 1)$ is insignificant. But, for small values of n (e.g., $n = 2$ or $n = 3$) the LFSR/LCAR code gives better aliasing probabilities that the residue code. On the other hand, a Berger code provides optimal coverage in the case of a PLA (it detects all unidirectional errors and single internal stuck-at faults) but it is not very effective for general circuits.

4.5 Testing Conventional Logic Using Boundary Scan Components

In this section, we suggest that partitioning of linear machines into bit slices which preserve certain properties may allow on-line and off-line testing of conventional ICs using boundary scan components assembled on printed circuit boards.

It has been argued [20] that "economics involved in designing and producing small logic devices and memories may preclude these parts from ever incorporating boundary scan. As a result, the most common board design approach by far will be one which mixes boundary scan components with clusters of conventional logic ICs and memory devices." If this is the case, we suggest that boundary scan devices should be used for on-line (concurrent) and off-line testing (BIST) of non-scan ICs with encoded outputs. For example, we may use the resources of boundary scan devices surrounding the non-scan one to load in parallel and compact its response.

Chapter 5

Partitioning of Linear Machines

In chapter four we discuss the applicability and usefulness of partitioning linear finite state machines. In this chapter we examine the partitioning behavior of these machines. First we introduce the principles of partitioning and concatenation of polynomials in $GF(2)$. Based on this, we explore the partitioning properties of LFSRs and LCARs. Tables are included showing our results for partitioning irreducible and primitive machines of length 4 up to length 16. We also discuss a bit-sliced implementation of LFSRs and LCARs.

5.1 Definitions and Problem Formulation

We establish a suitable definition for polynomial partitioning using the principle of polynomial concatenation as introduced by Bhavsar [9].

Definition 5.1 Let A and B be two polynomials of degree r and s respectively. Then, A and B are concatenated to realize two polynomials (distinct, if $A \neq B$), C_1 and C_2 of degree $r+s$ as follows

$$C_1 = x^s(A+1) + B \tag{5.1}$$

$$C_2 = x^r(B+1) + A \tag{5.2}$$

Example 5.1 Consider the polynomials $A = x^2+x+1$ and $B = x^4+x^2+1$. The polynomials C_1 and C_2 are formed by operations (5.1) and (5.2), respectively, as follows

$$\begin{aligned} C_1 &= x^4(x^2+x+1+1) + x^4 + x^2 + 1 \\ &= x^6 + x^5 + x^4 + x^2 + 1 \\ C_2 &= x^2(x^4+x^2+1+1) + x^2 + x + 1 \\ &= x^6 + x^4 + x^2 + x + 1 \end{aligned}$$

Since there is a one-to-one correspondence between an LFSR and its characteristic polynomial[1], *LFSR concatenation* is defined by definition 5.1.

[1]The non-zero terms of the characteristic polynomial corresponds to the feedback taps of the LFSR

However, a different definition is required for *LCAR concatenation*

Definition 5.2 [43] Let A and B be two LCARs of length r and s respectively. Then, A and B are concatenated to realize two LCARs (distinct, if $A \neq B$), C_{AB} and C_{BA} of length $r+s$ as follows

$$C_{AB} = A \ll s + B \tag{5.3}$$

$$C_{BA} = B \ll r + A \tag{5.4}$$

where $A \ll s$ denotes the shifting of A to the left by s bits and $+$ denotes the logic OR operation. The two equations are referred as *AB concatenation* and *BA concatenation* respectively.

Definition 5.3 If the polynomial (or machine) C formed by one of the above operations is primitive, then the operation is called a *primitive concatenation* and the polynomials A and B are referred to as *concatenable polynomials*.

Definition 5.4 If a concatenable polynomial is itself primitive, then it is referred to as a *concatenable primitive polynomial*.

Definition 5.5 The concatenation of a polynomial P of degree s with itself n times ($n > 1$) to form a polynomial of degree $n \times s$ is called *self-concatenation*.

Based on the above formulation we define partitioning as follows.

Definition 5.6 If a polynomial C is formed by concatenation of two polynomials A and B, then C is said to be *partitioned* into two parts, A and B.

Definition 5.7 Let $C = x^t + a_{t-1}x^{t-1} + \cdots + a_1x + 1$, be the characteristic polynomial of a length t LFSR which is partitioned into two parts A and B of length r and s respectively, such that $r + s = t$. Then, the two LFSRs A and B which are formed by the partitioning have characteristic polynomials C_1 and C_2, respectively, where

$$C_1 = x^r + C + a_r x^r + a_{r+1}x^{r+1} \cdots + x^t$$

$$C_2 = \frac{C + a_r x^r + a_{r-1}x^{r-1} + \cdots + 1}{x^r} + 1$$

Definition 5.8 Let C' be a length t LCAR represented by the binary vector $(v_1, v_2, \cdots, v_t)$, where v_i is either 0 or 1 denoting a rule 90 cell or a rule 150 cell, respectively. C' is partitioned into two parts A and B of length r and s respectively, such that $r + s = t$. Then, the two LCARs A and B which are formed by the partitioning have characteristic polynomials C_1 and C_2, respectively, where

C_1 is the characteristic polynomial of the transition matrix with diagonal vector $v_A = (v_1, v_2, \cdots, v_r)$, and

C_2 is the characteristic polynomial of the transition matrix with diagonal vector $v_B = (v_{r+1}, v_{r+2}, \cdots, v_t)$

Definition 5.9 The partition of a polynomial C, of degree t into two irreducible polynomials A and B, of degree r and s respectively, such that $r + s = t$, is called an *irreducible partition*. Otherwise, it is called a *reducible partition*

Definition 5.10 The partition of a polynomial C, of degree t into two primitive polynomials A and B, of degree r and s respectively, such that $r+s=t$, is called a *primitive partition*. Otherwise, it is called a *nonprimitive partition*.

Note that, our definition of irreducible (or primitive) partitions insists that both polynomials in the partition are irreducible (or primitive).

In the previous chapter, the applicability and usefulness of partitioning linear finite state machines is discussed. However, very little research has been conducted in this area. Bhavsar [9] first introduced the idea of employing the principle of bit-slicing to design LFSRs cost-effectively, but his study was restricted to the behavior of a small number of primitive and nonprimitive polynomials. More recently, Sun *et al.* [42, 43] have provided useful results concerning the concatenation properties of primitive and nonprimitive LCARs.

However, all the results which have appeared so far in the literature are inadequate to provide a good understanding of the general partitioning properties of LFSRs and LCARs. Moreover, previous investigations do not treat these machines in a uniform way and as a result a fair comparison is not possible. In this thesis, we focus our attention on this issue. The way we approach the problem of partitioning linear machines and the results of our research - a combination of theoretical and experimental results - are presented in the next sections.

5.2 Methodology

Our methodology serves two primary goals. First, it provides a way of achieving better insight into the partitioning properties of LFSRs and LCARs. Second, it allows us to measure and compare their behavior since similar tests are applied to the different machines. In this section, we clarify and discuss our methodology to obtain experimental results which serve the above two purposes.

First, we explore the whole space of linear finite state machines of length n. Two types of machines are considered in this study.

- machines which implement an irreducible characteristic polynomial and,
- machines which implement a primitive characteristic polynomial.

Note that the second set of machines is a subset of the first one. The motivation for such a consideration is twofold.

1. A linear finite state machine with an irreducible or primitive characteristic polynomial exhibits better behavior in terms of aliasing and randomness properties than a reducible machine [51, 52, 53]. Consequently, these are the machines of the most practical use either for compaction, or as pseudo-random pattern generators.

2. An LCAR with an irreducible characteristic polynomial, has the same behavior as the isomorphic LFSR of either Type 1 or Type 2, since they are based on the same characteristic polynomial [40].

Second, we are interested in partitioning a linear finite state machine of length n into two bit-slices of length r and s such that $r + s = n$. Obviously,

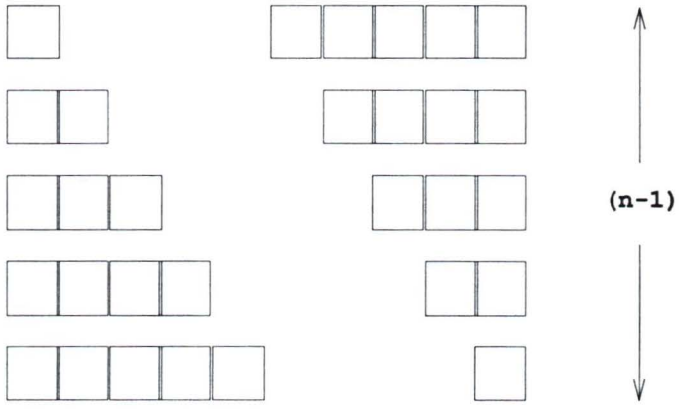


Figure 5.1 All the possible partitions of a length $n = 6$ machine

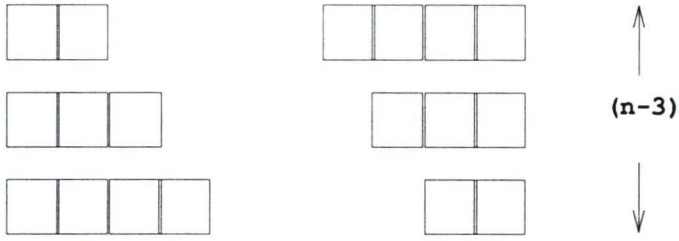


Figure 5.2 All the *proper* partitions of a length $n = 6$ machine

there are other choices as well, a machine of length n can be partitioned into at most n bit-slices. When such a machine is partitioned into two parts there are $(n-1)$ possible partitions (see figure 5.1). However, in this study we require that both bit-slices which result from the partitioning have length at least two, i.e., the partitions into length 1 and length $(n-1)$ submachines are excluded. These are termed *degenerate* partitions. The nondegenerate partitions are termed *proper* ones. Hence, we consider $(n-3)$ proper partitions out of the $(n-1)$ possible ones (see figure 5.2) and only machines with length $n > 3$ are of interest.

Our rationale for excluding degenerate partitions is that an LFSR with length one always implements the characteristic polynomial $x + 1$ which is primitive. On the other hand, a length one LCAR may have either a rule 90 cell or a rule 150 cell, and in the former case, implements the polynomial x. This machine starting with the initial state 1 and taking 0 from its boundaries, goes to state 0 and hence does not have a maximal length cycle[2].

Third, we measure the partitioning behavior of the irreducible and primitive LFSRs and LCARs of length n using two parameters:

1. *ATLOP*: the number of length n machines which have AT Least One Partition with the *desired* property.

2. *PEPP*: the PErcentage of Partitions which have the *desired* Property.

For irreducible machines, a partition with the desired property is an irreducible partition. Hence *PEPP* is defined as

$$\frac{P_{irred} * 100}{(n-3) * irred(n)}$$

where, P_{irred} is the total number of irreducible partitions and $irred(n)$ is the total number of irreducible polynomials of degree n.

For primitive machines, a partition with the desired property is a primitive partition, and hence *PEPP* is defined as

[2]However, the mathematical tool Maple V 4.2 which is used in this study considers the polynomial x as primitive.

$$\frac{P_{prim} * 100}{(n-3) * prim(n)}$$

where, P_{prim} is the total number of primitive partitions and $prim(n)$ is the total number of primitive polynomials of degree n.

In the following sections, the results of our research are presented. The irreducible/primitive LFSRs and the irreducible/primitive LCARs are discussed separately. Possible hardware for dynamically reconfiguring a linear machine is presented later in this chapter.

5.3 Partitioning of LFSRs

The partitioning behavior of LFSRs of length up to 16 which implement irreducible and primitive polynomials is given in tables 5.1 and 5.2, respectively.

An examination of the results in table 5.1 shows that the highest value of *PEPP* is obtained for $n = 4$, and 5. For increasing values of n up to 16 the percentage of irreducible partitions decreases gradually until it reaches the lowest value of 6.21 for $n = 16$. In terms of the number of machines of length n which have at least one irreducible partition (*ATLOP*) the values vary. The lowest value of *ATLOP* is for $n = 4$ where only one out of the three irreducible polynomials, i.e, 33%, has at least one irreducible partition. *ATLOP* (as a percentage) gets the highest value when $n = 8$. There are 22

Degree	*PEPP*	*Number of Irred polynomials*	*ATLOP*
4	33.33	3	1
5	33.33	6	4
6	14.81	9	4
7	19.44	18	10
8	18.00	30	22
9	11.30	56	32
10	11.39	99	55
11	9.67	186	112
12	9.02	335	206
13	7.87	630	356
14	7.46	1161	731
15	6.53	2182	1262
16	6.21	4080	2393

Table 5.1: The partitioning behavior of LFSRs which implement irreducible polynomials

length eight polynomials out of the 30 irreducible ones, i.e. 73.33%, which have the desired property.

The results shown in table 5.2 yield to similar observations for primitive polynomials. Note that in the first row there are entries which are zero. That means, there is no LFSR of length four which has primitive partitions and as a result the values of *PEPP* and *ATLOP* are both zero. These two parameters get their highest values for $n = 5, 6$. For $n = 9$ up to 16, less than 50% of the primitive polynomials have at least one primitive partition.

From tables 5.1 and 5.2, it can be seen that there is no irreducible polynomial of degree five and six which has irreducible partitions which are

Degree	*PEPP*	*Number of Prim polynomials*	*ATLOP*
4	0	2	0
5	33 33	6	4
6	22 22	6	4
7	16 66	18	10
8	12 50	16	10
9	9 72	48	22
10	8 57	60	26
11	6 81	176	80
12	6 32	144	70
13	5 42	630	276
14	4 93	756	336
15	3 94	1800	704
16	4 15	2048	904

Table 5 2 The partitioning behavior of LFSRs which implement primitive polynomials

not primitive

Further discussion on the partitioning behavior of LFSRs is provided in section 6 6

5.4 Partitioning of LCARs

The partitioning behavior of LCARs which implement irreducible and primitive polynomials is reported in tables 5 3 and 5 4, respectively

In table 5 3, the percentage of irreducible partitions (*PEPP*) varies be-

tween 33.33% and 7.39%. From $n = 7$, the longer the machine length, the lower is the value of *PEPP*. In terms of the number of machines of length n which have at least one irreducible partition (*ATLOP*), the values again vary. As for irreducible LFSRs, the lowest value of *ATLOP* is for $n = 4$. In general, less than 80% of the irreducible LCARs have at least one irreducible partition.

LCARs which implement primitive polynomials have similar performance which is shown in table 5.4. For $n = 4$, *PEPP* reaches the highest values that have been seen so far, i.e., 50%. For higher values of n, the percentage of primitive partitions decreases gradually until it reaches the lowest value 4.81 for $n = 16$. Generally, less than 67% of the primitive polynomials have at least one primitive partition.

The first two rows of tables 5.3, and 5.4 show that there is no irreducible LCAR of length four and five which has irreducible partitions.

In general, for irreducible/primitive LFSRs and irreducible/primitive LCARs there in no value of n for which there is at least one proper partition for each machine. However, LCARs have slightly better performance than LFSRs. Further discussion on the partitioning behavior of these machines is provided in section 6.6.

5.5 Design Principles

In this section, the additional hardware which must be added in the design to support the dynamic reconfiguration of the linear machine and its implementation cost are presented.

Degree	*PEPP*	*Number of Irred. Polynomials*	*ATLOP*
4	33.33	3	1
5	16.66	6	2
6	22.22	9	5
7	22.22	18	12
8	17.33	30	21
9	14.88	56	40
10	14.57	99	79
11	12.36	186	132
12	11.40	335	233
13	9.68	630	436
14	9.06	1161	801
15	8.60	2182	1504
16	7.39	4080	2735

Table 5.3: The partitioning behavior of LCARs which implement irreducible polynomials.

In our discussion, we consider the following linear machines:

- Type 1 LFSRs.
- Type 2 LFSRs.
- LCARs with computation rules 90 and 150.

In the figures which are presented, the area enclosed in dotted lines is the extra hardware which provides a machine with the ability to function either as one long machine (*normal mode*) or as two submachines (*partitioned mode*). One additional signal, named *PART* has been incorporated to control the

Degree	*PEPP*	*Number of Prim Polynomials*	*ATLOP*
4	50 00	2	1
5	16 66	6	2
6	27 77	6	4
7	18 05	18	11
8	12 50	16	8
9	12 15	48	27
10	11 42	60	40
11	8 59	176	91
12	7 40	144	78
13	6 38	630	311
14	5 73	756	378
15	5 53	1800	910
16	4 81	2048	1002

Table 5 4 The partitioning behavior of LCARs which implement primitive polynomials

dynamic reconfiguration In the normal mode of the machine, *PART* will be 0 However, when *PART* equals 1 the machine is partitioned into two small submachines

In the case of Type 1 LFSRs, the proposed hardware design is shown in figure 5 3 It can be seen that the extra combinational logic consists of two multiplexers This is required to control the feedback loops during the dynamic reconfiguration In the example shown in figure 5 3, the LFSR which implements the characteristic polynomial $x^5 + x^2 + 1$ can work during the partition mode as two machines which implement the polynomials $x^3 + x^2 + 1$ and $x^2 + 1$, respectively

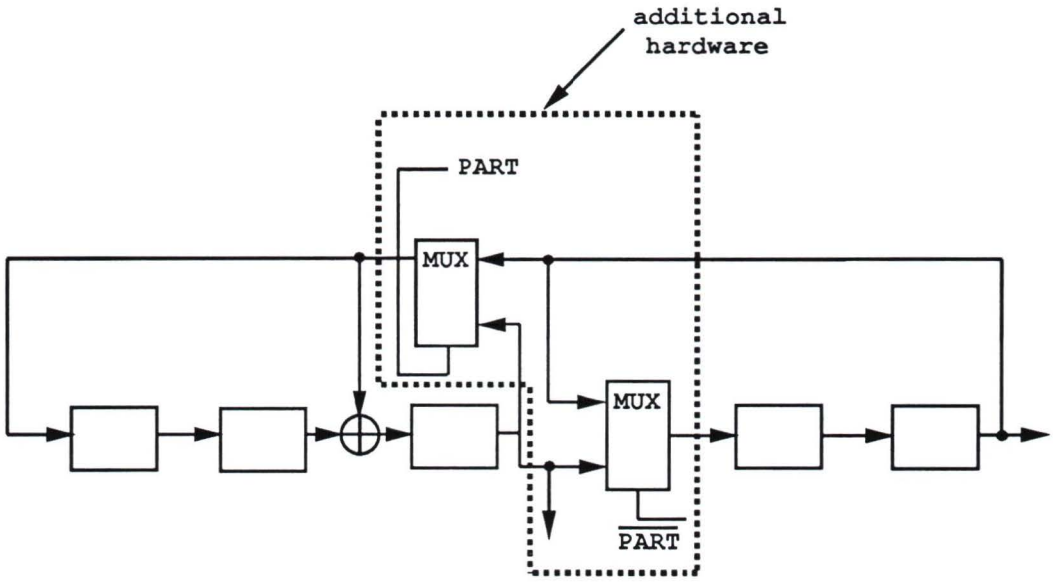


Figure 5.3 Type 1 LFSR. Dynamic reconfiguration of Type 1 LFSR.

The case of a Type 2 LFSR is similar to the case of a Type 1 LFSR. The proposed hardware design is shown in figure 5.4. Later in this section we discuss a particular situation which arises when the partition takes place where there is an embedded XOR gate.

The proposed hardware design for LCARs is shown in figures 5.5 and 5.6. This example shows that by adding the control signal *PART* and two AND gates the LCARs which implement the primitive polynomials x^2+x+1 and x^3+x^2+1 respectively, can function as either two machines or one machine which implements the primitive polynomial x^5+x^3+1. The two AND gates have been added in order to break the communication path between the two cells at the partition point.

It is worth commenting on the situation which is presented in figures 5.7 and 5.8. Figure 5.7 shows a Type 1 LFSR with irreducible characteristic

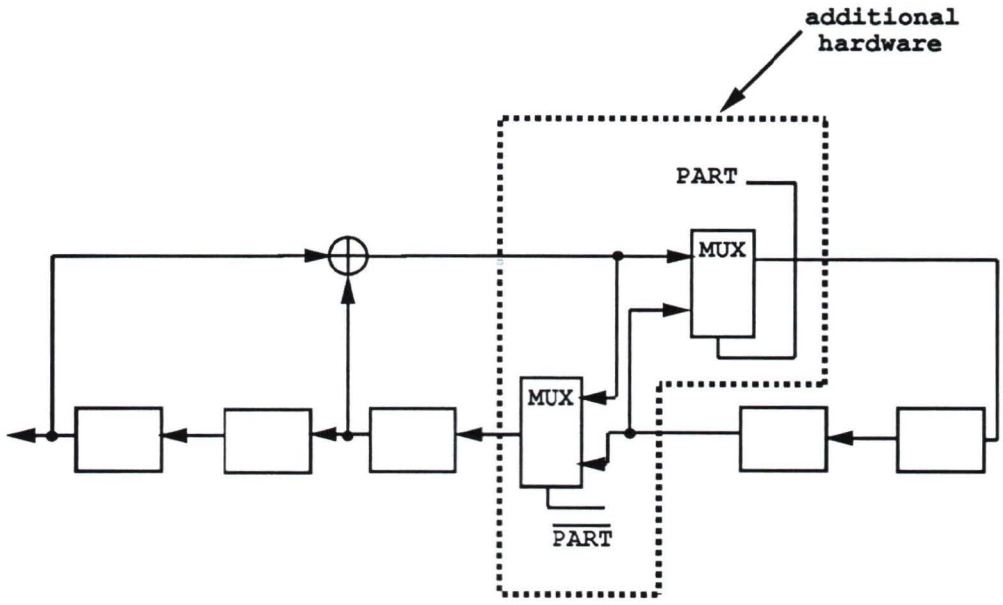


Figure 5.4 Type 2 LFSR Dynamic reconfiguration of Type 2 LFSR

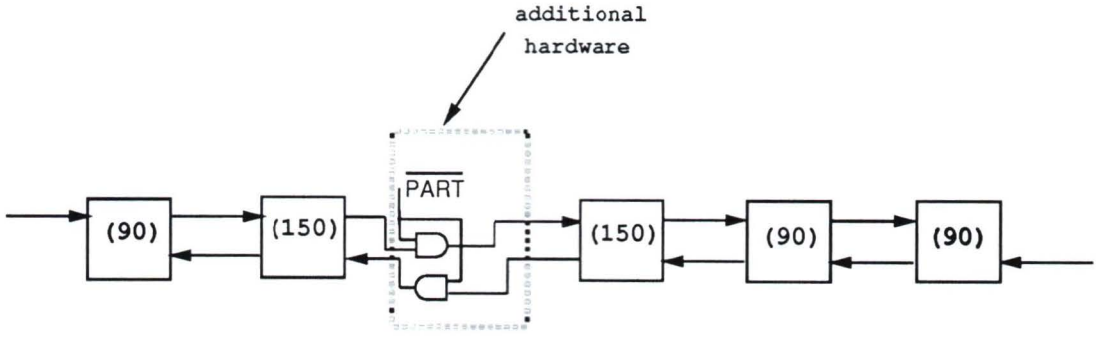


Figure 5.5 Dynamic reconfiguration of an LCAR High level view

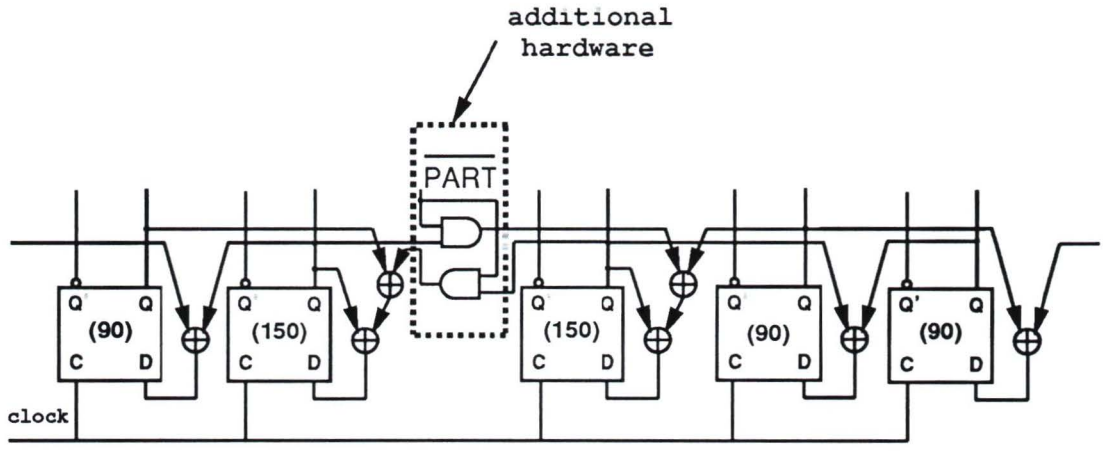


Figure 5.6 Dynamic reconfiguration of an LCAR. Low level view

polynomial $x^4 + x^3 + x^2 + x + 1$. Consider the partitioning of this LFSR into two equal size and maximal length submachines. It appears (see figure 5.8) that we can take advantage of the XOR gate at the break point and reduce the hardware which controls the dynamic reconfiguration by one OR and one AND gate. Hence, the partitioning of a Type 1 LFSR can have minimum implementation cost if there is an XOR gate at the break point. Figures 5.9 and 5.10 presents a minimum cost partitioning for the case of a Type 2 LFSR.

A more detailed evaluation of the hardware overhead is given in the next chapter.

5.6 Discussion

We have examined and evaluated the partitioning behavior of LFSRs and LCARs using two parameters, namely *PEPP* and *ATLOP*. Recall that *PEPP*

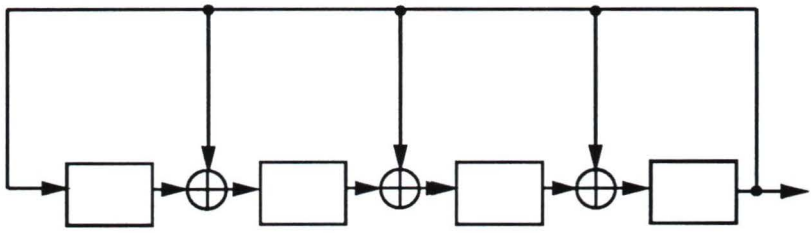


Figure 5.7 Type 1 LFSR The linear machine implements the irreducible polynomial $x^4 + x^3 + x^2 + x + 1$

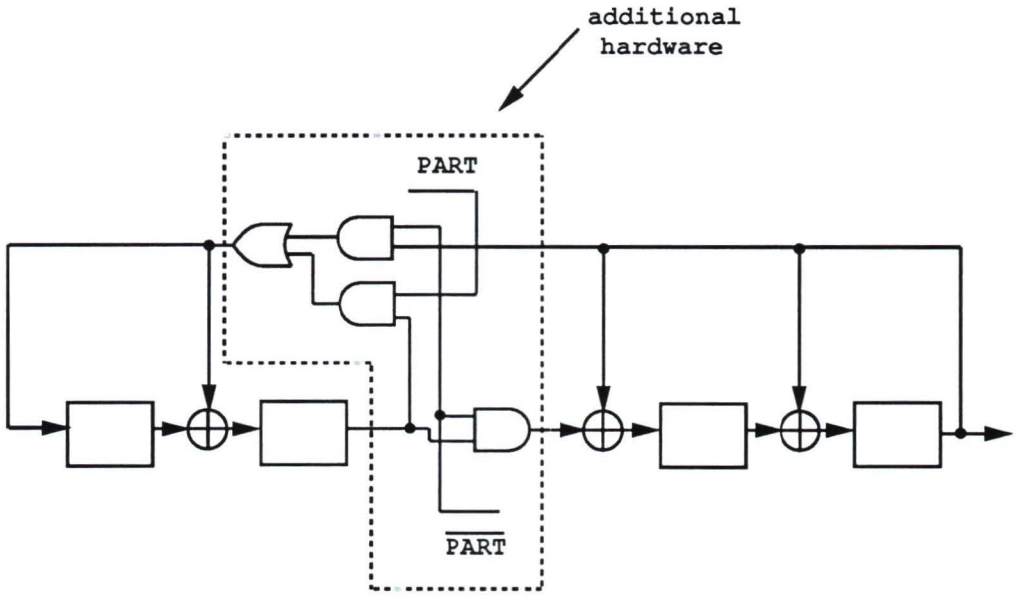


Figure 5.8 Type 1 LFSR The existence of an XOR gate at the break point yields a partitioning with minimum implementation cost

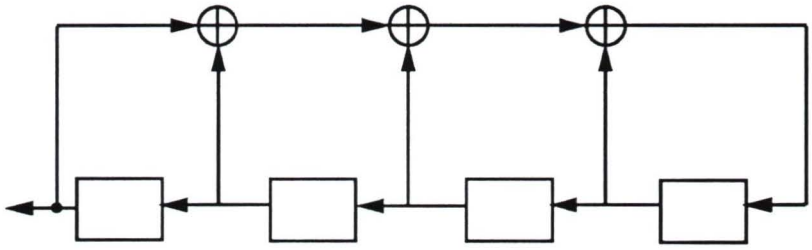


Figure 5.9: Type 2 LFSR. The linear machine implements the irreducible polynomial $x^4 + x^3 + x^2 + x + 1$.

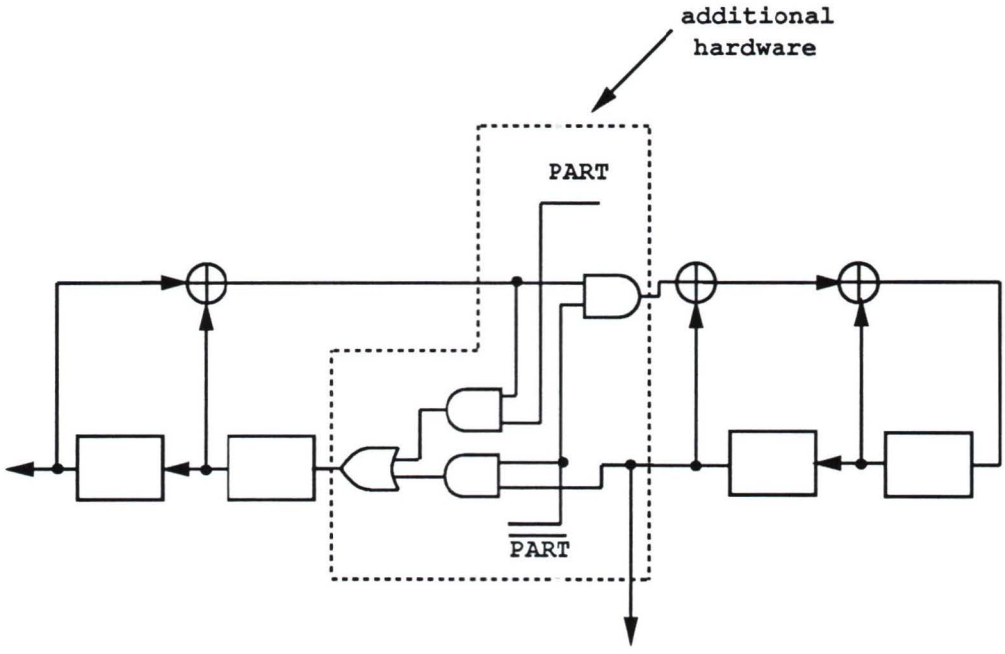


Figure 5.10: Type 2 LFSR. The existence of an XOR gate at the break point yields a partitioning with minimum implementation cost.

is defined as the percentage of proper partitions which are irreducible or primitive and *ATLOP* is defined as the number of machines which have at least one proper partition which is irreducible or primitive. The results of our experiments are listed in tables 5.1 – 5.4.

These tables provide insight into the partitioning behavior of LFSRs and LCARs and can be considered as the source of two hypotheses.

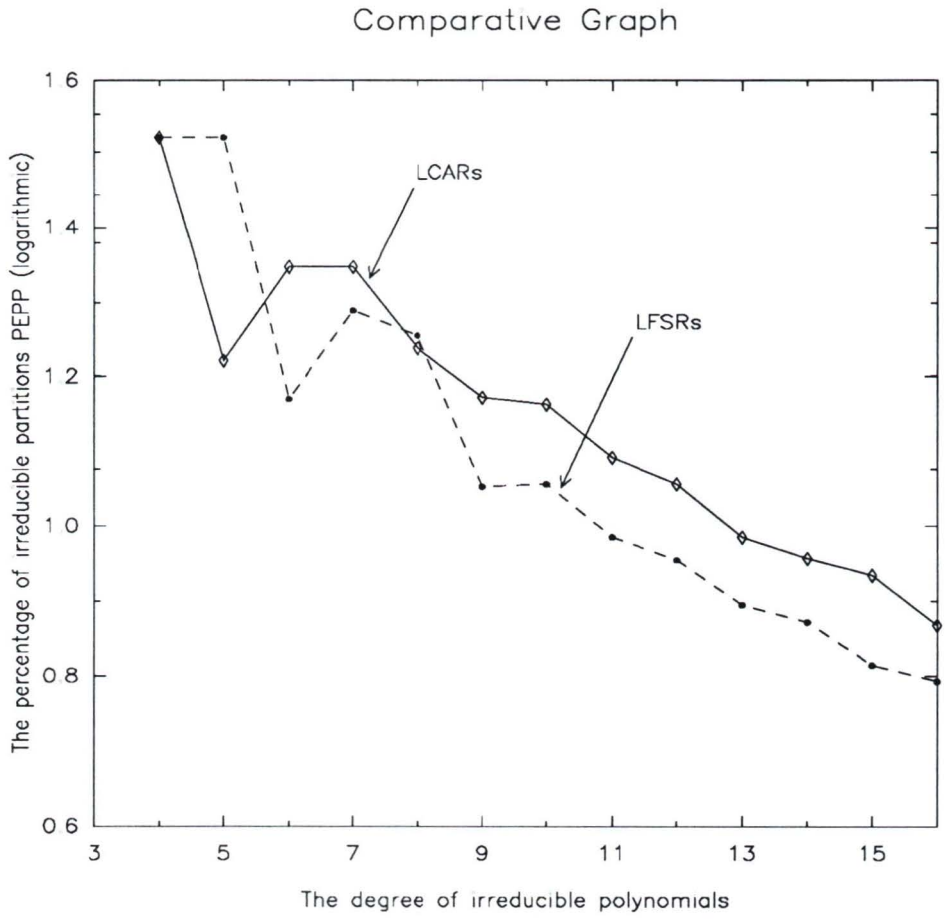
1. Isomorphic LFSRs and LCARs exhibit different partitioning behavior. LCARs appear to have slightly better performance than LFSRs.

2. Despite the fact that irreducible and primitive LFSRs and LCARs appear to have irreducible and primitive partitions, the number of such partitions is quite small.

To support the first argument, comparative graphs for irreducible and primitive machines are shown in figures 5.11 and 5.14 respectively. The y axis is the logarithm of *PEPP* while the x axis gives the length n of the machines or the degree of the characteristic polynomials. The performance of LCARs almost always exceeds the performance of LFSRs[3]. On the other hand, LFSRs appear to have slightly better behavior than LCARs for $n = 5$ and 8. The hypotheses is also supported by the values of *ATLOP*. In tables 5.1- 5.4, it can be seen that LFSRs exhibit higher values for *ATLOP* only for $n = 5$ and 8.

This last observation implies that in cases where *PEPP* has higher values for LFSRs, *ATLOP* is higher as well. This point is interesting because although the values of *ATLOP* go up and down the values of *PEPP* are more

[3]This can be seen from either figure 5.11 or 5.14.



$$PEPP = \frac{P_{irred} * 100}{(n-3) * irred(n)}$$

Figure 5.11: LFSRs - LCARs: The percentage of irreducible partitions.

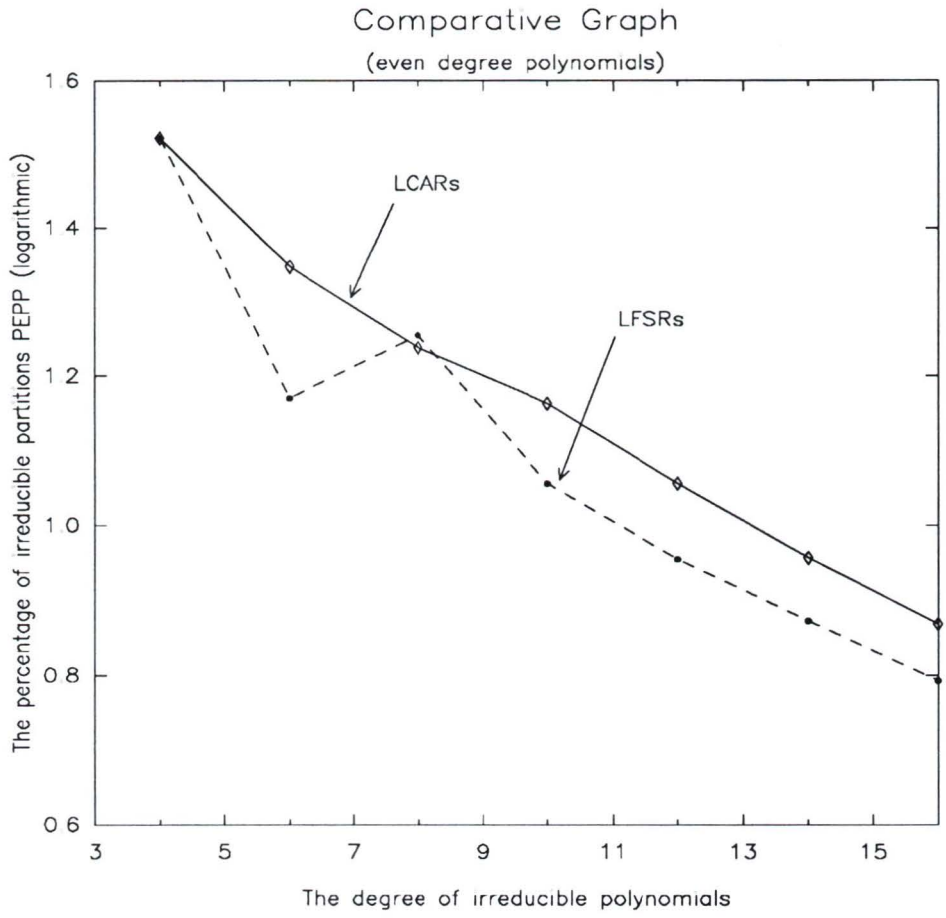
consistent, i e , they decrease as the machine length increases The results imply that *PEPP* is a key and reliable parameter in indicating and evaluating the partitioning behavior of linear machines Moreover, it allows us to extrapolate the performance of machines of length greater than 16

As shown in figures 5 11 and 5 14, the value of *PEPP* for $n > 8$ decreases as the degree of the polynomials increases It is worth observing that in figures 5 12, 5 13 and 5 15, 5 16, where odd and even degree polynomials are treated separately, *PEPP* with $n > 8$ decreases almost linearly This leads us to speculate that linear machines of length longer than 16 exhibit similar partitioning behavior

As a result, even for longer than length 16 machines, one would argue that *PEPP* is likely to decrease while n is increasing Considering that *PEPP* for $n = 16$ reaches the value 4 15%, one would expect that it exhibits very small values for longer machines Self test would look more affordable if the same long maximal length cycle machine could be partitioned into maximal length submachines This could allow the use of the same machine for other purposes, and hence achieve better utilization of the BIST hardware However, the decrease of *PEPP* as n increases means that it becomes increasingly harder for a designer to find a machine which has the desired primitive (or irreducible) partition

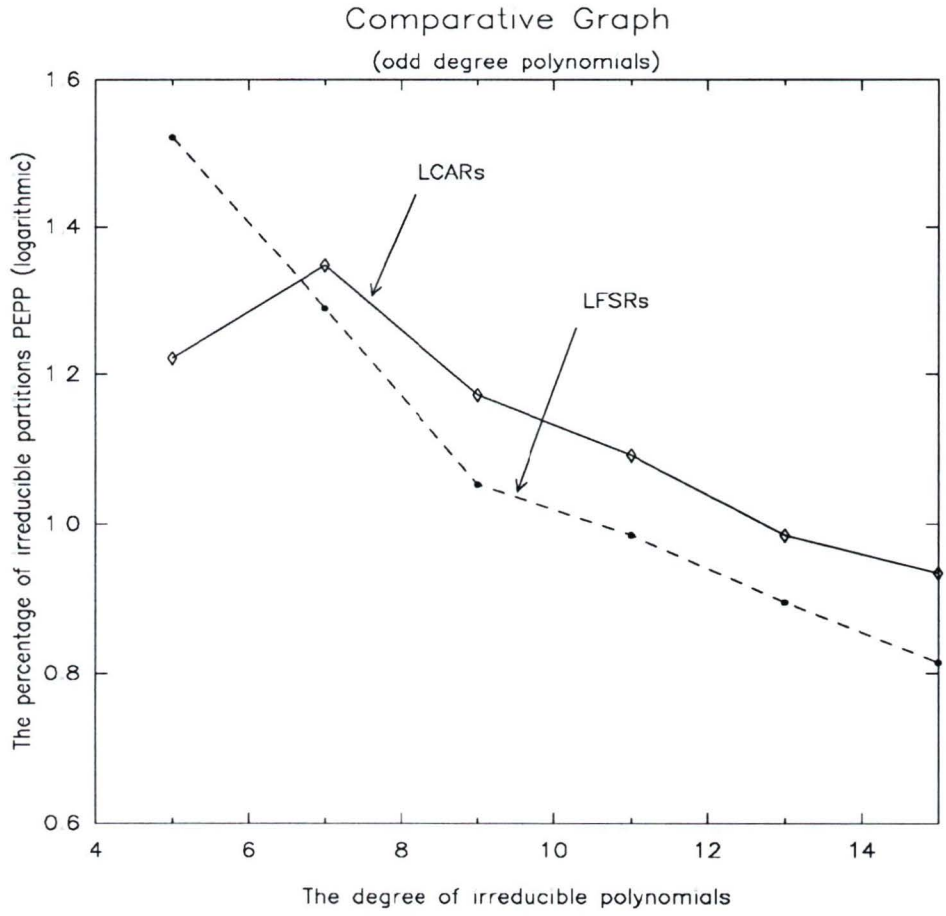
An interesting illustration of this argument is shown in tables 5 5 – 5 8 It can be seen that for $n > 7$ there is only one irreducible LFSR of degree 14, and four irreducible and one primitive LCAR of degree 16 which have five proper partitions which preserve the desired property Recall that for a machine of length n we assume $n - 3$ proper partitions

To conclude this chapter, we investigate the ability of irreducible and



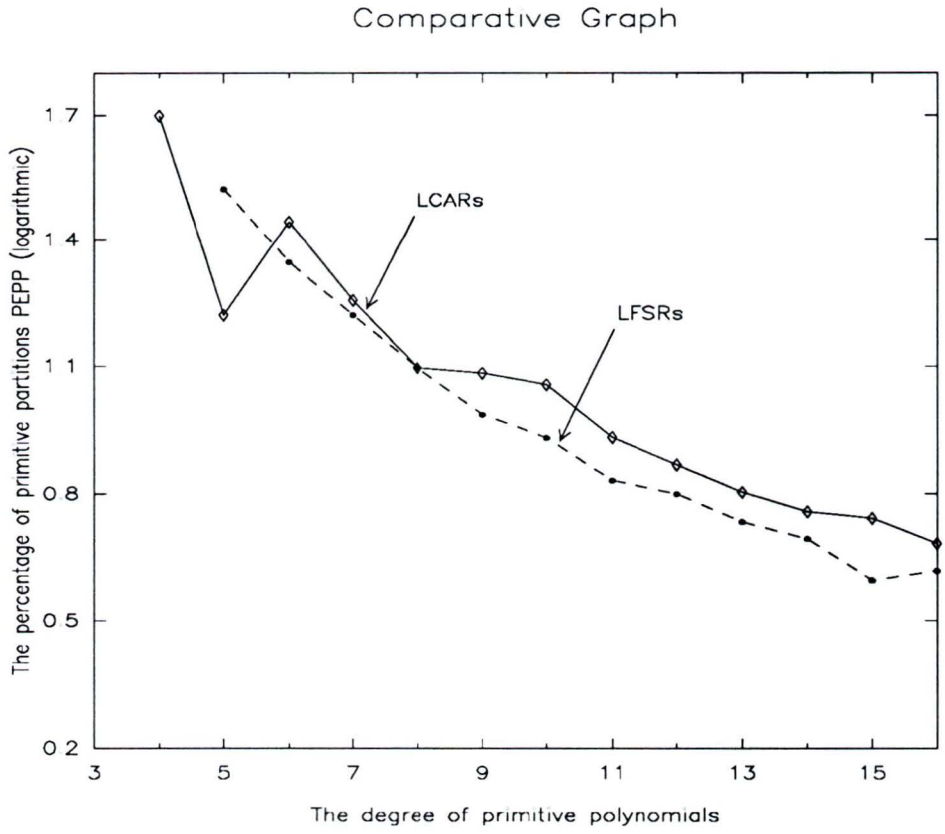
$$PEPP = \frac{P_{irred} * 100}{(n-3) * irred(n)}$$

Figure 5.12 LFSRs - LCARs (even degree) The percentage of irreducible partitions



$$PEPP = \frac{P_{irred} * 100}{(n-3) * irred(n)}$$

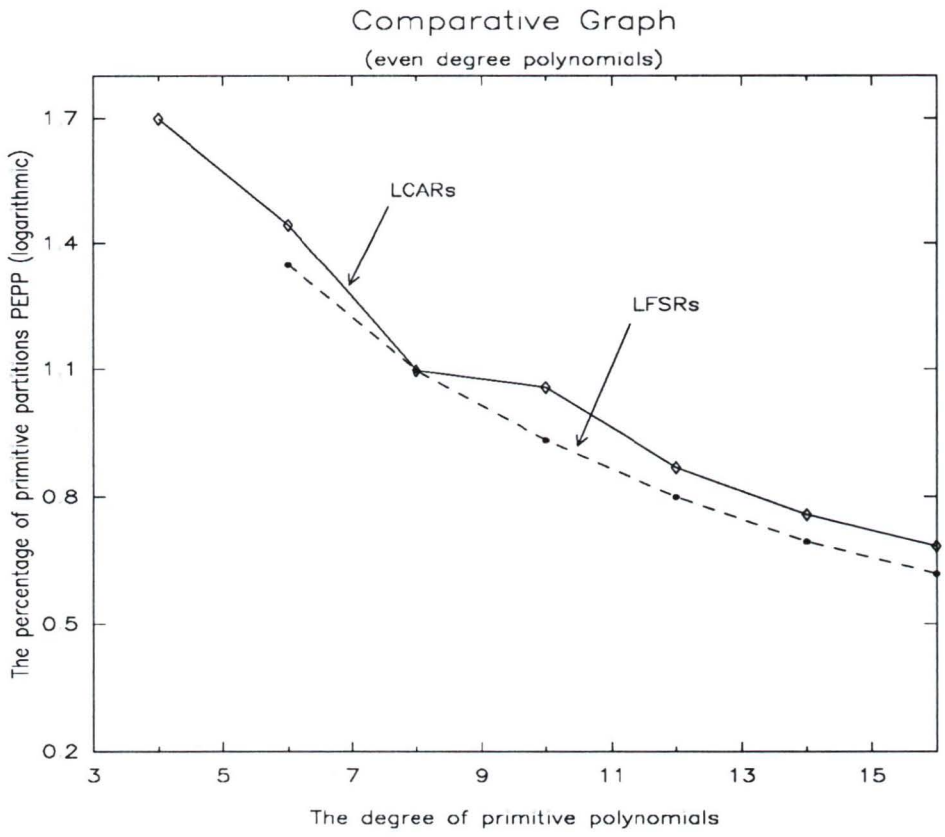
Figure 5.13 LFSRs - LCARs (odd degree) The percentage of irreducible partitions



Note For the LFSRs, the value of PEPP for n=4 is zero and it has been excluded from the graph

$$PEPP = \frac{P_{prim} * 100}{(n-3) * prim(n)}$$

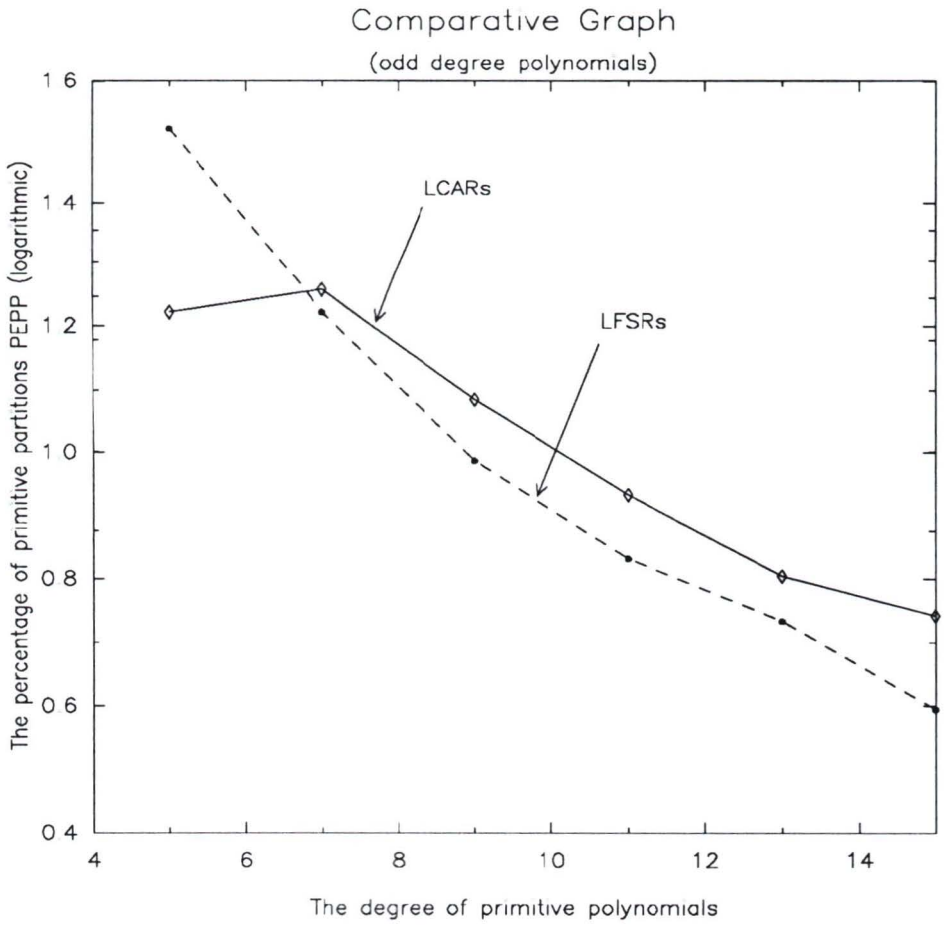
Figure 5 14 LFSRs - LCARs The percentage of primitive partitions



Note: For the LFSRs, the value of PEPP for n=4 is zero and it has been excluded from the graph

$$PEPP = \frac{P_{prim} * 100}{(n-3) * prim(n)}$$

Figure 5.15 LFSRs - LCARs (even degree) The percentage of primitive partitions



$$PEPP = \frac{P_{prim} * 100}{(n-3) * prim(n)}$$

Figure 5.16 LFSRs - LCARs (odd degree) The percentage of primitive partitions

n	Total num of polys	*The number of polynomials of degree n which have P irreducible partitions* ($P = 0, \cdots, n-3$)												
		0	1	2	3	4	5	6	7	8	9	10	11	12	13
4	3	2	1												
5	6	2	4	0											
6	9	5	4	0	0										
7	18	8	6	4	0	0									
8	30	8	17	5	0	0	0								
9	56	24	26	6	0	0	0	0							
10	99	44	33	20	2	0	0	0	0						
11	186	74	82	28	2	0	0	0	0	0					
12	335	129	148	50	8	0	0	0	0	0	0				
13	630	274	234	108	10	4	0	0	0	0	0	0			
14	1161	430	538	168	22	2	1	0	0	0	0	0	0		
15	2182	920	876	324	60	2	0	0	0	0	0	0	0	0	
16	4080	1687	1624	642	121	6	0	0	0	0	0	0	0	0	0

Table 5.5 LFSRs which implement irreducible polynomials

primitive machines to be partitioned into irreducible and primitive submachines. For each degree, there exists a number of polynomials which have this property. This facilitates the use of the same machine for different purposes. However, there may exist situations where this performance may not provide a lot of flexibility in the design. This is because the percentage of irreducible or primitive machines which have irreducible or primitive partitions appears to be fairly small, especially for longer machines. This leads to the question: how can we achieve better performance of LFSRs and LCARs? Can we improve the partitioning behavior of these machines just by allowing minimum modifications in the design? The next chapter investigates and answers these

n	Total num of polys	*The number of polynomials of degree n which have P primitive partitions* ($P = 0, \ldots, n-3$)													
		0	1	2	3	4	5	6	7	8	9	10	11	12	13
4	2	2	0												
5	6	2	4	0											
6	6	2	4	0	0										
7	18	8	8	2	0	0									
8	16	6	10	0	0	0	0								
9	48	26	16	6	0	0	0	0							
10	60	34	16	10	0	0	0	0	0						
11	176	96	66	12	2	0	0	0	0	0					
12	144	74	60	8	2	0	0	0	0	0	0				
13	630	354	220	48	6	2	0	0	0	0	0	0			
14	756	420	268	62	6	0	0	0	0	0	0	0	0		
15	1800	1096	568	124	12	0	0	0	0	0	0	0	0	0	
16	2048	1144	724	162	14	4	0	0	0	0	0	0	0	0	0

Table 5.6 LFSRs which implement primitive polynomials

two questions

n	Total num of polys	*The number of polynomials of degree n which have P irreducible partitions* ($P = 0, \ldots, n-3$)													
		0	1	2	3	4	5	6	7	8	9	10	11	12	13
4	3	2	1												
5	6	4	2	0											
6	9	4	4	1	0										
7	18	6	8	4	0	0									
8	30	9	16	5	0	0	0								
9	56	16	30	10	0	0	0	0							
10	99	20	59	18	2	0	0	0	0						
11	186	54	88	36	8	0	0	0	0	0					
12	335	102	139	79	13	2	0	0	0	0	0				
13	630	194	286	128	20	2	0	0	0	0	0	0			
14	1161	360	498	253	46	4	0	0	0	0	0	0	0		
15	2182	678	890	482	128	4	0	0	0	0	0	0	0	0	
16	4080	1345	1757	799	152	23	4	0	0	0	0	0	0	0	0

Table 5.7 LCARs which implement irreducible polynomials

n	Total num of polys	*The number of polynomials of degree n which have P primitive partitions* ($P = 0, \ldots, n-3$)												
		0	1	2	3	4	5	6	7	8	9	10	11	12	13
4	2	1	1												
5	6	4	2	0											
6	6	2	3	1	0										
7	18	7	9	2	0	0									
8	16	8	6	2	0	0	0								
9	48	21	19	8	0	0	0	0							
10	60	20	34	4	2	0	0	0	0						
11	176	85	66	20	5	0	0	0	0	0					
12	144	66	63	12	3	0	0	0	0	0	0				
13	630	319	228	76	6	1	0	0	0	0	0	0			
14	756	378	291	75	12	0	0	0	0	0	0	0	0		
15	1800	890	658	220	31	1	0	0	0	0	0	0	0	0	
16	2048	1046	754	220	25	2	1	0	0	0	0	0	0	0	0

Table 5.8: LCARs which implement primitive polynomials

Chapter 6

Improving the Partitioning Behavior

In chapter five, we discuss the partitioning behavior of two linear finite state machines. It appears that LCARs have slightly better performance than LFSRs. However, for both machines the percentage of irreducible or primitive partitions is small. This led us to investigate ways to improve their partitioning behavior.

In this chapter, it is demonstrated that better performance for LFSRs and LCARs can be accomplished by allowing minimum hardware modifications. Experimental results are presented to substantiate this claim.

Using the OASIS[1] Design System [32] and the Magic Layout Editor [47],

[1]Open Architecture Silicon Implementation Software distributed by the Microelectronics Center of North Caroline

we estimate the area overhead due to the extra hardware which is required for the dynamic reconfiguration and the modifications

6.1 Introducing Minimum Modifications

In this section the idea of minimum modifications is formalized Some examples are also presented to illustrate this modification of a linear machine

Definition 6.1 *One modification* to an LFSR is the introduction or the elimination of a nonzero term in its characteristic polynomial

Example 6.1 Consider the LFSR with reducible characteristic polynomial $x^{12} + x^9 + x^3 + 1$ With the introduction of the term x^2, the polynomial $x^{12} + x^9 + x^3 + x^2 + 1$ is formed This is primitive and the corresponding LFSR has a maximum length sequence

Example 6.2 Consider the LFSR which implements the reducible polynomial $x^{12} + x^9 + x^8 + x^5 + x^4 + 1$ By eliminating the term x^4 the primitive polynomial $x^{12} + x^9 + x^8 + x^5 + 1$ is formed, and the corresponding LFSR has maximum length sequence

For an LFSR, the introduction of a nonzero term in its characteristic polynomial is defined as a *change from 0 to 1*, while the elimination of a nonzero term is defined as a *change from 1 to 0*

Definition 6.2 *One modification* in a LCAR is the reconfiguration a rule 90 cell to a rule 150 cell or vice versa

Example 6.3 Consider the LCAR which implements the irreducible polynomial $x^{12}+x^{11}+x^{10}+x^8+x^7+x+1$. This is composed of rules $150-150-150-90-90-90-90-90-150-150-90-90$. Recall that the LCAR is also represented by the binary vector [111000001100] where 1 denotes a rule 150 cell and 0 denotes a rule 90 cell. By modifying the seventh rule 90 cell to rule 150 cell a primitive LCAR is obtained. This implements the characteristic polynomial $x^{12}+x^9+x^8+x^3+x^2+x+1$.

Example 6.4 Consider the LCAR with reducible characteristic polynomial $x^{12}+x^{11}+x^9+x^8+x^6+x^5$. This is represented by the binary vector [110111001100]. Suppose now that we modify the first rule 150 cell to a rule 90 cell. The resulting LCAR [010111001100] is primitive with characteristic polynomial $x^{12}+x^7+x^6+x^4+1$.

For an LCAR, the reconfiguration of a rule 90 cell to a rule 150 cell is defined as a *change from 0 to 1*, while the reconfiguration of a rule 150 cell to a rule 90 cell is defined as a *change from 1 to 0*.

The above definitions can be translated into simple hardware modifications which are illustrated in section 6.4.

We now discuss how an irreducible or a primitive machine can be obtained if we allow minimum modifications. This idea has been implemented in Maple [48], and tables have been constructed. All irreducible and primitive machines up to length 16 have been investigated. For each length ($n = 4, \ldots, 16$) the values of *ATLOP* and *PEPP* have been found, allowing one modification.

More specifically, for each irreducible machine all the proper partitions are examined. If there exists a partition where one bit-slice is irreducible

and the other one is reducible, then, we try *one* modification in the reducible part. The goal is to achieve a partition where both bit-slices are irreducible. However, if there exists a partition where both bit-slices are reducible, no modification is allowed. The reason is that we restrict ourself to only *one* modification. Obviously, one could allow more than one modifications. However, the increased area overhead would become unreasonable for some situations.

Similarly, for each primitive machine all the proper partitions are examined. In cases where a partition has only one primitive bit-slice, we try one modification in the other nonprimitive part. The goal is to achieve a partition where both bit-slices are primitive. The restriction of one modification does not allow us to try any change if both bit-slices are nonprimitive.

The experimental results are presented and discussed in the following sections.

6.2 Improving the Partitioning of LFSRs

The partitioning behavior of irreducible and primitive LFSRs up to degree 16, when minimum modifications are allowed, is shown in tables 6.1 and 6.2, respectively.

An examination of the results in these tables yields the following observations.

- Irreducible and primitive LFSRs exhibit better partitioning performance when minimum modifications are allowed.
- Almost all irreducible LFSRs have at least one irreducible partition

(*ATLOP*) For all these machines, up to degree 9 there is at least one partition with the desired property. For higher degrees, up to 16, the number of irreducible LFSRs which do not have irreducible partitions varies between one (for $n = 10$) and 18 (for $n = 16$).

- All primitive LFSRs up to degree eight have at least one primitive partition (*ATLOP*). For higher degrees up to 16, the number of primitive LFSRs which do not have any primitive partitions varies between one (for $n = 12$) and 65 (for $n = 15$).

- One modification results in larger values for *PEPP*. The percentage of irreducible partitions starts from 100% for $n = 4$ and gradually decreases down to 33.52% for $n = 16$. Recall that the corresponding value before we allow any modifications, was 6.21%. A similar improvement can be noticed for the percentage of primitive partitions. It is equal to 100% for $n = 4$ and gradually decreases to 24.50% for $n = 16$. Without any modifications the value of *PEPP* for $n = 16$ was 4.15%.

- For both irreducible and primitive LFSRs, *PEPP* and *ATLOP* reach maximal values for n equals four.

Further discussion on the improved partitioning behavior of LFSRs is given in section 6.6.

6.3 Improving the Partitioning of LCARs

The partitioning performance of irreducible and primitive LCARs up to degree 16 after the introduction of one modification, is reported in tables 6.3

Degree	*PEPP*	*Number of Irred Polynomials*	*ATLOP*
4	100	3	3
5	66 66	6	6
6	74 07	9	9
7	61 11	18	18
8	54 00	30	30
9	50 59	56	56
10	47 47	99	98
11	44 35	186	186
12	41 45	335	330
13	38 98	630	628
14	37 34	1161	1154
15	35 17	2182	2176
16	33 52	4080	4062

Table 6.1 The partitioning behavior of LFSRs which implement irreducible polynomials when one modification is allowed

and 6.4, respectively. Observations which can be derived from the tables are:

- The better partitioning behavior of LCARs with the introduction of one modification is evident from the values of *PEPP* and *ATLOP* shown in tables 6.3 and 6.4.

- Out of the 8795 irreducible LCARs of degree less than or equal to 16, there is *only one* which *done not have* irreducible partitions. For every other irreducible LCAR, there *always exists* at least one irreducible partition.

Degree	*PEPP*	*Number of Prim Polynomials*	*ATLOP*
4	100	2	2
5	66 66	6	6
6	83 33	6	6
7	51 38	18	18
8	45 00	16	16
9	38 54	48	46
10	37 61	60	58
11	33 94	176	171
12	31 55	144	143
13	28 33	630	605
14	26 29	756	735
15	24 45	1800	1733
16	24 50	2048	2011

Table 6 2 The partitioning behavior of LFSRs which implement primitive polynomials when one modification is allowed

- There always exists one primitive partition for each primitive LCAR up to degree 10 For higher degrees up to 16, the number of primitive polynomials which do not have primitive partitions varies between one (for $n = 11$) and five (for $n = 16$).

- One modification allows *PEPP* to exhibit higher values The percentage of irreducible partitions equals 100% for $n = 4$ and it gradually decreases while the degree increases The lowest value of *PEPP* is 44 08% for $n = 16$ Similarly, the percentage of primitive partitions, starting 100% for $n = 4$, gradually decreases to 34 34% for $n = 16$ We must comment that, without any modifications, the corresponding

Degree	*PEPP*	*Number of Irred polynomials*	*ATLOP*
4	100	3	3
5	83 33	6	6
6	81 48	9	9
7	66 66	18	18
8	56 00	30	30
9	60 71	56	56
10	55 84	99	99
11	55 64	186	186
12	50 34	335	335
13	50 12	630	630
14	47 57	1161	1160
15	45 91	2182	2182
16	44 08	4080	4080

Table 6 3 The partitioning behavior of LCARs which implement irreducible polynomials when one modification is allowed

values of *PEPP* for $n = 16$, are 7 39% and 4 81% for irreducible and primitive LCARs, respectively

Further discussion on the improved partitioning behavior of LCARs is given in section 6 6

6.4 Design Principles

In this section we present the hardware design which supports the minimum modifications during the partitioning of a linear machine In the examples used to explain the implementation of one change, the extra hardware is

Degree	*PEPP*	*Number of Prim Polynomials*	*ATLOP*
4	100	2	2
5	83 33	6	6
6	72 22	6	6
7	59 72	18	18
8	52 5	16	16
9	54 16	48	48
10	44 76	60	60
11	44 60	176	175
12	40 12	144	142
13	39 84	630	628
14	37 01	756	752
15	35 16	1800	1787
16	34 34	2048	2043

Table 6 4 The partitioning behavior of LCARs which implement primitive polynomials when one modification is allowed

enclosed in dotted lines Recall, that the *PART* signal which controls the dynamic reconfiguration is set to one during the partition mode In our discussion we consider the following machines

- Type 1 LFSRs
- Type 2 LFSRs
- LCARs with computation rules 90 and 150

For the case of a Type 1 LFSR, the hardware required to implement a change from 0 to 1 and a change from 1 to 0, is shown in figures 6 1 and 6 2, respectively Notice that the change from 0 to 1 (or the introduction

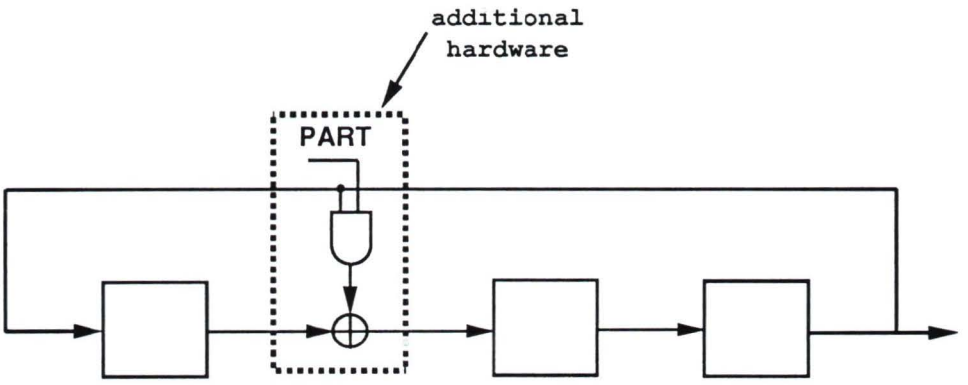


Figure 6.1: Type 1 LFSR. The linear machine which implements the reducible polynomial x^3+1 is reconfigured, after the partitioning, to a maximal length machine with characteristic polynomial x^3+x+1, change from 0 to 1.

of a nonzero term to the characteristic polynomial) requires one extra AND gate and one XOR gate. However, the change from 1 to 0 (or the elimination of a nonzero term from the characteristic polynomial) does not require an extra XOR gate. As a result, one would consider the latter modification as preferable in terms of area cost, to obtain an irreducible or primitive machine.

The case of a Type 2 LFSR is similar to that of a Type 1 LFSR. The proposed hardware for minimum modifications is shown in figures 6.3 and 6.4.

The proposed hardware design which supports minimum modifications for LCARs is shown in figures 6.5 and 6.6. Figure 6.5 shows the hardware required to reconfigure a rule 90 cell to a rule 150 cell. Two extra gates are needed, one AND gate and one XOR gate. However, as can be seen in figure 6.6, the reverse change, implies only one extra AND gate. As a result we conclude that a change from a rule 150 cell to a rule 90 cell is preferable since it introduces a smaller overhead.

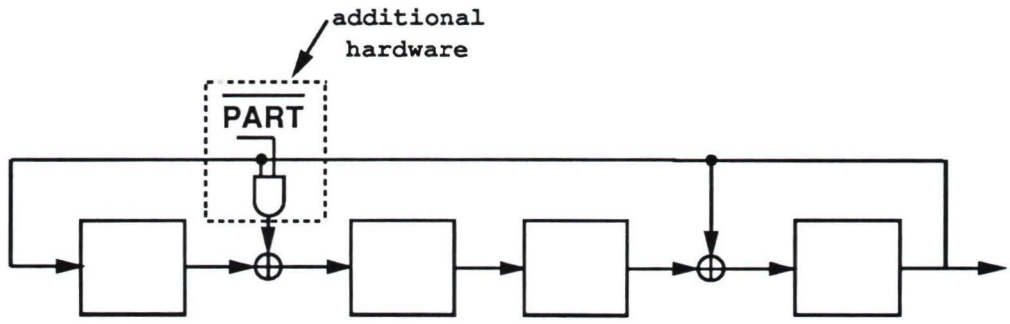


Figure 6.2: Type 1 LFSR. The linear machine which implements the reducible polynomial x^4+x^3+x+1 is reconfigured, after the partitioning, to a maximal length machine with characteristic polynomial x^4+x^3+1, change from 1 to 0.

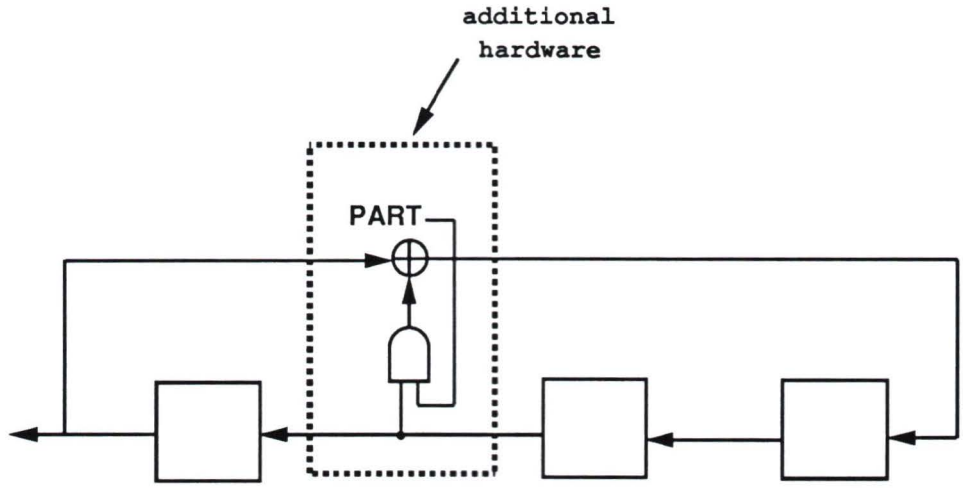


Figure 6.3: Type 2 LFSR. The linear machine which implements the reducible polynomial x^3+1 is reconfigured, after the partitioning, to a maximal length machine with characteristic polynomial x^3+x+1, change from 0 to 1.

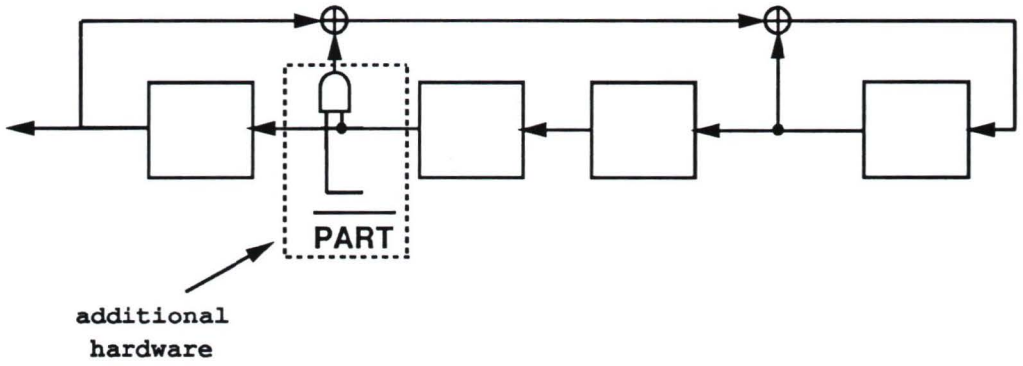


Figure 6 4 Type 2 LFSR The linear machine which implements the reducible polynomial x^4+x^3+x+1 is reconfigured, after the partitioning, to a maximal length machine with characteristic polynomial x^4+x^3+1, change from 1 to 0

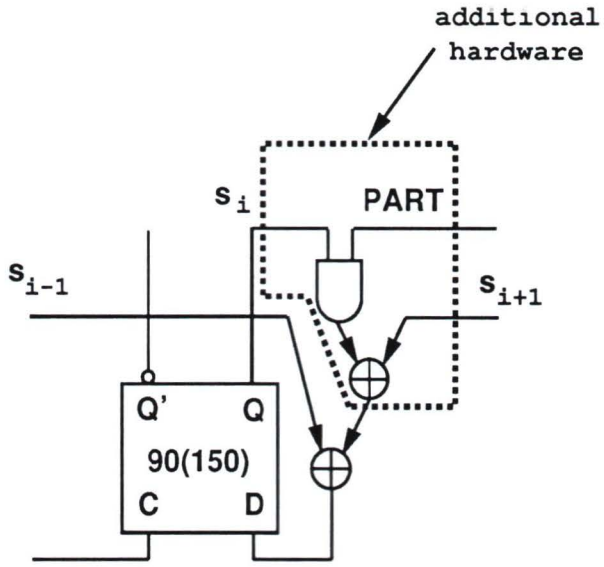


Figure 6 5 LCARs A rule 90 cell which is reconfigured to a rule 150 cell after the partitioning

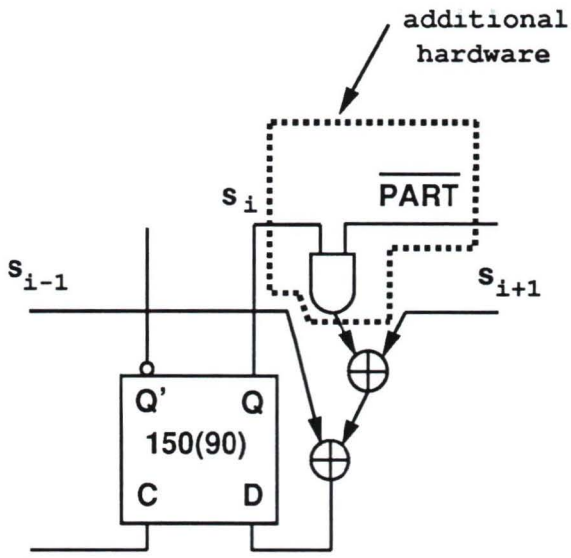


Figure 6.6 LCARs. A rule 150 cell which is reconfigured to a rule 90 cell after the partitioning.

Notice that the minimum modifications require the same hardware for both LFSRs and LCARs.

In the next section, we evaluate more precisely the hardware which supports the dynamic reconfiguration and the minimum modifications.

6.5 Estimation of the Hardware Overhead

This section presents an estimation of the overhead required by the extra hardware needed to support the dynamic reconfiguration and the minimum modifications of a machine. All area overhead estimates were made using the OASIS Design System and the Magic Layout Editor.

First, we compute the exact silicon area (in λ units) occupied by a partic-

ular machine Then, the required hardware for the dynamic reconfiguration and (possibly) for one modification is incorporated in the design and the area of the modified machine is estimated If we denote the area of the original machine as *Area of O M*, and the area of the modified machine as *Area of the M M*, then the overhead is defined by

$$Overhead = \frac{(Area\ of\ M\ M) - (Area\ of\ O\ M)}{Area\ of\ O\ M} \times 100\%$$

We calculate the overhead for three different primitive LFSRs and three different primitive LCARs All the machines under consideration have length 16 and we attempt to partition them into two length eight primitive bit-slices Comparisons for these machines are shown in tables 6 5 and 6 6 The numbers in parentheses present the overhead

Table 6 5 shows the results for the LFSRs The characteristic polynomial and the area of the original LFSR, is shown in the second and third column, respectively The fourth column of the table 6 5 gives the area occupied by an LFSR which also incorporates hardware for partitioning The overhead ranges from 5 79 to 17 72% The overhead for LFSR #1 is the lowest because there is an XOR gate at the break point (notice the x^8 term at its characteristic polynomial) However, only this machine can be partitioned in two length eight primitive bit-slices which both of them implement the same polynomial $x^8 + x^6 + x^3 + x^2 + 1$ LFSR #2 and LFSR #3 require minimum modifications in order to be partitioned into two primitive eight bit-slices

LFSR #2 has a primitive bit slice $x^8 + x^7 + x^6 + x + 1$ and a nonprimitive eight bit-slice $x^8 + x^2 + x + 1$ We allow one change from 0 to 1 in the

LFSR	*Polynomial*	*Original LFSR*	*LFSR + H/W for partition (+%)*	*LFSR + H/W for partition + one change (+%)*
1	$x^{16}+x^{14}+x^{11}+x^{10}$ $+x^8+x^6+x^3+x^2+1$	323 × 504	312 × 552 (5 79)	
2	$x^{16}+x^{10}+x^9+x^7$ $+x^6+x+1$	324 × 480	311 × 544 (8 78)	333 × 568 (21 62)
3	$x^{16}+x^{12}+x^{11}+x^{10}$ $+x^7+x^6+x^5+x^3+1$	299 × 504	308 × 576 (17 72)	311 × 576 (18 87)

Table 6 5 LFSRs The area overhead

nonprimitive part which results to the primitive polynomial $x^8+x^7+x^2+x+1$ The combined overhead (hardware for partitioning and for one change) is 21 62% and it is shown in the fifth column

The partition of LFSR #3 leads to the primitive bit-slice $x^8+x^4+x^3+x^2+1$ and to the nonprimitive one $x^8+x^7+x^6+x^5+x^3+1$ A change from 1 to 0 in the second bit-slice results to the primitive submachine $x^8+x^7+x^5+x^3+1$ and the combined area overhead is 18 87%

Table 6 6 summarizes the measurements for the LCARs Only LCAR #1 can be partitioned into two primitive eight bit slices which implement polynomials $x^8+x^7+x^5+x^3+1$ and $x^8+x^7+x^3+x^2+1$ During partition mode, LCAR #2 and #3 require a change from 0 to 1 and a change from 1 to 0, respectively The combined area overhead (hardware for partitioning and for minimum modifications) ranges from 0 42 to 2 58% and it is shown in sixth column of table 6 6

LCAR	*Polynomial*	*No of rule 150 cells*	*Original LCAR*	*LCAR + H/W for partition (+%)*	*LCAR + H/W for partition + one change (+%)*
1	$x^{16} + x^{12} + x^{11} + x^8$ $+x^7 + x^6 + x^5 + x^2 + 1$	6	424 × 536	415 × 552 (0 79)	
2	$x^{16} + x^{14} + x^{12} + x^{11} + x^{10}$ $+x^9 + x^5 + x^4 + x^2 + x + 1$	4	332 × 672	406 × 536 (-2 45)	**418 × 536 (0 42)**
3	$x^{16} + x^{15} + x^{14} + x^{11} + x^{10}$ $+x^6 + x^5 + x^4 + x^3 + x + 1$	5	418 × 520	422 × 536 (4 06)	416 × 536 (2 58)

Table 6 6 LCARs The area overhead

Looking the results shown in tables 6 5 and 6 6 one would argue that less area overhead is anticipated for LCARs In fact, this is true, but LCARs are almost 40% more expensive than LFSRs in terms of implementation cost

It is worth commenting on the overhead which appears in the second row, fifth column of table 6 6 It appears that incorporating the extra hardware for partitioning LCAR #2 occupies less area! A possible explanation is that after adding more hardware in the design, the cells are arranged in a better way, with less routing area This results in a more square and smaller design This behavior leads us to speculate that when the modified machine is embedded in a larger device the area overhead will be even smaller

Based on these observations we consider that the hardware overhead is affordable This makes the idea of partitioning linear machines and allowing minimum modifications more attractive for a systems designer more options can be offered with low implementation cost

6.6 Discussion

In chapter five, we show that irreducible and primitive LFSRs and LCARs exhibit small percentages of irreducible and primitive partitions. For this reason, we evaluate the performance of these machines after allowing minimum hardware modifications in the design.

Once again, the same parameters, *PEPP* and *ATLOP*, are used. Recall that *PEPP* is defined as the percentage of proper partitions which are irreducible or primitive and *ATLOP* is defined as the number of machines which have at least one proper partition which is irreducible or primitive. The results of our tests after minimum modifications are listed in tables 6.1 – 6.4.

The better partitioning behavior with the introduction of one change is evident from the values of *PEPP* and *ATLOP* shown in these tables. For almost every irreducible or primitive machine there exists at least one irreducible or primitive partition when minimum modifications are allowed. These modifications lead also *PEPP* to exhibit higher values. This is more evident by considering the comparative graphs shown in figures 6.7 and 6.10. The y axis is the logarithm of *PEPP* while the x axis gives the length n of the machines or the degree of the characteristic polynomials.

Figures 6.7 and 6.10 allow us to support the following arguments concerning the partitioning performance and behavior of LFSRs and LCARs.

- LFSRs and LCARs demonstrate significantly better performance when minimum modifications are allowed. In practice, this behavior promises a great economy in hardware since it allows the use of the same machine for more than one purposes.

  An interesting illustration of this argument is given in tables 6.7 –

6.10 Table 6.7 and 6.8 show that in the worst case irreducible and primitive LFSRs can have at most $n-6$ irreducible and $n-8$ primitive partitions, respectively ($n = 15, 16$). Tables 6.9 and 6.10 show that the corresponding value for irreducible and primitive LCARs, is $n-6$ for both cases ($n = 15, 16$). Recall that we assume $n-3$ proper partitions. It is worth here comparing these values with the ones listed in tables 5.5 – 5.8. Without any modifications, in the worst case irreducible/primitive LFSRs can have at most $n-12$ irreducible/primitive partitions out of the $n-3$ proper ones ($n = 15, 16$). The corresponding value for irreducible/primitive LCARs is $n-11$ ($n = 15, 16$).

- Initially, LCARs behave slightly better than LFSRs. After the introduction of minimum modifications LCARs are always superior to LFSRs. Intuitively, this can be explained by considering the fact that we are allowed to try more changes in LCARs. More specifically, in an LCAR of length n we are able to try n changes, i.e., each one of the n cells can be reconfigured either in rule 90 cell or in rule 150 cell. However, in an n length LFSR there are only $n-1$ possible changes. Considering one change in an LFSR as either the elimination or the introduction of a term in the characteristic polynomial, there are only $n-1$ changes. This is because the terms x^n and x^0 always have to appear in the characteristic polynomial.

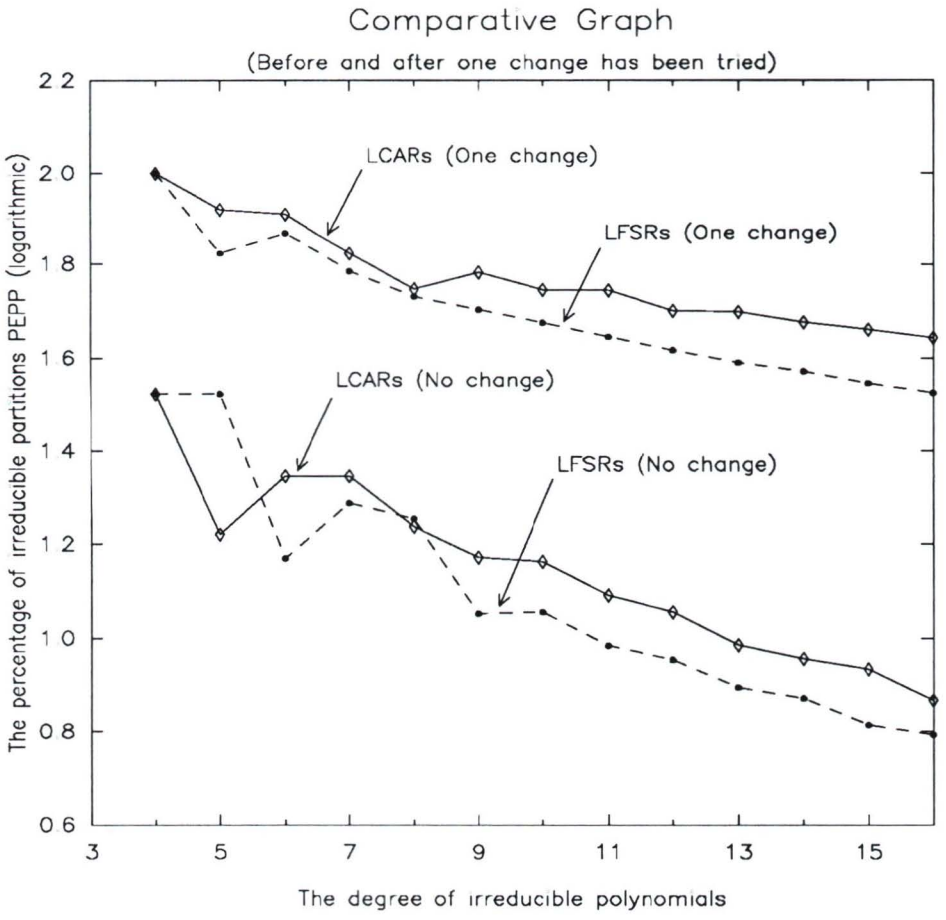
- Initially and after minimum modifications, the percentage of irreducible or primitive partitions (*PEPP*) exhibits consistent behavior. It decreases when the machine length increases. This lead us to the conclusion that even longer than length 16 machines demonstrate the same

n	Total num of polys	*The number of polynomials of degree n which have P irreducible partitions* ($P = 0, \ldots, n-3$)													
		0	1	2	3	4	5	6	7	8	9	10	11	12	13
4	3	0	3												
5	6	0	4	2											
6	9	0	2	3	4										
7	18	0	0	12	4	2									
8	30	0	5	6	12	7	0								
9	56	0	2	20	16	10	8	0							
10	99	1	4	16	38	28	5	7	0						
11	186	0	4	38	48	56	30	8	2	0					
12	335	5	8	47	76	115	54	20	10	0	0				
13	630	2	18	76	154	164	142	60	14	0	0	0			
14	1161	7	24	116	271	284	262	135	54	6	2	0	0		
15	2182	6	44	220	448	566	468	264	136	28	2	0	0	0	
16	4080	18	96	364	782	981	855	588	300	80	14	2	0	0	0

Table 6.7 Irreducible LFSRs after one modification

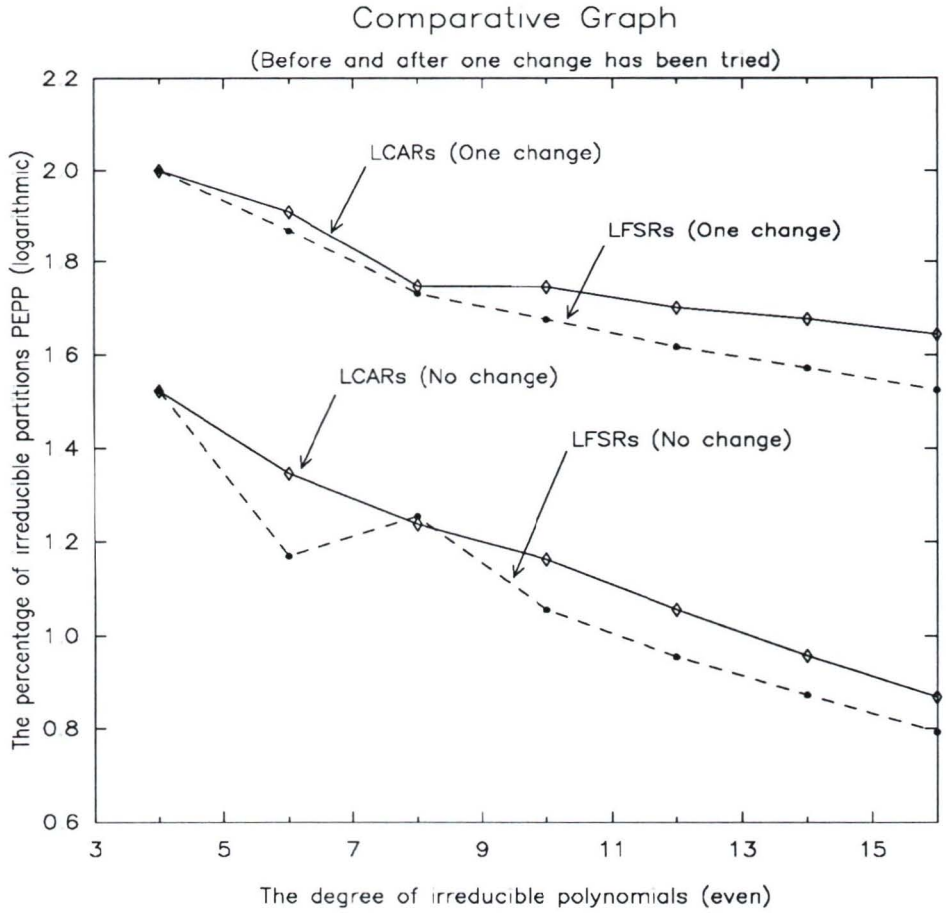
partitioning behavior. This argument is more evident in figures 6.8, 6.9 and 6.11, 6.12, where even and odd degree polynomials are treated separately. These figures show clearly that *PEPP* decreases linearly with the increase in the machine length.

In general, the experimental results which are presented in this chapter, strongly support the fact that the partitioning behavior of LFSRs and LCARs can be improved significantly with low implementation cost. Moreover, it is indicated that LCARs provide better partitioning performance, and hence more implementation options than LFSRs.



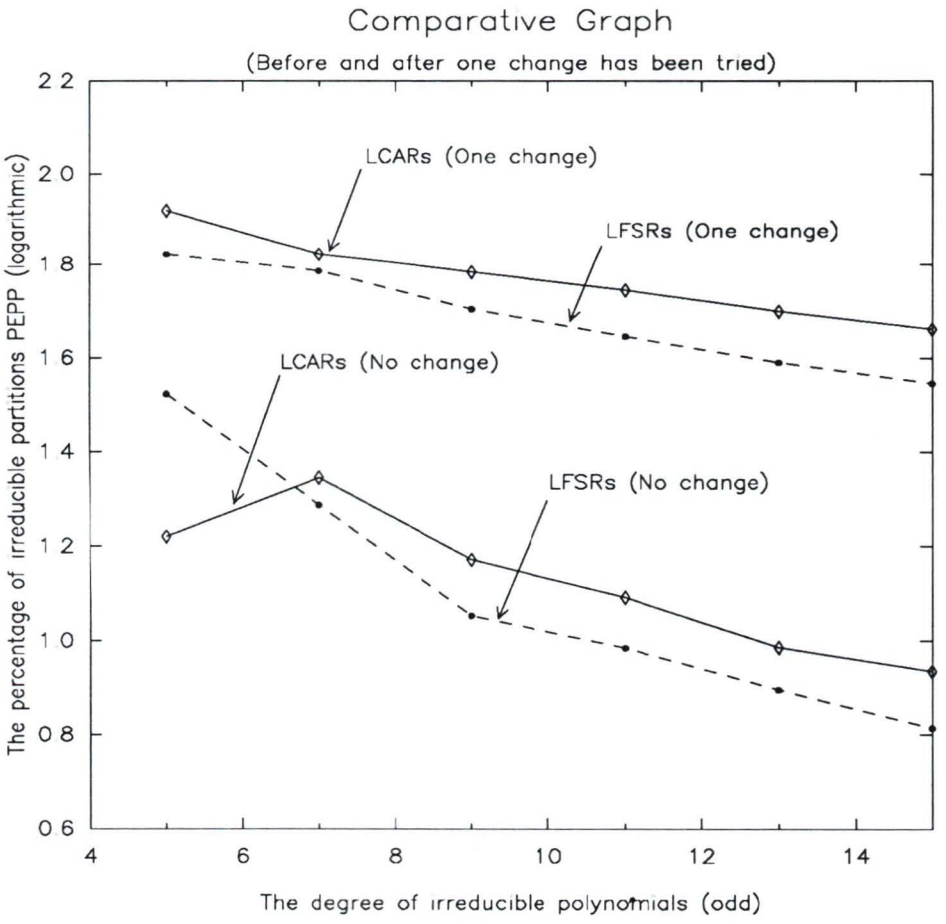
$$PEPP = \frac{P_{irred} * 100}{(n-3) * irred(n)}$$

Figure 6.7 LFSRs - LCARs The improved percentage of irreducible partitions



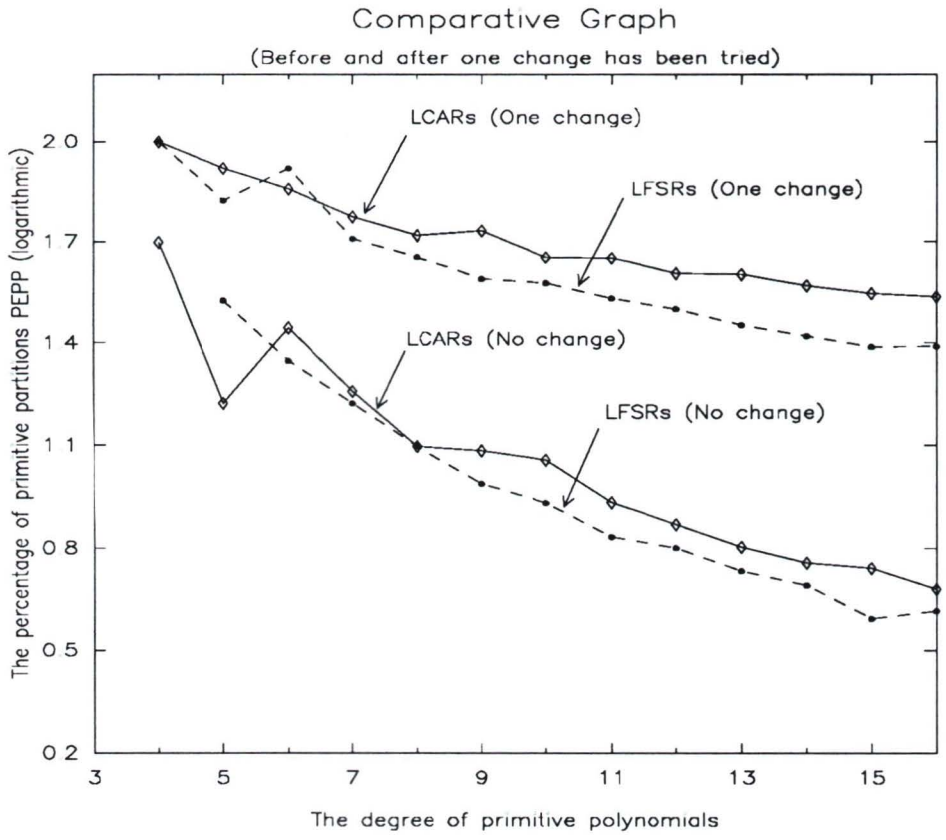
$$PEPP = \frac{P_{irred} * 100}{(n-3) * irred(n)}$$

Figure 6.8 LFSRs - LCARs (even degree) The improved percentage of irreducible partitions



$$PEPP = \frac{P_{irred} * 100}{(n-3) * irred(n)}$$

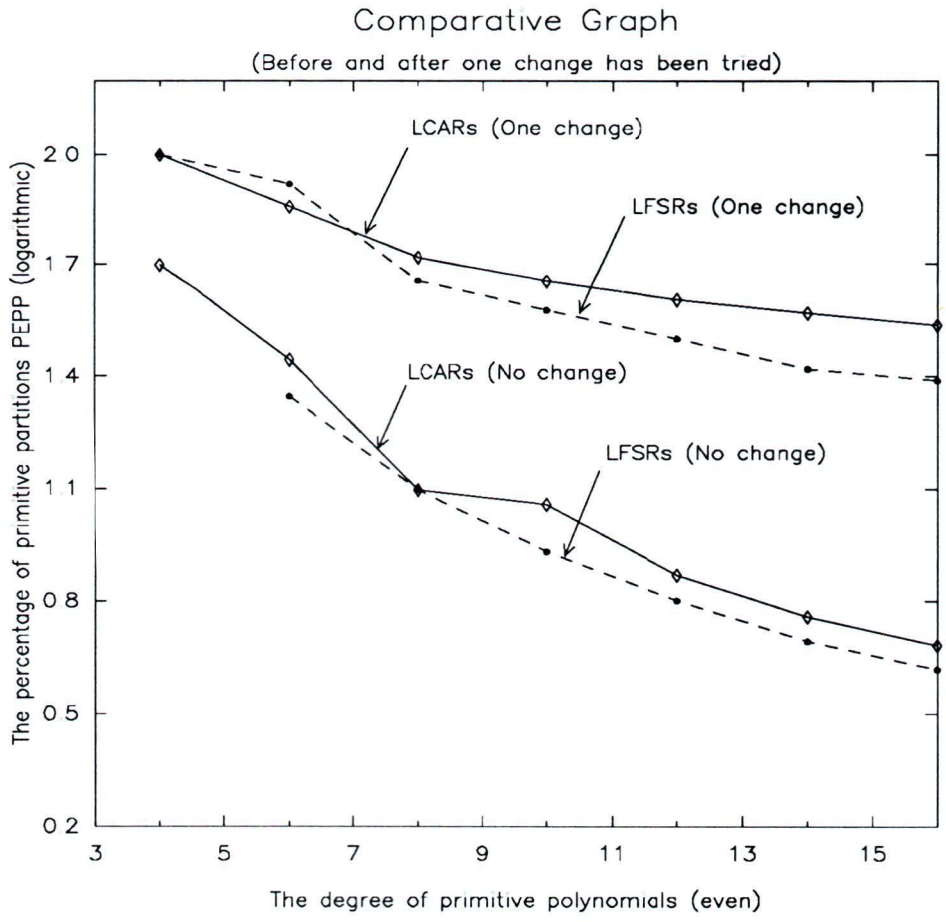
Figure 6.9 LFSRs - LCARs (odd degree): The improved percentage of irreducible partitions



Note For the LFSRs, the value of PEPP for n=4 is zero and it has been excluded from the graph

$$PEPP = \frac{P_{prim} * 100}{(n-3) * prim(n)}$$

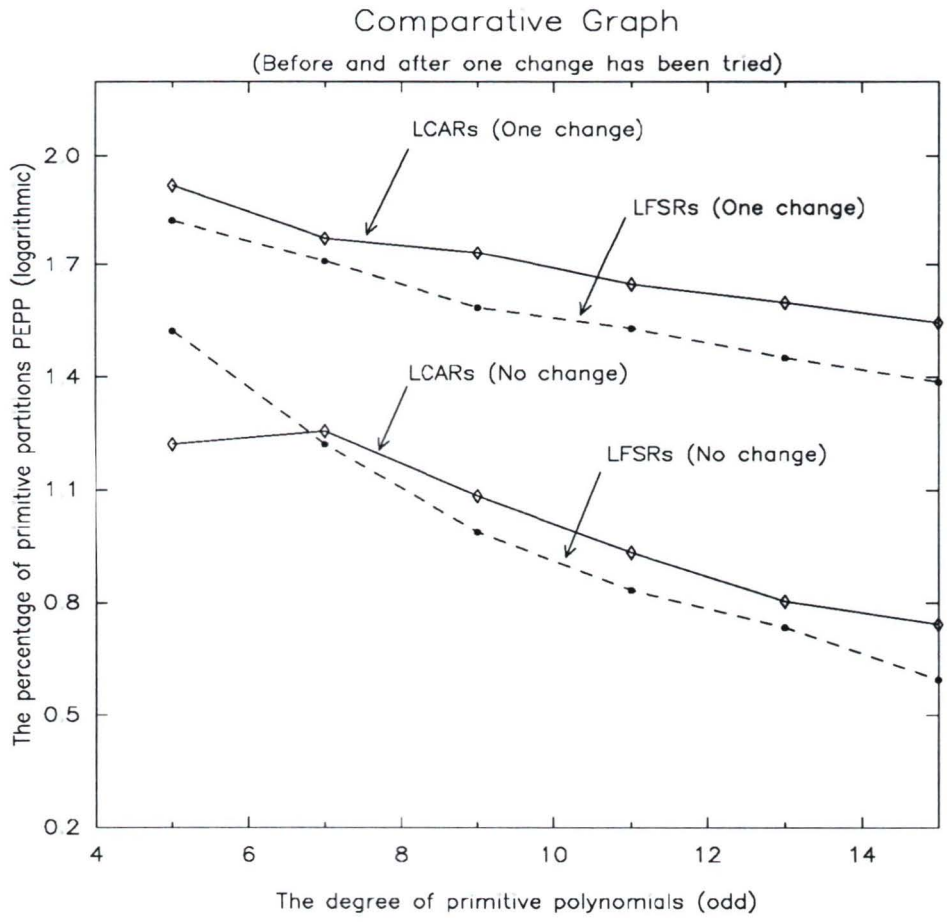
Figure 6.10 LFSRs - LCARs The improved percentage of primitive partitions



Note For the LFSRs, the value of PEPP for n=4 is zero and it has been excluded from the graph

$$PEPP = \frac{P_{prim} * 100}{(n-3) * prim(n)}$$

Figure 6.11 LFSRs - LCARs (even degree) The improved percentage of primitive partitions



$$PEPP = \frac{P_{prim} * 100}{(n-3) * prim(n)}$$

Figure 6.12: LFSRs - LCARs (odd degree): The improved percentage of primitive partitions

n	Total num of polys	*The number of polynomials of degree n which have P primitive partitions* ($P = 0, \ldots, n-3$)													
		0	1	2	3	4	5	6	7	8	9	10	11	12	13
4	2	0	2												
5	6	0	4	2											
6	6	0	0	3	3										
7	18	0	5	9	2	2									
8	16	0	4	5	6	1	0								
9	48	2	14	13	9	6	4	0							
10	60	2	7	15	27	5	4	0	0						
11	176	5	25	50	47	34	14	1	0	0					
12	144	1	17	46	38	28	10	4	0	0	0				
13	630	25	84	155	166	122	64	13	1	0	0	0			
14	756	21	77	198	229	149	62	19	1	0	0	0	0		
15	1800	67	245	428	461	323	170	85	21	0	0	0	0	0	
16	2048	37	201	437	563	436	256	91	25	2	0	0	0	0	0

Table 6.8 Primitive LFSRs after one modification

n	Total num of polys	*The number of polynomials of degree n which have P irreducible partitions* ($P = 0, \ldots, n-3$)													
		0	1	2	3	4	5	6	7	8	9	10	11	12	13
4	3	0	3												
5	6	0	2	4											
6	9	0	0	5	4										
7	18	0	0	6	12	0									
8	30	0	2	8	14	6	0								
9	56	0	0	6	18	22	10	0							
10	99	0	3	6	24	36	24	6	0						
11	186	0	2	6	28	58	56	34	2	0					
12	335	0	2	11	40	109	115	47	7	4	0				
13	630	0	2	10	72	138	176	144	80	8	0	0			
14	1161	1	2	23	108	225	270	313	180	35	4	0	0		
15	2182	0	4	38	138	324	540	610	378	124	26	0	0	0	
16	4080	0	2	53	230	571	914	1054	778	354	111	13	0	0	0

Table 6.9 Irreducible LCARs after one modification

n	Total num of polys	*The number of polynomials of degree n which have P primitive partitions* ($P = 0, \cdots, n-3$)													
		0	1	2	3	4	5	6	7	8	9	10	11	12	13
4	2	0	2												
5	6	0	2	4											
6	6	0	1	3	2										
7	18	0	2	7	9	0									
8	16	0	1	6	7	2	0								
9	48	0	1	10	18	14	5	0							
10	60	0	4	18	13	18	5	2	0						
11	176	1	8	25	54	44	32	12	0	0					
12	144	2	2	23	36	50	24	5	2	0	0				
13	630	2	22	70	136	171	135	78	15	1	0	0			
14	756	4	21	79	164	193	165	105	21	3	1	0	0		
15	1800	13	41	172	353	460	391	258	91	18	3	0	0	0	
16	2048	5	32	169	336	504	516	294	143	39	8	2	0	0	0

Table 6.10: Primitive LCARs after one modification

Chapter 7

A Probabilistic Treatment

In chapters four and five, we present mainly experimental results which give us an insight into the partitioning behavior of LFSRs and LCARs. We test the ability of irreducible and primitive machines to be partitioned into two irreducible and primitive bit-slices, respectively. We measure the number of machines which have this property as well as the percentage of irreducible and primitive partitions for each degree up to 16. These results allow us to make reasonable assumptions concerning the partitioning behavior of longer machines.

In this chapter, we consider a probabilistic treatment of the same problem. We investigate the probability that a randomly chosen irreducible or primitive polynomial has at least one irreducible or primitive partition. Initially, in our research we consider LCARs which have been found to have better partitioning performance than LFSRs. The particular case of LCARs of even degree is also considered.

7.1 Probabilistic Treatment of length n LCARs

In this section, we investigate the probability that a randomly chosen primitive LCAR of length n has at least one primitive partition We denote this probability as $Pr\{ATLOP\}$ It is appropriate at this point to define some suitable notation Let, $Pr\{A\}$ denote the probability that an event A occurs Typically, we describe an event A as follows $A = \{w \mid w$ satisfies the desired property for the event $A\}$, this is read as "A is the set of objects w such that w satisfies the desired property for the event A" Then, the probability that the even A occurs is defined as

$$Pr\{A\} = \frac{number\ of\ objects\ w\ which\ satisfy\ the\ desired\ property}{total\ number\ of\ objects} \quad (7.1)$$

Let us now calculate the probability that an LCAR is primitive

Theorem 7.1 [12, page 9] If an irreducible polynomial p is the characteristic polynomial of an LCAR, then p has exactly two LCARs, which are reversals of each other

From the above theorem, the conjecture proposed by Cattell (1991) follows,

Conjecture 7.1 Each primitive (or irreducible) polynomial is the characteristic polynomial of two LCARs

In [11] an algorithm for computing the LCAR for a given polynomial (if one exists), is described Using this algorithm, K Cattell and J Muzio have proved exhaustively that the conjecture is true for every polynomial up to degree 19 This also holds for every higher degree irreducible polynomial which

has been randomly selected and tested These results give us strong evidence about the correctness of the conjecture for every irreducible polynomial

Based on Equation 7 1 and the proposed conjecture the probability that a randomly chosen LCAR is primitive, is defined as

$$Pr\{an\ LCAR\ of\ length\ n\ is\ primitive\} = \frac{2 \times prim(n)}{2^n} \qquad (7\ 2)$$

where $prim(n)$ denotes the number of primitive polynomials of degree n, and 2^n is the total number of polynomials of the same degree Notice that, the probability that a randomly chosen degree n polynomial is primitive is

$$Pr\{a\ polynomial\ of\ degree\ n\ is\ primitive\} = \frac{prim(n)}{2^n} \qquad (7\ 3)$$

Bardell presents a formula for the number of primitive polynomials of degree n [formula 7 4] This formula is expressed in terms of the Euler φ-function which is defined as the number of positive integers less than or equal to n that are relatively prime to n, for any positive integer n, or

$$\varphi(n) = n \prod_{p|n} \left(1 - \frac{1}{p}\right)$$

where p runs through the primes dividing n In particular, if P denotes a prime, $\varphi(P) = P - 1$ If Q is also a prime, $\varphi(PQ) = (P-1)(Q-1)$ An equivalent definition is $\varphi(n)$ is the number of positive irreducible fractions

not greater than 1, with denominator n. The number of primitive polynomials of degree n is given by

$$\lambda_2(n) = \frac{\varphi(2^n - 1)}{n} \tag{7.4}$$

A detailed development of the formula for $\lambda_2(n)$ can be found in Golomb (1982). The values of $\lambda_2(n)$ for n up to 32 are given in [7].

Now using our formula 7.2 we calculate the probability that a partition into i and $(n - i)$ length bit slices is primitive. Recall that a partition is called primitive when both bit slices are primitive. Hence,

$$\begin{aligned} Pr\{\textit{a partition into } (n-i) \textit{ and } i \textit{ length bit slices is primitive}\} &= \\ Pr\{\textit{the } (n-i) \textit{ bit slice is primitive and the } i \textit{ bit slice is primitive}\} &= \\ Pr\{\textit{the } (n-i) \textit{ bit slice is primitive}\} \times Pr\{\textit{the } i \textit{ bit slice is primitive}\} &= \\ \frac{2 \times prim(n-i)}{2^{n-i}} \times \frac{2 \times prim(i)}{2^i} & \end{aligned} \tag{7.5}$$

Now we can define the probability that a randomly chosen length n LCAR has at least one primitive partition as

$$\begin{aligned} Pr\{\ ATLOP\} = Pr\{\ \textit{an LCAR has at least one primitive partition}\} = \\ Pr\{\textit{a partition into 2 and } (n-2) \textit{ length bit slices is primitive}\} \\ \textit{or,}\quad Pr\{\textit{a partition into 3 and } (n-3) \textit{ length bit slices is primitive}\} \\ \textit{or,} \quad \ldots \end{aligned}$$

or, $Pr\{$*a partition into* $(n-2)$ *and 2 length bit slices is primitive*$\}$

From this observation and the result of Equation 7.5 we could express $Pr\{ATLOP\}$ as

$$\sum_{i=2}^{n-2} \frac{2 \times prim(n-i)}{2^{n-i}} \times \frac{2 \times prim(i)}{2^{i}} =$$

$$\frac{1}{2^{n-2}} \times \sum_{i=2}^{n-2} prim(n-i) \times prim(i) \qquad (7.6)$$

However, formula 7.6 is not precise. This is due to the fact that a polynomial p may have more than one primitive partition. For values of n equal to four and five the formula gives the same results as the experimentally computed ones. There exist higher values of n though, for which the probabilistic results exhibit a deviation from the experimental ones. Notice in the experimental results which appear in table 5.8 that when no modifications are allowed, all primitive LCARs of degree $n = 4,\ 5$ have *only one* primitive partition.

With respect to this fact, we attempt to express the probability under consideration, $Pr\{ATLOP\}$, in a different way. First, consider the probability that a partition into $(n-i)$ and i length bit slices is not primitive. This can be defined as

$Pr_{np} = Pr\{$*a partition into* $(n-i)$ *and* i *length bit slices is not primitive*$\} =$

$Pr\{$*the* $(n-i)$ *length bit slice is not primitive*$\}$ +

$Pr\{$*the* i *length bit slice is not primitive*$\}$ −

$$Pr\{both\ (n-i)\ and\ i\ length\ bit\ slices\ are\ not\ primitive\}$$

Using Equation 7.2, we define the probability that an i length bit slice is not primitive, Pr^{i}_{np}, as follows

$$\begin{aligned} Pr^{i}_{np} &= 1 - Pr\{the\ i\ length\ bit\ slice\ is\ primitive\} \\ &= 1 - \frac{2 \times prim(i)}{2^{i}} \end{aligned} \tag{7.7}$$

Using our notation for the probability that an i length bit slice is not primitive, Pr^{i}_{np}, we may rewrite the definition for Pr_{np} as

$$Pr_{np} = Pr^{n-i}_{np} + Pr^{i}_{np} - Pr^{n-i}_{np} \times Pr^{i}_{np} \tag{7.8}$$

where Pr^{i}_{np} is given by Equation 7.7. Hence, the probability that a randomly chosen LCAR of length n has at least one primitive partition, $Pr\{ATLOP\}$, can be defined as

$$Pr\{ATLOP\} = 1 - Pr\{an\ LCAR\ has\ no\ primitive\ partition\} \tag{7.9}$$

where

$Pr\{an\ LCAR\ has\ no\ primitive\ partition\} =$

$$Pr\{a\ partition\ into\ 2\ and\ (n-2)\ length\ bit\ slices\ is\ not\ primitive\}$$

and, $Pr\{a\ partition\ into\ 3\ and\ (n-3)\ length\ bit\ slices\ is\ not\ primitive\}$

and,

and, $Pr\{a\ partition\ into\ (n-2)\ and\ 2\ length\ bit\ slices\ is\ not\ primitive\}$

From this last definition and Equation 7.8 we obtain

$$Pr\{ATLOP\} = 1 - \prod_{i=2}^{n-2} \left(Pr_{np}^{n-i} + Pr_{np}^{i} - Pr_{np}^{n-i} \times Pr_{np}^{i} \right) \quad (7.10)$$

We consider here the two bit slices as independent since the LCAR is chosen at random. It is clear from formulas 7.7 and 7.10, that the probability $Pr\{ATLOP\}$ is dependent on the number of primitive polynomials of some degree. This number increases like $2^n/n$ when the polynomial degree n increases. The denominator in formula 7.7 increases also exponentially with the increase in the polynomial degree. This justifies our experimental result that the percentage of primitive partitions decreases as the machine length increases.

According to formulas 7.3 and 7.10 the probability P, that a randomly chosen polynomial of degree n is primitive with at least one primitive partition is calculated as

$$P = \left(\frac{prim(n)}{2^n} \right) \times \left(1 - \prod_{i=2}^{n-2} \left(Pr_{np}^{n-i} + Pr_{np}^{i} - Pr_{np}^{n-i} \times Pr_{np}^{i} \right) \right) (7.11)$$

Formula 7.11 can also be used as an approximation for LFSRs because formula 7.2 works for LFSRs too. Based on the probability P, for each degree n, one would expect

$$2^n \times P \tag{7.12}$$

polynomials to be primitive with at least one primitive partition. Moreover, the expected values should be in agreement with the ones we obtain by simulation (table 5.8). However, this is not the case. The results which appear in table 7.1, show that there is a deviation between the expected and the computed results. This fact leads us to conclude that formula 7.10 is only a good approximation of the desired probability. However, this approximation is superior to the one obtained by formula 7.6. Moreover, it gives reasonable results for all values of n examined.

Formula 7.10 is derived assuming that the percentage of degree n polynomials which are primitive with at least one primitive partition is always the same in any subset of the 2^n possible polynomials.

Stated somewhat differently, we expect that the number of polynomials which have the desired property are in some sense "uniformly distributed" throughout the entire space. However, this is not the case. The random distribution of the polynomials which have the expected property allows only a good approximation of the desired probability. This is clear in table 7.2 where formula 7.12 is calculated for $n = 7$. $P_{i,j}$ is the probability that the partition into i and j $(= n - i)$ length bit slices is primitive. $E_{i,j}$ and $C_{i,j}$ is the expected and experimentally computed number of primitive polynomials which can be partitioned into i and j primitive bit slices. The values of

Degree	*Predicted values*	*Computed values*
4	0 5	1
5	2 62	2
6	2 55	4
7	8 90	11
8	7 86	8
9	24 08	27
10	29 59	40
11	81 02	91
12	68 05	78
13	284 54	311
14	335 42	378
15	764 08	910
16	914 75	1002

Table 7 1 Formula 7 10 Predicted versus computed results

$E_{i,j}$ are obtained according to the following formulas where 18 is number of primitive polynomials of degree seven

$$
\begin{aligned}
E_{2,5} &= P_{2,5} \times 18 \\
E_{3,4} &= P_{3,4} \times (18 - E_{2,5}) \\
E_{4,3} &= P_{4,3} \times (18 - E_{2,5} - E_{3,4}) \\
E_{5,2} &= P_{5,2} \times (18 - E_{2,5} - E_{3,4} - E_{4,3})
\end{aligned}
$$

All the formulas which are provided in this section involve primitive poly-

Partition	2/5	3/4	4/3	5/2	*Total*
Expected values	$P_{2,5} = 0.187$ $E_{2,5} = 3.375$	$P_{3,4} = 0.125$ $E_{3,4} = 1.828$	$P_{4,3} = 0.125$ $E_{4,3} = 1.599$	$P_{5,2} = 0.187$ $E_{5,2} = 2.099$	8.90
Computed values	$C_{2,5} = 3$	$C_{3,4} = 3$	$C_{4,3} = 2$	$C_{5,2} = 3$	11

Table 7.2 Formula 7.10 A detailed calculation for $n = 7$.

nomials. However, we can extend all the presented results to irreducible polynomials.

7.2 Probabilistic Treatment of length $2k$ LCARs

In this section we restrict our discussion into $2k$ length LCARs. We calculate the probability both halves of a randomly chosen $2k$ length LCAR is primitive. This probability is especially interesting when $2k$ is equal to 8, 16 or 32. These are the data word sizes which dominate present day computer systems. The desired probability can be calculated as

$Pr\{a$ *length* $2k$ *LCAR has both halves primitive*$\} =$

$Pr\{$*the first k length bit slice is primitive*$\}$

and, $Pr\{$*the second k length bit slice is primitive*$\}$.

This probability can be expressed by the following formula

$$Pr\{a\ length\ 2k\ LCAR\ has\ primitive\ halves\} = P_{ph}$$
$$= \left(\frac{2 \times prim(k)}{2^k}\right)^2 \qquad (7.13)$$

Now we can calculate the probability a randomly chosen length $2k$ LCAR is primitive with primitive halves, P_{pph}, according to formulas 7.2 and 7.13 as follows

$$P_{pph} = \left(\frac{2 \times prim(2k)}{2^{2k}}\right) \times \left(\frac{2 \times prim(k)}{2^k}\right)^2 \qquad (7.14)$$

Based on formula 7.14 the expected number of length $2k$ LCARs which are primitive with primitive halves is

$$2^{2k} \times P_{pph}$$

Recall, the results which appear in this section and concern primitive polynomials can be easily extended to irreducible ones

7.3 Discussion

In this chapter we consider a probabilistic treatment of the partitioning of LCARs It is concluded that in particular cases only an approximation of the expected probability is possible This is due to the random distribution of the polynomials which have the desired property

Chapter 8

Conclusion

In this chapter, we discuss the contributions of our research and interesting aspects of it that should be further investigated.

8.1 Contributions

Linear feedback shift registers (LFSRs) are used extensively in two capacities in DFT (Design For Testability) and BIST (Built-In Self-Test) designs. One application is as a source of pseudorandom binary test sequences, the other is as a means to carry out response compaction - known as signature analysis. Recently ([24, 33]), one dimensional linear cellular automata registers (LCARs) have been proposed as an alternative to LFSRs for both purposes.

Ongoing research in the area of VLSI testing is extensively studying the behavior of LFSRs and LCARs as random pattern generators and signature analyzers. However, little work has been done in investigating the partition-

ing behavior of these devices In the context of digital circuit testing, it is important if a maximal length cycle machine can be partitioned into a number of smaller maximal length cycle submachines Such a property shows promise of great economy in hardware and facilitates testing

In this thesis, the partitioning behavior of LFSRs and LCARs has been studied Our methodology provides a means to evaluate and compare these machines in terms of their partitioning performance The contributions of our research can be summarized as follows

1 It is the first systematic study of the partitioning behavior of LFSRs and the isomorphic LCARs We have exhaustively examined all machines up to degree 16 It has been found that there exists a number of irreducible and primitive machines which can be partitioned into smaller irreducible and primitive submachines These results allow us to extrapolate the behavior of machines of length greater than 16

2 Our methodology has shown that the partitioning properties of linear finite state machines is dependent on their implementation It has been determined that LCARs have better partitioning behavior than LFSRs

3 We suggest that better partitioning performance for LFSRs and LCARs can be accomplished with low implementation cost After allowing minimum hardware modifications, almost every irreducible (or primitive) machine has at least one irreducible (or primitive) partition

There are many open questions that should be examined These are discussed in the next section

8.2 Future Work

There are many interesting aspects of our research that should be further investigated. The most important of them are the following.

- A mathematical basis for predicting if a given degree n polynomial can be partitioned into k and $n-k$ length primitive or irreducible bit slices is a subject for further study.

- It is known that a linear finite state machine with an irreducible or primitive characteristic polynomial exhibits better behavior in terms of aliasing and randomness properties than a reducible machine. An investigation whether this is also true in terms of partitioning behavior, would be interesting.

- In our study we consider all possible irreducible or primitive LCARs. It would be interesting to determine whether minimal rule 150 LCAR have better partitioning performance than other LCAR structures, such as LCAR with 50% rule 150 cells.

- More detailed layout studies are required to fully assess the cost of partitioning a linear machine and allowing minimum modifications.

Research on the above topics will enhance our understanding on the behavior of LFSRs and LCARs. Moreover, it may result in interesting suggestions for the selection of the appropriate machine for self-test.

Bibliography

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

[2] S M I Adhem and H T Mouftah Built-in self-test Techniques, attributes and selection *Journal of Semicustom ICs*, pages 3–11, March 1991

[3] S B Akers and W Jansz Test set embedding in a built-in self-test environment In *Proc of the International Test Conference*, pages 257–263, 1989

[4] K Akiyama and K K Saluja A method of reducing aliasing in a built-in self-test environment *IEEE Transactions on Computer-Aided Design*, 10(4) 548–553, April 1991

[5] P H Bardell Calculating the effects of linear dependencies in *m*-sequences used as test stimuli In *Proc of the International Test Conference*, pages 252–256, 1989

[6] P H Bardell Analysis of cellular automata used as pseudorandom generators In *Proc of the International Test Conference*, pages 762–

768, 1990

[7] P H Bardell, W H McAnney, and J Savir *Built-in Test for VLSI Pseudorandom Techniques* John Wiley and Sons, New York, 1987

[8] R G Bennetts and A Osseyran IEEE standard 1149 1-1990 on boundary scan History, literature survey, and current status *Journal of Electronic Testing Theory and Applications (JETTA)*, pages 11–25, March 1991

[9] D K Bhavsar Concatenable polydividers Bit-sliced LFSR chips for board self-test In *Proc of the International Test Conference*, pages 88–93, 1985

[10] F Brglez, C Gloster, and G Kedem Hardware-based weighted random patterns generation for boundary scan In *Proc of the International Test Conference*, pages 264–274, 1989

[11] K Cattell and J Muzio A linear cellular automata algorithm Algorithm summary Technical report, Computer Science Department, University of Victoria, June 1991

[12] K Cattell and J Muzio A linear cellular automata algorithm Theory Technical report, Computer Science Department, University of Victoria, June 1991

[13] M Damiani, P Olivo, M Favalli, S Ercolani, and B Ricco Aliasing in signature analysis testing with multiple input shift registers *IEEE Transactions on Computer-Aided Design*, 12(9) 1344–1353, December 1990

[14] A K Das, D Saha, A R Chowdhury, S Misra, and P P Chaudhuri Signature analysers based on additive cellular automata In *Proc of the IEEE Fault Tolerant Computing Symposium*, pages 265–272, 1990

[15] K D Fitch and J Kane Application of boundary scan and full-chip bist to a 3 ASIC chip set In *Proc of the Custom Integrated Circuits Conference*, pages 17 5 1–17 5 4, 1991

[16] C S Gloster and F Brglez Boundary scan with cellular-based built-in self-test In *Proc of the International Test Conference*, pages 138–145, 1988

[17] C S Gloster and F Brglez Boundary scan with built-in self-test *IEEE Design and Test of Computers*, pages 36–44, February 1989

[18] H W Gschwind and E J McCluskey *Design of Digital Computers* Springer, 1975

[19] S K Gupta, D K Pradhan, and S M Reddy Zero aliasing compression In *Proc of the International Test Conference*, pages 254–263, 1990

[20] P Hansen Testing conventional logic and memory clusters using boundary scan devices as virtual ate channels In *Proc of the International Test Conference*, pages 166–173, 1989

[21] J P Hayes Check sum methods for test data compression *Journal of Design Automation Fault-Tolerant Comput* , pages 3–17, 1976

[22] J P Hayes Transition count testing of combinational logic circuits *IEEE Transactions on Computers*, 25 613–620, June 1976

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

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

[25] IEEE, New York *ANSI/IEEE Std 1149.1 – Standard Test Access Port and Boundary-Scan Architecture*, 1990

[26] A Ivanov, C W Starke, V K Agarwal, W Daehn, M Gruetzner, and T W Williams Iretative algorithms for computing aliasing probabilities *IEEE Transactions on Computer-Aided Design*, 10(2) 260–265, February 1991

[27] B Johnson *Design and Analysis of Fault Tolerant Digital Systems* Addison-Wesley, 1989

[28] C M Maunder and R E Tulloss An introduction to the boundary scan standard ANSI/IEEE Std 1149.1 *Journal of Electronic Testing Theory and Applications (JETTA)*, pages 27–42, March 1991

[29] E J McCluskey Built-in self-test structures *IEEE Design and Test of Computers*, pages 29–36, April 1985

[30] E J McCluskey Built-in self-test techniques *IEEE Design and Test of Computers*, pages 21–28, April 1985

[31] E J McCluskey *Logic Design Principles - with emphasis on testable semicustom circuits* Prentice-Hall, New Jersey, 1986

[32] MCNC's Center for Microelectronics, North Carolina *Open Architecture Silicon Implementation Software (OASIS)*, 1990

[33] D M Miller Review of built-in self-test methodologies In *Proc of the Canadian Conference on Electrical & Computer Eng* , pages 375–378, 1988

[34] K P Parker Compact testing Testing with compressed data In *Proc of the 6th Annual Fault-Tolerant Computing Symposium*, pages 93–98, 1976

[35] W W Peterson *Error-Correcting Codes* M I T Press, 1961

[36] W Pries, A Thanailakis, and H C Card Group properties of cellular automata and vlsi applications *IEEE Trans on Computers*, C-35(12):1013–1024, December 1986

[37] G Russell and I L Sayers *Advanced Simulation and Test Methodologies for VLSI Design* Van Nostrand Reinhold (International) Co Ltd, 1989

[38] J Savir Syndrome-testable design of combinational circuits *IEEE Transactions on Computers*, 29 442–451, June 1980

[39] M Serra Fault detection using concurrent checking In *Proc of the Canadian Conference on VLSI*, pages 74–79, October 1988

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

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

[42] X. Sun and M. Serra. Linear cellular automata and their concatenation properties. Technical report, Computer Science Department, University of Victoria, October 1989.

[43] X. Sun and M. Serra. Concurrent checking and off-line data compaction testing with shared resources in plas. *Journal of Semicustom ICs*, pages 8–16, September 1990.

[44] A. K. Susskind. Testing by verifying Walsh coefficients. *IEEE Transactions on Computers*, C-32:198–201, February 1983.

[45] J. Turino. *Design to Test*. Van Nostrand Reinhold, 1990.

[46] A. Tzidon, I. Berger, and M. Yoeli. A practical approach to fault detection in combinational networks. *IEEE Transactions on Computers*, 27(10):968–971, October 1978.

[47] University of California, Berkeley. *Magic Layout Editor*, 1986.

[48] University of Waterloo, Canada. *Maple, a system for symbolic mathematical computation*, 1990.

[49] J. Wakerly. *Error Detection Codes, Self-Checking Circuits and Applications*. Elsevier North-Holland Inc., 1978.

[50] L T Wang Autonomous linear feedback shift register with on-line fault-detection capability In *Proc of the IEEE Fault Tolerant Computing Symposium*, pages 311–314, 1982

[51] T W Williams, W Daehn, M Gruetzner, and C W Starke Comparison of aliasing errors for primitive and non-primitive polynomials In *Proc of the International Test Conference*, pages 282–288, 1986

[52] T W Williams, W Daehn, M Gruetzner, and C W Starke Aliasing errors with primitive and non-primitive polynomials In *Proc of the International Test Conference*, pages 637–644, 1987

[53] 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 Computer-Aided Design*, 7(1), January 1988

[54] T W Williams and K P Parker Design for testability – a survey *IEEE Trans on Computers*, C-31(1) pp 2–15, January 1982

[55] S Wolfram Statistical mechanics of cellular automata *Rev Modern Phys* , 55 601–644, 1983

[56] S Wolfram University and complexity in cellular automata *Physica*, 10D 1–35, 1984

[57] S Zhang and D M Miller A comparison of LFSR and cellular automata BIST In *Proc of the Canadian Conference on VLSI*, pages 8 4 1–8 4 9, 1990

[58] Z. Zhang and R. D. McLeod. Estimating stuck-open fault coverage for CMOS combinational circuits. In *Proc. of the Canadian Conference on VLSI*, pages 8.3.1–8.3.8, 1990.

VITA

Surname Kontopidi Given Name Evaggelia
Place of Birth Athens, Greece

Educational Institutions Attended

University of Victoria, B C , Canada	September 1990 to April 1992
University of Patras, Greece	September 1984 to June 1989

Degrees Awarded

Honors in Comp Eng and Informatics University of Patras 1989

Honours and Awards

University of Victoria Research Assistantship	1990-1992
University of Victoria Graduate Teaching Fellowship	1990 and 1991
University of Victoria Graduate Teaching Award	1990 and 1991

Publications

1 E Kontopidi, "Software Maintenance," Thesis for the Honors in Comp Eng and Inform , Univ of Patras, Greece, June 1989

PARTIAL COPYRIGHT LICENSE

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

Title of Thesis: The Partitioning Behavior of Linear Finite State Machines and VLSI Applications

Author: [redacted]
(Signature)

EVAGGELIA KONTOPIDI
(Name in Block Letters)

April 7, 1992
(Date)