The Concatenation and Partitioning of Linear Finite State Machines [1]

X. Sun
Dept. of Electrical Engineering
University of Alberta
Edmonton, AB T6G 2G7

E. Kontopidi M. Serra J. Muzio
Dept. of Comp. Science
University of Victoria
Victoria, BC V8W 3P6

Abstract

This paper discusses the behaviour of linear finite state machines when partitioned into a number of smaller machines or concatenated into longer ones. We examine linear feedback shift registers (LFSRs) and linear cellular automata registers (LCARs) used in built-in self-test, cryptography and coding theory. We examine the concatenating and/or partitioning of machines which have a maximal length cycle structure to obtain longer or smaller ones maintaining this property. We show that LCARs have better overall behaviour than LFSRs. We introduce some minimum one-cell modifications which improve the number of concatenated or partitioned machines and maintain a maximal length cycle. The hardware cost is discussed.

1 Introduction

In this paper we discuss the behaviour of Linear Finite State Machines when concatenated to themselves or to other machines to yield longer ones, or when partitioned into smaller machines. We examine the most common implementation of general Linear Finite State Machines (LFSMs), namely Linear Feedback Shift Registers (LFSRs) and Linear Cellular Automata Registers (LCARs). Our primary interest is in partitioning machines with maximal length cycles into a number of smaller machines which still maintain the maximal length cycle property. Conversely, we examine the concatenation of machines with maximal length cycles to themselves or to other machines with maximal length cycles, aiming to obtain arbitrarily longer machines which still maintain the maximal length cycle property.

The initial motivation of this research comes from the application of LFSRs and LCARs in the testing of digital circuits, and in particular in built-in self-test (BIST) and self-checking. In this cases, LFSMs are used either as sources of pseudo-random pattern generators to supply test

[1]This work was supported by research grants from the Natural Sciences and Engineering Research Council of Canada and an Equipment Loan Grant from the Canadian Microelectronics Corporation.

vectors to the unit being tested, or as signature analyzers, to compact and evaluate the responses to the test vectors. Different portions of circuits may require different machine lengths and LFSM structures; similarly for board testing, where the boundary scan approach may give dynamic access to each circuit separately. The examination of concatenation and partition of LFSMs allows greater flexibility to a designer, where resources can be dynamically reconfigured at different lengths, still maintaining essential properties, with resulting economy in testing hardware. For example, the scan chain used in boundary scan might be easily reconfigured into a number of smaller registers, each producing maximal length cycles. LFSMs are widely used in other applications, where their concatenation and partition properties can be valuable. As examples, they are used in cryptography for linear encryption and in coding theory for the generation of cyclic codes.

The goal is to provide theoretical and practical information about concatenation and partition of LFSRs and LCARs. We give the theoretical background and we study systematically the behaviour of machines. Up to length 16, we examine *all* possible partitions into two non degenerate submachines; we examine *all* concatenations of machines up to length 64, plus concatenations of *low-cost* machines up to length 256. Tables of results and statistical evaluations are included. Moreover, in cases where the partitioning or concatenation does not lead to maximal length machines, we provide simple one-cell modifications which increase the percentage of machines of the desired type. The study shows that LCARs have better partitioning and concatenation behaviour than LFSRs, and the minimum modifications suggested to the registers are of the same cost to both. The extra hardware required for partitioning, concatenation and possible modifications is estimated and experiments are given.

Much is known about linear finite state machines in general; less has been researched about their reconfiguration properties. The original work for LFSRs originates with Elspas [6] and some more practical implementations on a limited scale can be found in [2]. Initial results on partitioning can also be found in [12] and practical applications of concatenation to BIST is presented in [27, 28].

Section 2 provides the background for LFSMs, their implementations and the representations which are useful in different contexts. Section 3 states the general definitions for partitioning and concatenation, while section 4 and 5 provide the results for each respectively. In section 6 we discuss the possible minimum modifications which can be useful to obtain maximal length cycle machines. In section 7 we present design guidelines and their relative estimated cost in hardware, while section 8 summarizes some recent applications of concatenation and partition.

2 Background

In this section we give some theoretical background of linear finite state machines, their algebraic context, and show their different representations.

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 [26, page 208]. 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*. 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 all v_1 and v_2 in V

$$f(c_1 v_1 + c_2 v_2) = c_1 f(v_1) + c_2 f(v_2) \ [26, page 297].$$

For our purposes, the field is binary and the + operation is modulo-2 (XOR) addition. The linear transformation describes the behaviour of a corresponding linear finite state machine.

Thus a machine M is a *linear finite state machine* (denoted as LFSM) if [26]

(a) *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 the spaces be n, p and r, respectively); and*

(b) *for the state vector s_i, input vector u_i, and output vector y_i at time i, the next-state s_i^+ and the output of M are of the forms*

$$\begin{aligned} M(s_i, u_i) &= s_i^+ = A \cdot s_i + B \cdot u_i \\ \delta(s_i) &= y_i = C \cdot s_i \end{aligned}$$

where A, B, and C, 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 the binary field, the + operation is addition modulo 2 (XOR) and 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* and u_i is absent from the equations.

A basic result in linear algebra is that a linear transformation in a vector space can be represented by a matrix and that, in turn, such a transition matrix represents a linear transformation. Thus we can use matrices to represent a linear finite state machine and analyze it.

If two matrices A and B are square matrices for which there exists an invertible matrix P such that $B = P^{-1}AP$, then A is said to be *similar* to B [26, page 306]. Moreover, two matrices A and B represent the same linear operator T if and only if they are similar to each other [14,

page 155]. Of importance here is the result that if two 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 [26, page 307].

From these results, 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. Since the matrix can represent a linear finite state machine, the theorem give us the freedom to select the machine of least cost in an equivalence class, knowing its state graph has the same cycle structure as any similar state transition matrix.

The *characteristic polynomial* of a square matrix A is defined as $\Delta_A(\lambda) = det(\lambda I - A)$ [26, page 310]. Such polynomial also gives a direct link to the linear machine corresponding to the matrix A (see below). A polynomial $p(X)$ of degree n which is not divisible by any polynomial of degree k, where $0 < k < n$, is called *irreducible* [22, page 148]. An irreducible polynomial of degree n over a binary field is *primitive* and if, and only if, it divides $X^m - 1$ for no m less than $2^k - 1$ [22, page 161]. The important consequence of the primitivity of a characteristic polynomial is that the corresponding linear finite state machine in its autonomous operation traverses all possible $2^n - 1$ non-zero states before returning to its initial configuration. This property is of importance in some applications such as testing.

2.1 Linear Feedback Shift Registers

A *Linear Feedback Shift Register* (LFSR) [26] is a finite state machine defined as a collection of storage elements and XOR gates which perform addition modulo 2, chained together and controlled by a synchronous clock. An example of a four-stage shift register is shown in figure 1, where the symbol $\oplus$ represents XOR gates, and each box represents a memory cell. 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 1 are:

$$\begin{array}{lcllcl} s_0^+ & = & s_3 & \qquad s_2^+ & = & s_1 \\ s_1^+ & = & s_0 \oplus s_3 & \qquad s_3^+ & = & s_2 \end{array}$$

where s_i represents the present state of cell i, and s_i^+ represents its next state.

The corresponding state transition matrix follows from the set of equations such that:

$$\begin{pmatrix} s_0^+ \\ s_1^+ \\ s_2^+ \\ s_3^+ \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \times \begin{pmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{pmatrix}$$

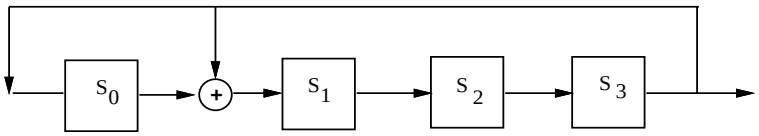


Figure 1: Type 1 LFSR.

The generating function $f(x)$, corresponding to a LFSR, is called the *characteristic polynomial*. This is also the characteristic polynomial of the transition matrix A. The LFSR in figure 1 has $f(x) = x^4 + x + 1$ as characteristic polynomial. A nonzero polynomial coefficient implies that a connection exists in the machine implementation, while a zero polynomial coefficient implies that no connection exists. An autonomous LFSR of n cells is cyclic in the sense that, when clocked repeatedly starting from a nonzero state, it traverses a fixed sequence of at most $2^n - 1$ different states (the successor of the all zero state is itself). If all $2^n - 1$ nonzero states can be generated within the same cycle, it is called a *maximum-length sequence* . The characteristic polynomial associated with a maximum length linear finite state machine is primitive [1, page 77].

If it is desirable to extend the period of a sequence from $2^n - 1$ to 2^n, nonlinear feedback functions are required [1, page 74]. Sequences generated by linear finite state machines are called *pseudorandom sequences,* since they are periodic and deterministic, but they have many of the properties of random sequences.

2.2 One-Dimensional Linear Cellular Automata Registers

Cellular automata 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, where 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, called its *computation rule*, based on the information received from its nearest neighbors.

For our purposes we are interested in one-dimensional cellular automata with a three site neighbourhood, and *linear* computation rules. There are eight such rules, of which only two, rules 90 and 150 (see below), lead to non trivial machines. A Linear Cellular Automata Register (LCAR)

is such a machine, and it is a *hybrid* LCAR if it uses both rules 90 and 150 (it can be proved that any LCAR which yields a maximal length sequence must be hybrid [25]).

Figure 2 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[2].

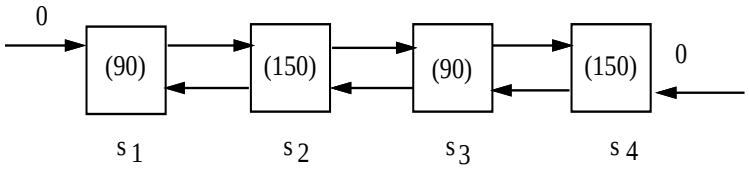


Figure 2: A linear cellular automata register.

The computation rules 90 and 150 are defined as follows:

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

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

According to rule 90, the value of a particular site s_i^+ is 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 s_i. If we use '1' to denote a rule 150 cell and '0' to denote a rule 90 cell, then a LCAR can be represented by a binary vector. For the example of figure 2 the binary vector has the form $< 0101 >$.

For any LCAR, one can also write the set of finite next state equations and 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 2 the next state equations are

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

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 2 produces a maximum length cycle. This is also true of the LFSR in figure 1 which has the same characteristic polynomial.

[2]There are alternative possible boundary conditions that can be also considered (see [23]).

2.3 Different Representations

Both LCARs and LFSRs can be represented by transition matrices, for which characteristic polynomials can be computed. The relationship between LFSRs and LCARs is stated as follows:

Theorem 1 [25] A LCAR and a LFSR with the same irreducible (or primitive) characteristic polynomial are isomorphic, and the corresponding transition matrices are similar.

The consequence is that a LCAR and a LFSR, which are based on the same irreducible or primitive polynomial, have the same behaviour as linear finite state machines up to permutation of the order in which the states appear, and *the cycle structure* of the states [26, page 307] is identical. This is the case for the machines shown in figure 1 and 2.

Example 1 Figure 3 shows a LCAR and a LFSR with their corresponding transition matrices and their characteristic polynomial. The cycle structure is shown in the state transition diagrams. Since this polynomial is irreducible, but not primitive, the states form four separate cycles, where state 0 always goes back to itself.

There are three different representations which are used interchangeably in polynomials and their LFSR implementations: polynomials in a binary field, binary string representations, and the LFSR implementation of polynomials. Each representation provides a convenient expression in a corresponding domain, and can be easily transformed to either of the other two. A polynomial $P(x)$ can be directly mapped into a LFSR implementation, where the zero and non-zero coefficients correspond to feedback taps of the LFSR; and it can also be mapped to a binary string, where the non-zero and zero coefficients correspond to 1's and 0's, respectively (here we always keep the high order coefficients to the left in the binary string representation). The reverse transformations hold as well. In figure 3, these transformations are shown on the left side.

The situation is slightly more complex for LCARs. Given a characteristic polynomial, there are three different algorithms which are implemented to find its corresponding LCAR [3, 24, 25]. The most recent one in [3] applies Euclid's greatest common divisor algorithm to compute the LCAR. It has a polynomial running time, which is sufficiently fast to generate LCARs for polynomials of very large degree.

Conversely, given a LCAR, it is easy to calculate the characteristic polynomial of its transition matrix [25]. The mapping between a LCAR implementation and its binary form is also simple: rule-90 and rule-150 cells correspond to 0's and 1's respectively and form the pattern of the main diagonal of the transition matrix. This is also shown in Figure 3 on the right side. Figure 4 summarizes the transformations above, and visualizes the relations among the different representations.

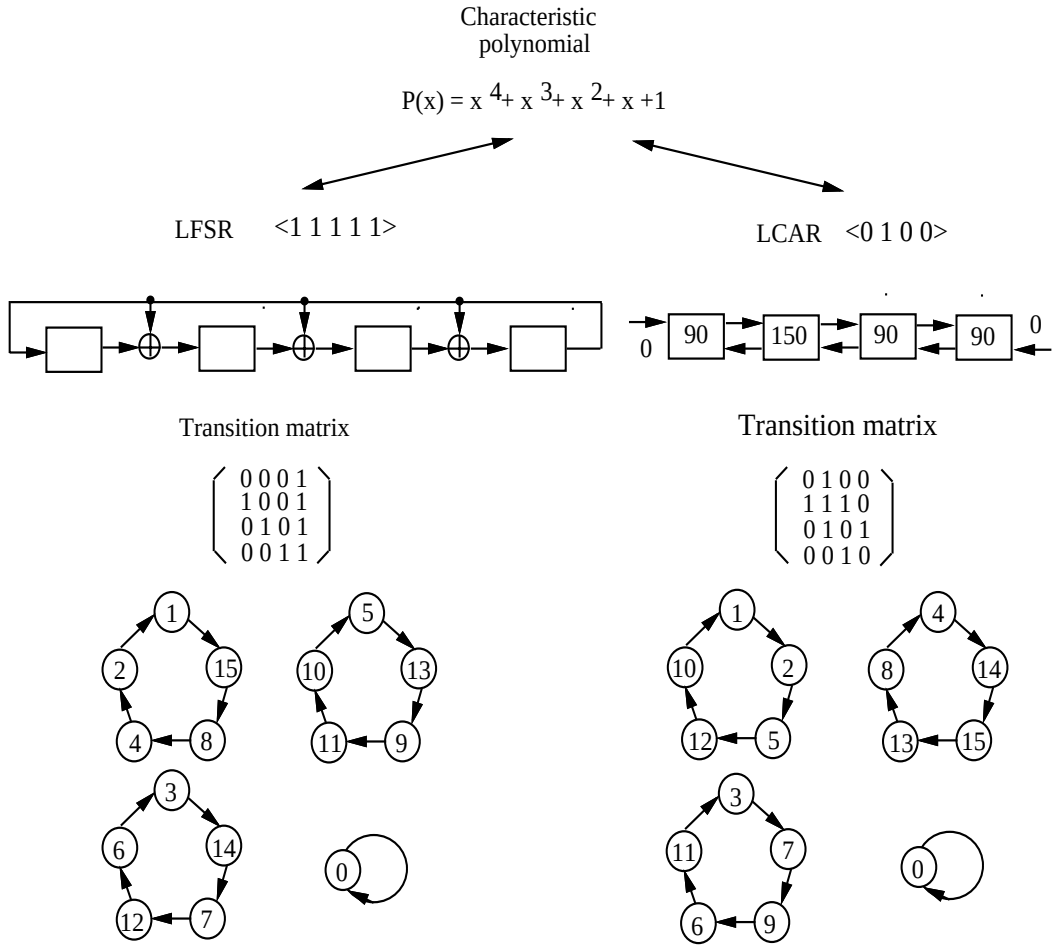


Figure 3: Matrices and cycle graphs.

Both LFSRs and LCARs can be used as pseudo-random pattern generators and signature analyzers. Empirical results showed that a LCAR has better randomness distribution and hence better fault coverage than an LFSR [10, 31, 33]. Further discussion on applications can be found in section 8.

The easiest way to identify a LFSR is by its characteristic polynomial, abbreviated as a bit string (so the LFSR implementing $x^3 + x + 1$ is identified by $< 1011 >$, for example). For a LCAR, the simplest unique identifier is the diagonal of the transition matrix, these bits also indicating whether a cell implements rule 90 or rule 150. For example, in Figure 3, the LCAR implementing $x^4 + x^3 + x^2 + x + 1$ is $< 0100 >$. These representations are used throughout this paper.

Without ambiguity we refer to a LFSR or LCAR which has a corresponding primitive characteristic polynomial as a "primitive LFSR" or "primitive LCAR", and similarly for irreducibility.

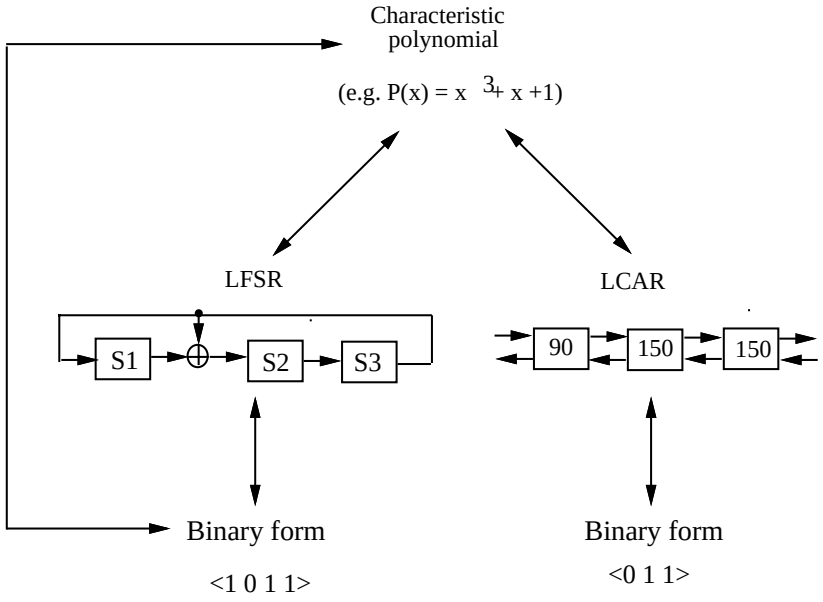


Figure 4: Representations for LFSR and LCAR.

3 Partitions and Concatenations: General Definitions

The main interest lies in finding structures which can be partitioned and/or concatenated and still maintain certain attributes. Bhavsar [2] 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 polynomials. We treat all such machines in a uniform way with a combination of theoretical and experimental results.

Since there is a direct $1:1$ isomorphism between LFSRs and polynomials, Bhavsar [2] could discuss the question of concatenation in terms of polynomials. This is not the case for LCAR, and even for LFSR there are actually two possible concatenation structures to be considered. Consequently, our definitions have to be tailored to match precisely the relevant machine structure.

Definition 1 Let A and B be two LFSRs implementing polynomials $A(X)$ and $B(X)$ of degree r and s respectively. A and B are *concatenated* to realize four distinct LFSRs of degree $r+s$ as follows:

$$\begin{aligned} C_{AB} &= x^s(A(X)+1)+B(X) \\ C_{BA} &= x^r(B(X)+1)+A(X) \\ D_{AB} &= x^sA(X)+B(X) \\ D_{BA} &= x^rB(X)+A(X). \end{aligned}$$

The four possible concatenation structures given above arise because A can be placed either at

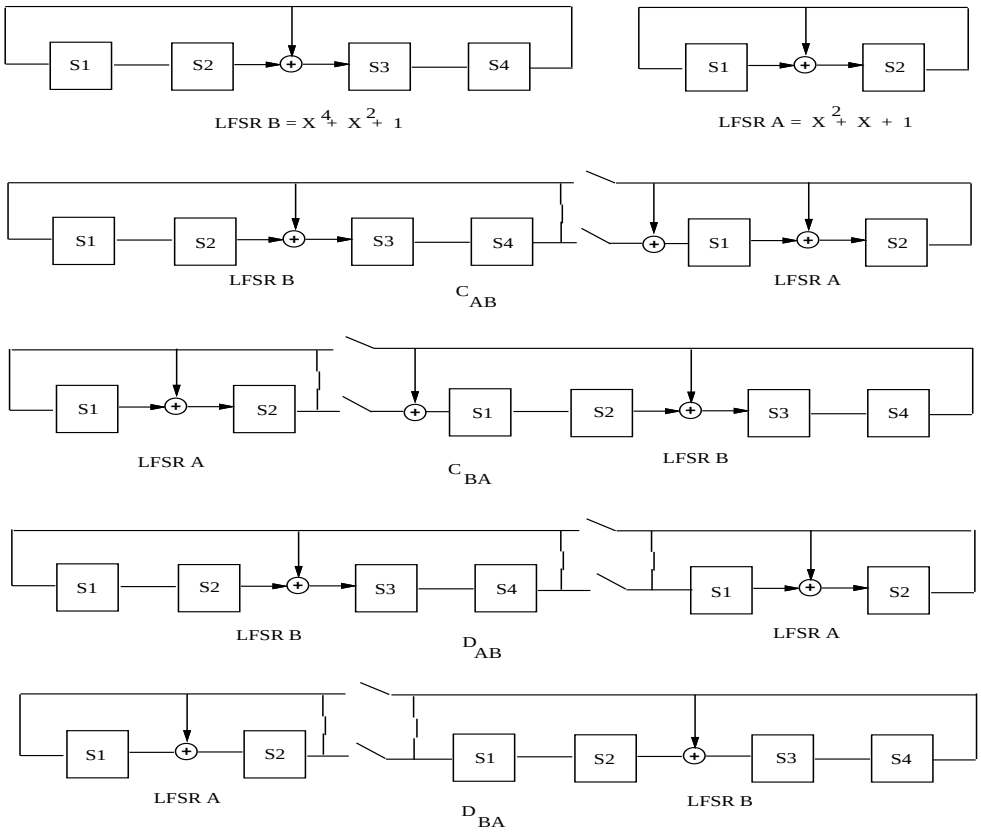


Figure 5: LFSR Concatenation.

the right or left end of the composite machine and the link between the two machines can either include an XOR or exclude it. In practice, the former is the more useful structure. Note that in C_{AB}, D_{AB}, the machine A is physically positioned at the right hand end of the composite machine (the "high order" end).

Example 2 Consider the LFSRs implementing the polynomials $A = x^2+x+1$ and $B = x^4 + x^2 + 1$. The polynomial D_{AB} is formed as follows:

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

This is illustrated in Figure 5.

Definition 2 Let $A =< a_1, \ldots, a_r >$ and $B =< b_1, \ldots, b_s >$ be two LCARs of length r and s respectively, and let $A' =< a_r, \ldots, a_1 >$ and $B' =< b_s, \ldots, b_1 >$ be their mirror images. A, A', B and B' are *concatenated* to form LCARs of length $r + s$ as follows:

$$C_{AB} = < a_1, \ldots, a_r, b_1, \ldots, b_s >$$

$$
\begin{aligned}
C_{AB'} &= < a_1, \ldots, a_r, b_s, \ldots, b_1 > \\
C_{A'B} &= < a_r, \ldots, a_1, b_1, \ldots, b_s > \\
C_{A'B'} &= < a_r, \ldots, a_1, b_s, \ldots, b_1 > \\
C_{BA} &= < b_1, \ldots, b_s, a_1, \ldots, a_r > \\
C_{BA'} &= < b_1, \ldots, b_s, a_r, \ldots, a_1 > \\
C_{B'A} &= < b_s, \ldots, b_1, a_1, \ldots, a_r > \\
C_{B'A'} &= < b_s, \ldots, b_1, a_r, \ldots, a_1 >
\end{aligned}
$$

It is clear than a LCAR $< a_1, \ldots, a_r >$ and its mirror image $< a_r, \ldots, a_1 >$ represent machines with identical cycle structures, so the eight possible concatenations listed above can be reduced to a maximum of four distinct machines with distinct cycle structures (each of the machines in the pairs $(C_{AB}, C_{B'A'})$, $(C_{AB'}, C_{BA'})$, $(C_{A'B}, C_{B'A})$, $(C_{A'B'}, C_{BA})$ has the same cycle structure). Note that the concatenation of two LCARs is a little simpler than that for LFSRs because the latter may have to incorporate an extra XOR at the join.

Example 3 Consider the LCARs $A =< 110 >$ and $B =< 0100 >$ *with characteristic polynomials* $x^3 + x + 1$ and $x^4 + x^3 + x^2 + x + 1$ respectively. Figure 6 illustrates the possibilities.

Definition 3 If the LFSM formed by concatenation implements a primitive characteristic polynomial, it is a *primitive concatenation.*

Definition 4 The concatenation of single LFSM of degree s with itself n times $(n > 1)$ to realize a machine of length $n \times s$ is called *self-concatenation.* Otherwise a concatenation is *non-self-concatenation.*

Definition 5 A LFSM formed by the concatenation of two LFSMs, A and B, can also be *partitioned* into the two machines, A and B.

Definition 6 The partition of a LFSM C, of degree t, into two irreducible LFSMs A and B, of length r and s respectively, such that $r + s = t$, is called an *irreducible partitioning.* Otherwise, it is called a *reducible partitioning.*

Definition 7 The partition of a LFSM C, of degree t, into two primitive LFSMs A and B, of degree r and s respectively, such that $r + s = t$, is called a *primitive partitioning.* Otherwise, it is called a *nonprimitive partitioning.*

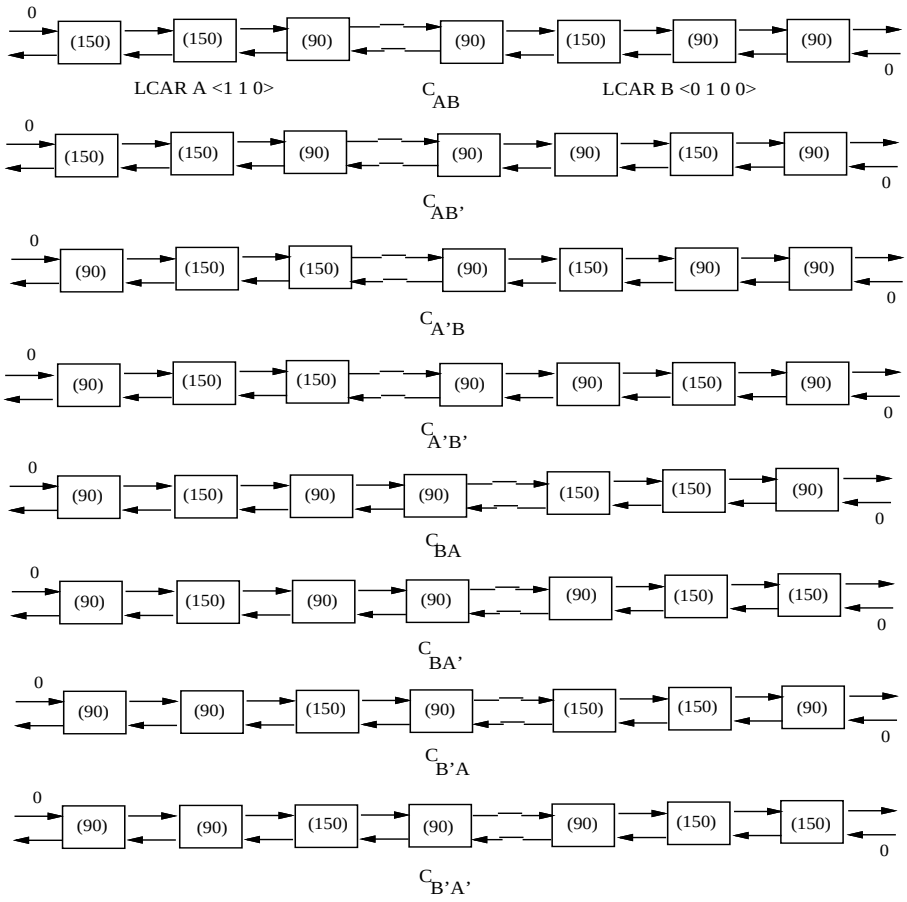


Figure 6: LCAR Concatenation.

It should be noted that primitive concatenation implies primitivity of the newly-formed polynomial, while primitive partitioning means that both polynomials obtained from the partitioning are primitive. The logic behind these seemingly inconsistent definitions is that in both cases, concatenation and partitioning, the attribute of primitivity is only related to the result of the operations (primitivity of the resulting C in concatenation, and the resulting A and B in partitioning), while the inputs (the polynomials A and B before concatenation, and polynomial C before partitioning) are not necessarily primitive. All definitions above apply equally to both linear finite state machines, LFSRs and LCARs.

Definition 8 If a LFSM of length t is partitioned into two machines of lengths r and s respectively, such that $r + s = t$, $r = 1$ or $s = 1$, then it is called a *degenerate partitioning.*

Definition 9 If a LFSM of length t is partitioned into two machines of lengths r and s respectively, such that $r + s = t$, $1 < r, s < t - 1$, then it is called a *proper partitioning.*

Example 4 The degenerate partitions of length 6 finite state machines are:

$$\square \quad \| \quad \square\ \square\ \square\ \square\ \square$$

$$\square\ \square\ \square\ \square\ \square \quad \| \quad \square$$

where "$\|$" indicates the partition point. The proper partitionings are:

$$\square\ \square \quad \| \quad \square\ \square\ \square\ \square$$

$$\square\ \square\ \square \quad \| \quad \square\ \square\ \square$$

$$\square\ \square\ \square\ \square \quad \| \quad \square\ \square.$$

Hence, a finite state machine of length n has $(n - 1)$ possible partitionings, two degenerate partitionings, and $(n - 3)$ proper partitionings.

Example 5 *The LFSR based on the characteristic polynomial* $x^{16} + x^{14} + x^{11} + x^{10} + x^8 + x^6 + x^3 + x^2 + 1$ *can be partitioned into two length eight primitive machines, both of which implement the same polynomial* $x^8 + x^6 + x^3 + x^2 + 1$.

Example 6 *The LCAR based on the characteristic polynomial* $x^{16} + x^{12} + x^{11} + x^8 + x^7 + x^6 + x^5 + x^2 + 1$ *and represented by the binary vector* $< 0100010101010100 >$ *can be partitioned into two length eight primitive machines, which implement the polynomials* $x^8 + x^7 + x^5 + x^3 + 1$ *and* $x^8 + x^7 + x^3 + x^2 + 1$.

4 Concatenation

There are two distinct approaches to studying the concatenation properties of linear finite state machines: find the mathematical basis of concatenation or use a simulation method. The concatenation of linear machines, both for LFSRs and LCARs, implies a composition of their respective transition matrices. For example, in the concatenation of two LCARs, with A_1 and A_2 as transition matrices, the direct sum is shown by the two transition matrices on the diagonal of the composite matrix B, as shown below. The primitivity and irreducibility of the characteristic polynomial of the resulting matrix B can not be deduced by the attributes of the characteristic polynomial of the submatrices. Thus, it is investigated by simulation.

$$B = \begin{pmatrix} A_1 & 0 \\ 0 & A_2 \end{pmatrix}$$

Concatenation can be examined from three different perspectives: the method of concatenation, the attributes of the resulting machines, and the attributes of the participating machines.

There are four possible cases for the attributes of the participating linear machines which are considered:

(1) Primitive-Primitive concatenation (PP),

(2) Primitive-Nonprimitive concatenation (PN),

(3) Nonprimitive-Primitive concatenation (NP), and

(4) Nonprimitive-Nonprimitive concatenation (NN).

The concatenation operations given above can be used repeatedly to combine any number of machines. Obviously, there are many choices. Figure 7 summarizes the taxonomy of concatenation. We assume that the root is level 1, then levels 2, 3 and 4 in turn represent the three perspectives of the classification: method of concatenation (self or non-self), attributes of resulting machine (primitive ot not), and attributes of participating machines (primitive ot not), respectively.

Table 1 shows *the primitive self-concatenations* of primitive machines of degree 3 to degree 6, forming machines up to degree 64. Both the initial and the resulting machines are primitive.

A subset of this table can be found originally in [2]. Table 1 includes reciprocal polynomials[3]of the polynomials. Tables up to degree 256 can be found in the Appendix. When concatenating

[3]The reciprocal polynomial is defined as $P(1/x)$ and its binary representation is the reverse image of the original $P(x)$.

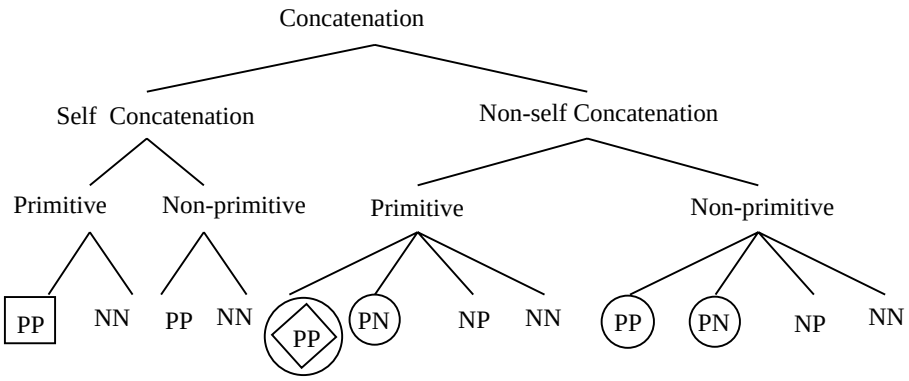


Figure 7: The taxonomy of concatenation.

Degree	Polynomial	LCAR	LFSR Concatenation	LCAR Concatenation
2	111	01	-	2,3,5,6,9,11,14
3	1011	011	2,3,9	16
	1101	001	2,3,9	4
4	10011	0101	-	3,7,13,15
	11001	1011	-	17
5	100101	01111	3,12	-
	101001	00110	3,12	3,9
	110111	00001	5	-
	111011	00111	5	6,7
	101111	00011	-	6,7
	111101	10011	-	3,8,9
6	1000011	000110	3,5	10
	1100001	010110	3,5	2,6,8
	1100111	100101	2	2,4,6
	1110011	000001	2	3
	1011011	011101	-	9
	1101101	010101	-	2,3,10

Table 1: Self-concatenation of degree 3 to 6 polynomials.

polynomials, and their respective LFSRs, there are implications for the implementation, as the feedback taps must be adjusted appropriately (see section 7). Note that, for LFSRs, self concatenation without an XOR at the join always leads to a reducible machine [6].

The study in LCAR concatenation has yielded a variety of tables, available from various technical reports and summarized below to correspond to the concatenation hierarchy shown in Figure 7.

(1) Primitive Self Concatenation.

This type of concatenation corresponds to the square box of Figure 7. Tables are complete for lengths from 2 to 16. We list *all* primitive LCARs forming longer primitive LCARs by self-concatenation up to length 64 [29]. As examples, the primitive LCAR $< 1010 >$ self-concatenates 3, 7, 13 and 15 times to form primitive LCARs of length 12, 28, 52 and 60 respectively; the length 8 primitive LCARs, $< 11101110 >$ and $< 11110111 >$, self-concatenate twice and four times respectively to give length 16 and 32 primitives.

(2) Primitive and Irreducible Non-self Concatenation: low cost.

This type of concatenation corresponds to a subset of the 'PP' leaves in the self-concatenation left side of Figure 7. Here:

i) the maximum length of the concatenated LCAR is 256;

ii) the resulting LCAR can be either primitive or irreducible;

iii) the initial LCAR are primitive and low cost.

A low cost primitive LCAR has a minimal number of cells with rule 150, which are the most expensive in implementation (see also section 7). A complete list of minimal cost LCAR up to degree 300 can be found in [4]. The low cost primitive LCAR of length 2 to 8, 10 and 12 have a maximum of 3 rules 150. For length 16, the minimal cost is 4 (none exist with 3). As an example, a subset of the tables in this category are listed in the Appendix, as they are the most useful in applications.

(3) Primitive Non-self Concatenation.

Due to the very large number of possible concatenations of each length, the tables generated are not exhaustive [29]. The circled diamond in Figure 7 represents the concatenations in this category.

(4) Non-self primitive/non-primitive concatenation.

Tables here are not exhaustive and correspond to the circles in Figure 7.

5 Partitioning

The partitioning behaviour of linear finite state machines, for all degree 2 to 16 primitive and irreducible polynomials, are examined, with extrapolation for higher degrees. The exhaustive search is applied to primitive and irreducible polynomials with primitive partitioning and/or irreducible partitioning respectively. It implies that both the original and the resulting machines are also primitive and irreducible. The study is restricted to proper partitionings. The partitioning behaviour is measured using two parameters:

(1) *ATLOP:* the number of degree n polynomials which each have AT Least One Partitioning with the desired property.

(2) *PEPP:* the PErcentage of the total number of Partitionings which have the desired property.

For example, for primitive polynomials of degree n, a partition with the desired property is a primitive partitioning, and hence PEPP is defined as

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

where P_{prim} is the total number of (proper) primitive partitionings, and $prim(n)$ is the total number of primitive polynomials of degree n. The factor $(n-3)$ is used to subtract degenerate cases.

5.1 Partitioning of LFSRs

The partitioning behavior of LFSRs of length up to 16 which implement irreducible and primitive polynomials is given in table 2. Table 2 shows that 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 results yield to similar observations for primitive polynomials. Further discussion on improvements can be found below.

5.2 Partitioning of LCARs

The partitioning behavior of LCARs which implement irreducible and primitive polynomials is reported in table 3. The percentage of irreducible partitions ($PEPP$) varies between 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

Degree	*Irreducible*			*Primitive*		
	PEPP	*Total Number*	*ATLOP*	*PEPP*	*Total Number*	*ATLOP*
4	33.33	3	1	0	2	0
5	33.33	6	4	33.33	6	4
6	14.81	9	4	22.22	6	4
7	19.44	18	10	16.66	18	10
8	18.00	30	22	12.50	16	10
9	11.30	56	32	9.72	48	22
10	11.39	99	55	8.57	60	26
11	9.67	186	112	6.81	176	80
12	9.02	335	206	6.32	144	70
13	7.87	630	356	5.42	630	276
14	7.46	1161	731	4.93	756	336
15	6.53	2182	1262	3.94	1800	704
16	6.21	4080	2393	4.15	2048	904

Table 2: The partitioning behavior of LFSRs - irreducible and primitive.

Degree	*Irreducible*			*Primitive*		
	PEPP	*Total Number*	*ATLOP*	*PEPP*	*Total Number*	*ATLOP*
4	33.33	3	1	50.00	2	1
5	16.66	6	2	16.66	6	2
6	22.22	9	5	27.77	6	4
7	22.22	18	12	18.05	18	11
8	17.33	30	21	12.50	16	8
9	14.88	56	40	12.15	48	27
10	14.57	99	79	11.42	60	40
11	12.36	186	132	8.59	176	91
12	11.40	335	233	7.40	144	78
13	9.68	630	436	6.38	630	311
14	9.06	1161	801	5.73	756	378
15	8.60	2182	1504	5.53	1800	910
16	7.39	4080	2735	4.81	2048	1002

Table 3: The partitioning behavior of LCARs - irreducible and primitive.

again vary, with the lowest value of $ATLOP$ 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 3. In general, both for LFSRs and LCARs there is no value of n for which there is at least one proper partition for each machine. However, LCARs have slightly better performance than LFSRs.

5.3 Discussion

The results using $PEPP$ and $ATLOP$ provide insight into the partitioning behavior of LFSRs and LCARs and give rise to 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.

The evidence in support of the first hypotheses is clearer from the comparative graphs for irreducible and primitive machines which are shown in the lower portion of figures 8 and 9 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.

The values of *PEPP* are more consistent, i.e., they decrease as the machine length increases. The results imply that *PEPP* is a 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. If the results for odd and even degree polynomials are treated separately, *PEPP* with $n > 8$ decreases almost linearly. As a result, even for longer than length 16 machines, one would expect that *PEPP* is likely to decrease as n increases. Considering that *PEPP* for $n = 16$ reaches the value 4.15%, it is likely to be very small for longer machines. For the application of built-in self-test (BIST) for digital circuits, it would be more flexible 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. This leads to question whether we can achieve better performance of LFSRs and LCARs, and whether we can improve the partitioning behavior of these machines just by allowing minimum modifications in the design. Some answers are provided below.

6 Improvements and Comparisons

Examining the partitioning and concatenation behavior, it appears that LCARs have slightly better performance than LFSRs. However, for both machines the percentage of irreducible or primitive partitions is small. With regards to concatenation, there are many possible self-concatenations of the primitive polynomials. However, from degree 2 to 8, there are only 13 primitive polynomials leading to a total of 23 possible primitive self-concatenations to form primitive LFSRs up to degree 64 [29]. Adding the corresponding reciprocal polynomials, we double the two numbers above, and have 26 primitive polynomials with primitive self-concatenations and 46 primitive self-concatenations from the 26 initial ones. On the other hand, there exist 58 initial primitive LCARs with primitive self-concatenation, from length 2 to 8, and 110 possible primitive self-concatenations from the initial LCARs. The mirror image (corresponding to the reciprocal of a polynomial) of a LCAR is considered as a distinct LCAR. In spite of the fact that there are equal number of initial primitive polynomials and primitive LCARs at any degree, LCARs provide a richer set of self-concatenable machines that maintain primitivity.

This led us to investigate ways to improve the partitioning (or concatenation) behavior. We demonstrate that better performance for both LFSRs and LCARs can be accomplished by allowing minimum hardware modifications. Experimental results are presented to substantiate this claim.

We define *one modification* to a LFSR as the introduction or the elimination of a nonzero term in its characteristic polynomial. Alternatively, we define *one modification* in a LCAR as the reconfiguration of a rule 90 cell to a rule 150 cell or vice versa. All irreducible and primitive machines up to length 16 have been investigated. For each length ($n = 4, \cdots, 16$) 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 since we restrict ourself to only *one* modification. Obviously, one could allow more than one modifications to minimize area overhead.

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.

Example 7 *The LFSR with characteristic polynomial* $x^{16} + x^{10} + x^9 + x^7 + x^6 + x + 1$ *can be partitioned into two length eight machines with characteristic polynomials* $x^8 + x^7 + x^6 + x + 1$ *and* $x^8 + x^2 + x + 1$*, respectively. Only the first bit-slice is primitive. The introduction of the term* x^7 *in the non primitive partition (a change from 0 to 1) results in the polynomial* $x^8 + x^7 + x^2 + x + 1$*, which is primitive.*

Example 8 *The LFSR with characteristic polynomial* $x^{16} + x^{12} + x^{11} + x^{10} + x^7 + x^6 + x^5 + x^3 + 1$ *can be partitioned into one primitive machine with* $x^8 + x^4 + x^3 + x^2 + 1$ *and one non primitive with* $x^8 + x^7 + x^6 + x^5 + x^3 + 1$*. By eliminating the term* x^6 *(a change from 1 to 0), the primitive machine is formed with* $x^8 + x^7 + x^5 + x^3 + 1$*.*

Example 9 *The LCAR with characteristic polynomial* $x^{16} + x^{14} + x^{12} + x^{11} + x^{10} + x^9 + x^5 + x^4 + x^2 + x + 1$ *is represented by the binary vector* $< 0110000010100000 >$ *and can be partitioned into one primitive machine* $< 01100000 >$ *with characteristic polynomial* $x^8 + x^4 + x^3 + x^2 + 1$ *and one non primitive* $< 10100000 >$ *with* $x^8 + x^5 + x^4 + x^3 + x^2 + x + 1$*. By changing the seventh cell (from the left) from rule 90 to rule 150 we obtain a new primitive bit machine* $< 10100010 >$ *with characteristic polynomial* $x^8 + x^7 + x^5 + x^3 + 1$*.*

Example 10 *The LCAR with characteristic polynomial* $x^{16}+x^{15}+x^{14}+x^{11}+x^{10}+x^6+x^5+x^4+x^3+x+1$ *is represented by the binary vector* $<1000011001100000>$. *By modifying the first cell from rule 150 to rule 90, the LCAR can be partitioned into two primitive machines,* $<00000110>$ *and* $<01100000>$ *with the same characteristic polynomial* $x^8+x^4+x^3+x^2+1$.

We evaluate the performance of these machines using again *PEPP* and *ATLOP*.

The partitioning behavior of irreducible and primitive LFSRs up to degree 16, when minimum modifications are allowed, is shown in tables 4 and 5, respectively, columns 3 and 5.

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

(1) Irreducible and primitive LFSRs exhibit better partitioning performance when minimum modifications are allowed.

(2) 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 $n = 16$).

(3) 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 $n = 15$).

(4) One modification results in larger values for *PEPP*. The percentage of irreducible partitions starts from 100% for $n = 4$ and gradually decreases to 33.52% for $n = 16$. The corresponding value before any modifications, was 6.21%. Similarly for primitive partitions *PEPP* is equal to 100% for $n = 4$ and gradually decreases to 24.50% for $n = 16$. Without any modifications, *PEPP* was 4.15%.

The partitioning performance of irreducible and primitive LCARs up to degree 16 after the introduction of one modification, is reported in tables 4 and 5, respectively, columns 4 and 6. Observations which can be derived from the tables are:

(1) The better partitioning behavior of LCARs with the introduction of one modification is evident from the values of *PEPP* and *ATLOP* shown in tables 4 and 5.

(2) 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	*Number of Irred. Polynomials*	*PEPP LFSR*	*PEPP LCAR*	*ATLOP LFSR*	*ATLOP LFSR*
4	3	100.00	100.00	3	3
5	6	66.66	83.33	6	6
6	9	74.07	81.48	9	9
7	18	61.11	66.66	18	18
8	30	54.00	56.00	30	30
9	56	50.59	60.71	56	56
10	99	47.47	55.84	98	99
11	186	44.35	55.64	186	186
12	335	41.45	50.34	330	335
13	630	38.98	50.12	628	630
14	1161	37.34	47.57	1154	1160
15	2182	35.17	45.91	2176	2182
16	4080	33.52	44.08	4062	4080

Table 4: The partitioning behavior of LFSRs and LCARs (irreducible) with one modification.

(3) 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 $n = 16$).

(4) *PEPP* exhibits higher values, as it equals 100% for $n = 4$ and it gradually decreases to 44.08% for $n = 16$. Similarly, for primitive partitions, *PEPP* is equal to 100% for $n = 4$, decreasing to 34.34% for $n = 16$. Without any modifications, the corresponding values of *PEPP* for $n = 16$, are 7.39% and 4.81% for irreducible and primitive LCARs, respectively.

The better partitioning behavior with the introduction of one change is evident from the values of *PEPP* and *ATLOP* shown in these tables. This is shown further in the comparative graphs of figures 8 and 9. 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 8 and 9 allow us to support the following arguments concerning the partitioning performance and behavior of LFSRs and LCARs:

(1) LFSRs and LCARs demonstrate significantly better performance when minimum modifications are allowed. In practice, this behavior promises a great economy in hardware

Degree	*Number of Irred. Polynomials*	*PEPP LFSR*	*PEPP LCAR*	*ATLOP LFSR*	*ATLOP LFSR*
4	2	100	100	2	2
5	6	66.66	83.33	6	6
6	6	83.33	72.22	6	6
7	18	51.38	59.72	18	18
8	16	45.00	52.5	16	16
9	48	38.54	54.16	46	48
10	60	37.61	44.76	58	60
11	176	33.94	44.60	171	175
12	144	31.55	40.12	143	142
13	630	28.33	39.84	605	628
14	756	26.29	37.01	735	752
15	1800	24.45	35.16	1733	1787
16	2048	24.50	34.34	2011	2043

Table 5: The partitioning behavior of LFSRs and LCARs (primitive) with one modification.

since it allows the use of the same machine for more than one purposes.

(2) 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 a 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.

(3) 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 partitioning behavior. This argument is more evident if separate plots are made where even and odd degree polynomials are treated separately. In these cases, *PEPP* decreases linearly with the increase in the machine length.

In general, the experimental results 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 performance, and hence more implementation options, than

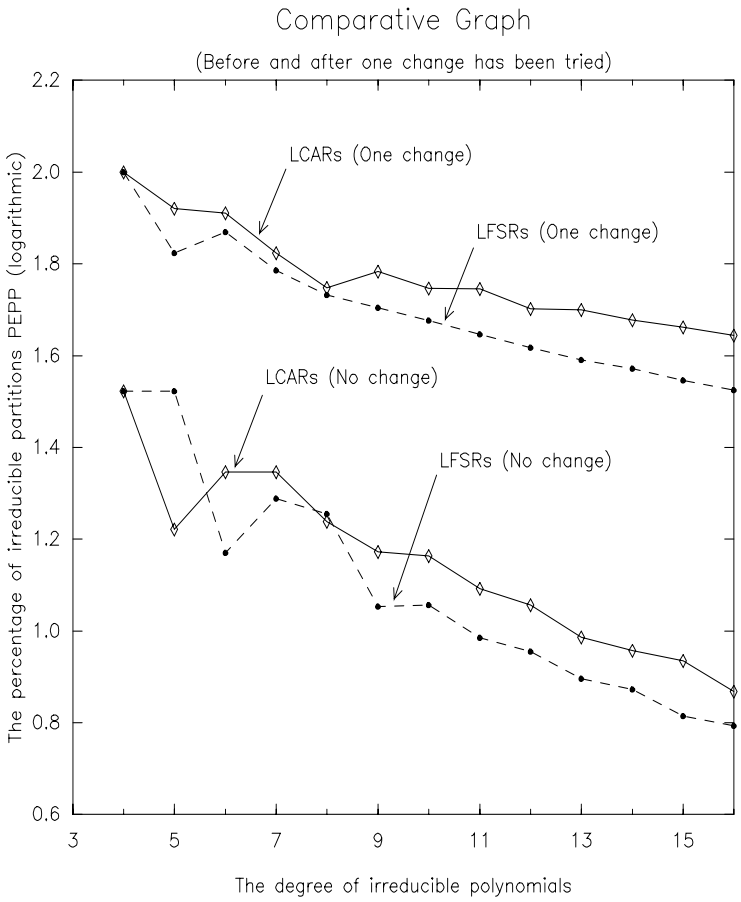


Figure 8: LFSRs - LCARs: The improved percentage of irreducible partitions.

LFSRs. The comparative graphs for LFSRs and LCARs after one modification are shown in the top portion of figures 8 and 9.

7 Design Principles

LFSR and LCAR implementations are of importance in a number of applications including digital testing. Modifications to original designs of the linear machines are required to support the dynamic reconfiguration, through the introduction of additional hardware. We present here various implementation structures of LFSRs/LCARs and discuss their silicon costs.

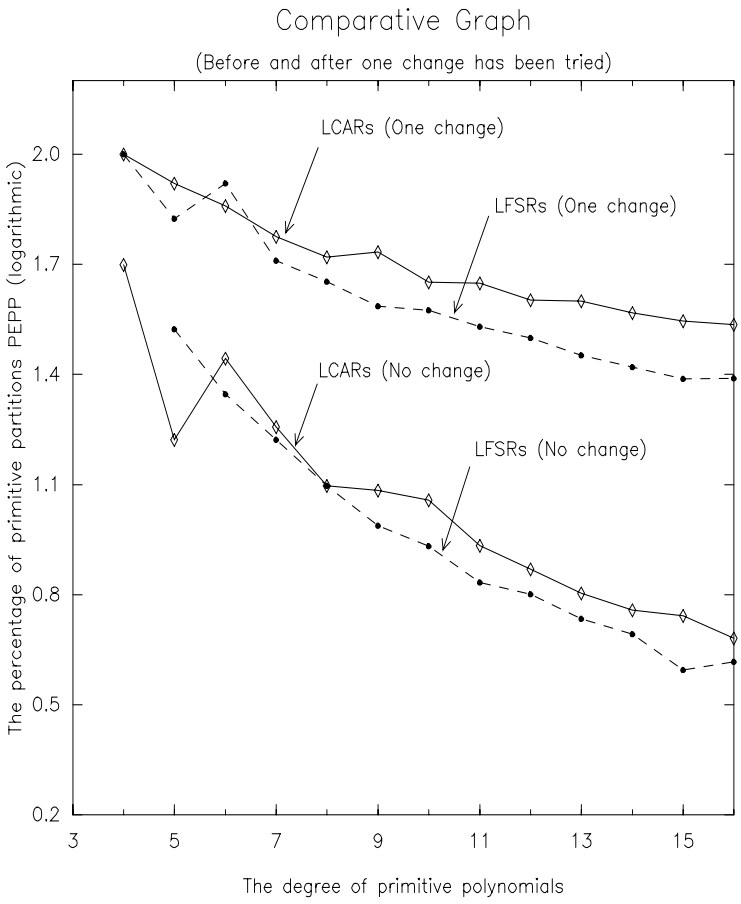


Figure 9: LFSRs - LCARs: The improved percentage of primitive partitions.

Degree	Minimum-cost primitive polynomial	Minimum-cost primitive LCAR
2	111	10
3	1011	001
4	10011	1010
5	100011	10000
6	1000011	100000
7	10000011	0010000
8	101100011	01100000
9	1000010001	100000000
10	10000001001	0100001000
11	100000000101	00001000000
12	1000001001101	001000100000
13	10000000011011	0000100000000
14	101100000000011	10000000000000
15	1000000000000011	000000100000000
16	10000000000001101	0000101000000000

Table 6: Minimum cost LFSRs and LCARs

7.1 Low Cost LFSRs and LCARs

A n-stage LFSR implementing a degree n polynomial requires $m-2$ XOR gates, corresponding to m non-zero coefficients, where $m <= n+1$. A *n-cell* LCAR requires one XOR gate for each site, with these being a two- or three-input gate depending whether the rule is 90 or 150. The two boundary cells only require a two-input XOR if they use rule 150. LFSRs with minimum m and LCARs with the minimum number of rule 150 cells are desirable to reduce the hardware cost. Table 6 lists a selection of primitive polynomials and LCARs with the minimum number of "1" coefficients from degree 2 to 16, where column 2 is reprinted from [1] and column 3 is from [29, 32]. It can be seen from the table that the lowest cost primitive polynomials are trinomials (having three terms), and pentanomials (having five terms), requiring 1 or 3 XOR gates respectively in their LFSR implementation. There are no primitive trinomials of degree 8 or degree $8n$ for any n [8]. The low cost primitive LCARs all have at most two cells with rule 150.

LCARs in general require more XOR gates than LFSRs. However, it should be noted that routing is not taken into consideration here. In practice, long feedback loops in LFSRs may require the use of larger XOR gates, hierarchical drivers or wider and thicker metal line segments. The cost of the feedback routing grows linearly with the number of stages. The detailed comparison of the hardware cost of the two machine structures can be found in section 7.3.

7.2 Concatenation and Partitioning Circuitry

In this section, we first discuss the cost of concatenation or partitioning circuitry at logic and circuit levels. Then, we estimate the hardware cost of the minimum LFSR/LCAR modification.

7.2.1 Logic Level

Global broadcasting forms an essential part of LFSRs. The feedback connection of a LFSR runs across its length and connects at least the output of the last cell to the input of the first. Given two LFSRs, there are four possible ways to concatenate them, as defined in section 3 and illustrated in Figure 5. Note that these concatenations can be expanded logically to more than two machines. The concatenations D_{AB} and D_{BA} always lead to a longer LFSR implementing a reducible polynomial and, thus, without a maximal length cycle [6].

Since communication in a LCAR implementation is restricted to nearest neighbours, two LCARs can be concatenated directly side by side. This is defined in section 3 and illustrated in Figure 6. Direct concatenations are logically simpler for LCARs and provide more choices.

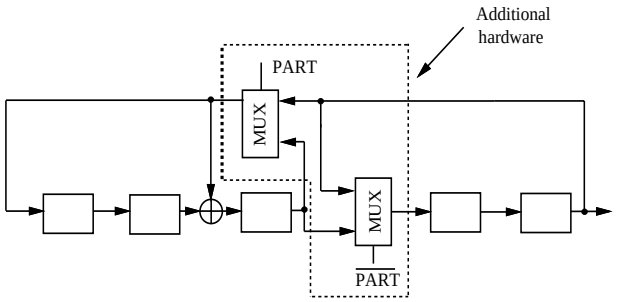
7.2.2 Circuit Level

Figures 10 and 11 show the configuration circuitry required for an LFSR and an LCAR, respectively. When signal $PART$ is high, the two machines are disconnected and work independently; when $PART$ is low, the concatenated machines are constructed. In practice, the concatenation of LFSRs is even more complex. If one concatenates two long LFSRs, the driving capacity of the last stage cell in the newly constructed LFSR could be crucial to the system performance. Solutions to this problem can be either to design a special driver for it, and/or use thick metal line or to allow the system to run at lower speed. Obviously, LCARs do not share this problem.

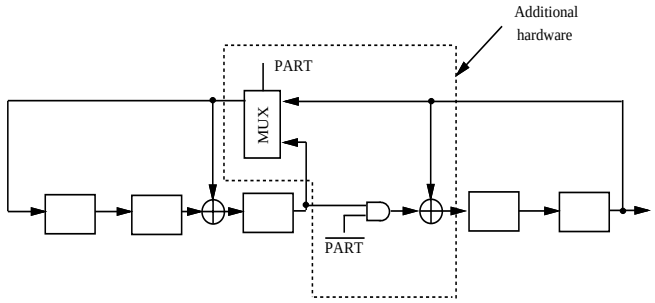
7.2.3 Hardware Required for Modifications

In section 6, we show that the partitioning performance of LFSRs/LCARs can be enhanced by allowing one modification of the original machines. This section is devoted to the estimation of the hardware overhead required for such a modification.

For an LFSR, the hardware required to implement a change from 0 to 1 and a change from 1 to 0, is shown in figures 12 and 13, respectively. Notice that the change from 0 to 1 requires one extra AND gate and one XOR gate. However, the change from 1 to 0 does not require an extra XOR gate. As a result, one would consider the latter modification as preferable in terms of area cost.



(a) Side by side configuration



(b) Configuration with an XOR at the join

Figure 10: Dynamic reconfiguration of a LFSR

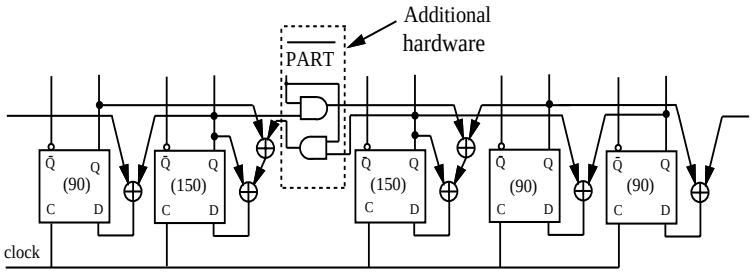


Figure 11: Dynamic reconfiguration of LCARs

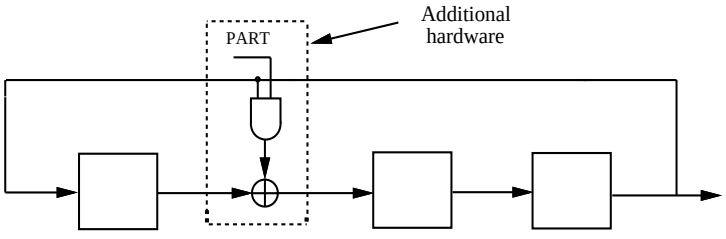


Figure 12: Type 1 LFSR: The linear machine which implements the reducible polynomial x^3+1 is reconfigured, after the modification, to a maximal length machine with characteristic polynomial x^3+x+1; change from 0 to 1.

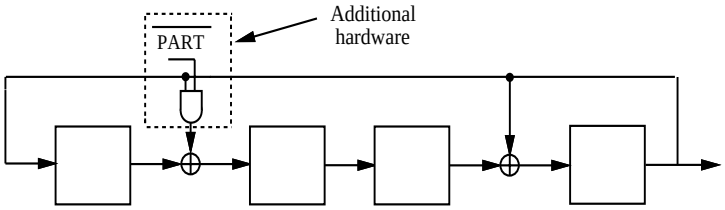


Figure 13: Type 1 LFSR: The linear machine which implements the reducible polynomial $x^4 + x^3 + x + 1$ is reconfigured, after the modification, to a maximal length machine with characteristic polynomial $x^4 + x^3 + 1$; change from 1 to 0.

The proposed hardware design which supports minimum modifications for LCARs is shown in figures 14 and 15. Figure 14 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 15, the reverse change, implies only one extra AND gate, and thus is preferable. Notice that the minimum modifications require the same hardware for both LFSRs and LCARs.

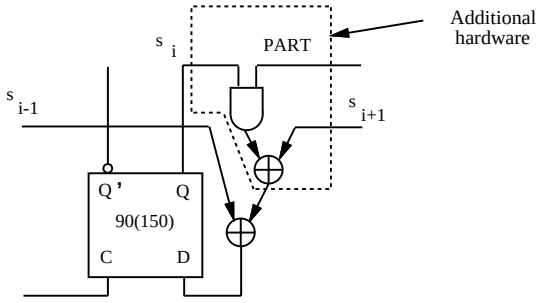


Figure 14: LCARs: A rule 90 cell which is reconfigured to a rule 150 cell after the modification.

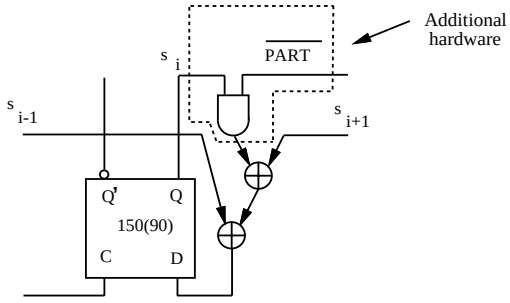


Figure 15: LCARs: A rule 150 cell which is reconfigured to a rule 90 cell after the modification.

7.3 Overall Hardware Overhead: an Experiment

As an experiment, using OASIS[4]for the measures, we consider three different primitive LFSRs and three different primitive LCARs, all of length 16. The goal is to partition them into two length eight primitive machines, with or without modifications. The LFSRs are: < 10100110101001101 >, < 10000011011000011 > and < 10001110011101001 >; the LCARs are < 0100010101010100 >, < 0110000010100000 > and < 1000011001100000 >. The cost of the reconfiguration hardware for the three LFSRs is 5.8%, 21.6% and 18.8% respectively, where the second and third machines need one modification to maintain primitivity. The corresponding increase for the LCARs is 0.8%, 0.4% and 2.6%, where the second and the third machines need one modification to maintain primitivity. Thus the percentage of hardware overhead for reconfiguration is lower for LCARs.

However, the length 16 LCARs are about 40% more expensive than the LFSRs of the same length in terms of the initial implementation cost. Since the area overhead is calculated as a percentage of the exact silicon area occupied by the original machine, (either LFSR or LCAR), it is expected that the cost of reconfiguration circuitry is smaller for larger size, and hence practical length, machines. The same margin of hardware increase is also reported on CALBO (cellular automata logic block observation) and BILBO (built-in logic block observation) designs [11]. The cost of the BIST resources can be reduced by using low cost XOR gates [21], and carefully designed layouts.

Overall, the hardware cost of LFSR and LCAR is still very low, compared with other BIST circuitry in implementation. One has to evaluate tradeoffs of better performance (as PRPG and delay fault detection) and area overhead. If used in the context of boundary scan, the difference in area between LFSRs and LCARs is not significant since the boundary scan cells are placed in the unused area next to the I/O pads.

8 Applications

Given the principles of LFSR/LCAR concatenation and partitioning, we outline some of their direct applications.

In testing applications, it is often desirable to configure dynamically built-in self-test (BIST) resources, such as pseudo-random pattern generators (PRPG), signature analyzers and BILBOs/CALBOs, into different lengths and/or functional blocks. Concatenation and partitioning of LFSRs/LCARs permit efficient use of the hardware resources and reduction of the overall hardware overhead for

[4]Open Architecture Silicon Implementation Software, by the Microelectronic Center of North Carolina.

testing [16, 28].

Applications of boundary scan testing have been extensively studied in the past two years and concatenation and/or partitioning of linear machines can play an important part. The recent developments reported in the literature can be categorized as follows.

(1) Applications of the boundary scan testing standard to digital systems of all kinds can be found in [5, 18, 19]. Solutions to some of the traditional problems possessed by boundary scan, such as speed limits of scan-in test patterns and the depth of scan paths, have attracted particular attentions of design engineers and researchers.

(2) The utilization of the hardware resources for boundary scan to implement conventional BIST circuitry is an important topic [7, 13, 17]. It facilitates on-chip testing, reduces the overall silicon cost and improves fault detection and system reliability.

(3) The combination of boundary scan testing with other testing techniques is also quite successfull [9, 15, 30, 28]. For instance, in [9], boundary scan and probe card technology are coupled with electrical programmable substrates to test multiple-chip module based image processing computer, during and after assembly. Lubaszewski and Courtois merge the boundary scan and the UBIST [20] to support the manufacturing, field maintenance and concurrent error detection [15]. Another recent application involves utilizing on-line, off-line BIST and boundary scan resources through the cyclic redundancy checking and provides dual testability [28].

In the applications above, concatenation and partitioning of LFSRs/LCARs and register-based devices can be used for dynamic configuration, provide flexibility of hardware resources and facilitate on-chip testing at different levels of manufacturing and maintenance processes.

Linear finite state machines are also used extensively in two other fields: cryptography, for the encryption of messages, and coding theory, for the generation of cyclic codes. In both cases, the possible dynamic reconfiguration of machines which maintain the maximal length cycle property can be very useful, as it provides a range of outputs with the minimal hardware resources. For instance, messages can be encoded dynamically choosing different lengths of machines, and thus different cyclic codes. The hardware cost remains fixed at the length of the largest machines, which would be used anyway, while the error masking can be decreased by varying the encoding.

9 Conclusion

With the increasing use of LFSMs in many applications, there are often situations where a number of machines of differing lengths are required within the same system. There are clearly advantages to being able to easily reconfigure a single larger machine to several smaller ones as required. Since it is frequently the case that all the required machines must be maximal length, it is important that this property is retained in any partitioning or concatentation of such LFSMs.

In this paper, we investigate the concatenation and partitionig properties of the two most frequently used LFSMs, namely LFSRs and LCARs. A detailed theoretical analysis of the relevant properties is accompanied by a systematic evaluation of the concatenation and partitioning of such machines for almost all the practical sizes that are required in typical systems today. All concatenations of machines up to length 64 are examined, as well as concatenations of of low-cost machines up to length 256. The results of complete study of the partitioning of all irreducible machines up to length 16 are included, from which we can extrapolate the frequency with which an arbitrary partitioning of a primitive machine will result in two primitive partitions. Since the frequency of this is quite low (and decreasing as the length increases), we give a low cost method of providing a minor modification to the register to increase the number of primitive partitions.

An analysis of the design principles, and the use of low-cost LFSRs and LCARs and the impact at both the logic and circuit level is included as well as a discussion of the comparatively low overall hardware overhead which results. The results of a detailed experiment using OASIS on three different primitive LFSRs and three different LCARs, all of length 16, are included. Overall, the hardware cost of using either LFSRs or LCARs in a system is quite low, and the extra hardware required for the flexibility of having the property that a register can be easily reconfigured to different length maximal registers is very low.

References

[1] P. H. Bardell, W. H. McAnney, and J. Savir. *Built-In Test for VLSI: Pseudorandom Techniques*. J. Wiley & Sons Inc., New York, 1987.

[2] Dilip K. Bhavsar. Concatenable polydividers: bit-sliced LFSR chips for board self-test. *Proceedings of International Test Conference*, pages 88–93, October 1985.

[3] K. Cattell and J.C. Muzio. Synthesis of one-dimensional linear hybrid cellular automata. *submitted to IEEE Transactions on Computer-Aided Design*, 1993.

[4] K. Cattell and J.C. Muzio. Tables of linear cellular automata for minimal weight primitive polynomials of degrees up to 300. Technical report, University of Victoria, Department of Computer Science, 1993.

[5] V. Chickermane, J. Lee, and J. H. Patel. Design for testability using architectural descriptions. *Proceedings of International Test Conference*, pages 752–761, 1992.

[6] Bernard Elspas. The theory of autonomous linear sequential networks. *IRE Transactions on Circuit Theory*, pages 45–60, March 1959.

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

[8] S. W. Golomb. *Shift Register Sequences*. Aegean Park Press, 1982.

[9] S. Hilla. Boundary scan testing for multichip modules. *Proceedings of International Test Conference*, pages 224–231, 1992.

[10] 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.

[11] S. L. Hurst. A hardware consideration of CALBO testing. *The Third Technical Workshop: New Directions for IC Testing*, pages 129–146, October 1988.

[12] E. Kontopidi and J.C. Muzio. The partitioning of linear registers for testing applications. *Microelectronics Journal*, 24:533–546, 1993.

[13] T. Langford. Utilizing boundary scan to implement BIST. *Proceedings of International Test Conference*, pages 167–173, 93.

[14] S. Lipschutz. *Linear Algebra*. MacGraw-Hill, 1968.

[15] M. Lubaszewski and B. Courtois. On the design of self-checking boundary scannable boards. *Proceedings of International Test Conference*, pages 372–381, 1992.

[16] M. Martinez and S. Bracho. Concatenated LFSR makes a weighted built-in logic block observation. *Proceedings of CCC, Hungary*, pages 113–121, 1993.

[17] J. Matos, F. Pinto, and J. Ferreira. A boundary scan test controller for hierarchical BIST. *Proceedings of International Test Conference*, pages 217–223, 1992.

[18] M. Muris and A. Biewenga. Using boundary scan test to test random access memory clusters. *Proceedings of International Test Conference*, pages 174–179, 1993.

[19] B. Nadeau-Dostie, D. Burek, and A. Hassan. Scan bist: A multi-frequency scan-based BIST method. *Proceedings of International Test Conference*, pages 506–513, 1992.

[20] M. Nicolaidis. Self-exercising checker for unified built-in self-test (UBIST). *IEEE Transactions on Computer-Aided Design*, pages 203–218, March 1989.

[21] Bong-Hee Park and Prem R. Menon. Robustly scan-testable CMOS sequential circuits. *Proceedings of International Test Conference*, pages 263–272, October 1991.

[22] W. W. Peterson and E. J. Weldon. *Error-Correcting Codes*. The M.I.T. Press, 1972.

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

[24] M. Serra and T. Slater. A Lanczos algorithm in a finite field and its application. *Journal of Combinatorial Mathematics and Combinatorial Computing*, pages 11–32, April 1990.

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

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

[27] X. Sun and M. Serra. Concurrentchecking and off-line data compaction testing with shared resources in PLAs. *Journal of Semicustom ICs*, 21:8–16, 1990.

[28] X. Sun and M. Serra. Merging concurrent checking and off-line BIST. *Proceedings of International Test Conference*, pages 958–967, 1992.

[29] X. Sun and M. Serra. LCAR and their concatenation. Technical report, University of Victoria, Department of Computer Science, 1993.

[30] L. Whetsela. A proposed method of accessing 1149.1 in a backplane environment. *Proceedings of International Test Conference*, pages 206–216, 1992.

[31] S. Zhang and D. M. Miller. A comparison of LFSR and cellular automata BIST. *Proceedings of the Canadian Conference on VLSI*, pages 8.4.1–8.4.9, 1990.

[32] S. Zhang, D. M. Miller, and J. C. Muzio. Determination of minimal cost one-dimensional linear hybrid cellular automata. *Electronics Letters*, pages 1625–1626, August 1991.

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