One-Dimensional Linear Hybrid Cellular Automata: Their Synthesis, Properties and Applications to Digital Circuits Testing

M. Serra, K. Cattell, S. Zhang, J.C. Muzio, D.M. Miller
Dept. of Computer Science
University of Victoria
Victoria, B.C., Canada

January 27, 2009

Abstract

The increasing use of linear hybrid cellular automata (LHCA) in VLSI design and test and other applications for such purposes as pseudo random pattern generation has made it important for users to understand their design, use and properties. In this tutorial paper, the background for cellular automata is explained, and a recent synthesis algorithm with low complexity, which solves the problem of finding a particular linear hybrid cellular automata, is described. LHCA have powerful concatenation and partitioning properties allowing several smaller maximal length LHCA to be combined into a much larger maximal length LHCA. Their performance in this regard is compared with linear feedback shift registers (LFSRs), which do not have quite as much flexibility. The basis for LHCA being better generators than LFSRs for testing delay type faults is explained by showing the richer nature of the transition pairs generated by the LHCA. [1]

1 Introduction

Recently, we have seen a large increase in both the interest in and use of linear hybrid cellular automata (LHCA) in many areas of digital design, but especially in testing applications. In this paper, we do not discuss all of the existing work on these finite state machines, but limit observes to introducing the required background which potential users need before they can make effective design decisions concerning the use of such devices. We give readers sufficient background to understand the areas where LHCA can be used effectively, how to design them, and why they perform better than linear feedback shift registers (LFSRs) for certain applications (such as generators for testing delay type faults in digital circuits).

The LHCA considered are linear finite state machines (LFSMs)[28] each composed of a one-dimensional array of cells. Cells are only allowed to communicate with their immediate neighbors. We only examine LHCA consisting of rule 90 and rule 150 cells since it is shown in [26] that this is a necessary condition for the LHCA to have a maximum length cycle which is a desirable property in most instances.

A fundamental problem for LHCA is whether there exists an LHCA for any given polynomial over GF(2). If one does, we want to identify it. Some researchers (*e.g.,* Das, *et al.* [11] and Bardell [1]) have performed exhaustive simulation studies and have conjectured that there are two LHCA for any given primitive polynomial. However, to our knowledge, no one has studied this theoretically.

[1]This work was supported in part by research grants and a strategic grant from the Natural Sciences and Engineering Research Council of Canada and by an equipment loan from the Canadian Microelectronics Corporation.

In section 3, we introduce an elegant synthesis algorithm based on the Euclidean algorithm which can easily determines an LHCA for any given irreducible polynomial of practical order (for example, it took 10 CPU minutes on a SPARC 10 workstation to find a 1600-cell LHCA). As a general result, Cattell and Muzio [8] have shown that for any given irreducible polynomial, there exist two LHCA.

The partitioning problem concerns how an LFSM with maximum length cycle can be easily divided into smaller LFSMs which also have maximum length cycles. Conversely, the concatenation problem concerns how smaller LFSMs with maximum length cycles can be easily combined into a larger LFSM which also has a maximum length cycle. LFSMs with superior partitioning and concatenation properties, *i.e.,* those which are easily split and combined without loss of the maximum length cycle, offer the most flexibility in VLSI design and test. Therefore, it is important to investigate such properties for a given LFSM. In section 4, we discuss partitioning and concatenation, as well as some potential advantages of LHCA versus LFSRs in regard to these operations.

For many years, researchers (*e.g.,* Bardell [1, 2], Hortensius, *et al.* [15, 14], and Wolfram [33]) have considered the evaluation of the effectiveness of test vectors generated by a given LFSM (LHCA or LFSR). For the detection of faults with *sequential* behavior, Furuya and McCluskey in [12] propose a method of assessing the two-vector testing capabilities of a given sequence of vectors. For a $2n$-cell LHCA, an approach is given in [23] to select an n-cell substate vector such that the corresponding vector sequence has 2^{2n} transitions. In section 5, we extend the earlier work in [34] to a general case in which, we concentrate on any k-cell substate vector and evaluate the number of distinct k-cell ($1 \leq k \leq \lfloor n/2 \rfloor$) substate vectors which produce the maximum number 2^{2k} distinct transitions for any n-cell LHCA and LFSR with maximum length cycles. The derivation is based on the concept of the partner set in [36, 35] rather than the evaluation of the rank for a specified matrix as mentioned in [12].

In section 6, simulation studies of the ISCAS85 benchmark circuits provide evidence that our analysis of transition properties presented is indeed a reasonable metric of the effectiveness of the test vector generator. We also examine the use of an LHCA as a signature register, which is one of the applications of LFSM in Built-In Self-Test. Finally in section 7, we summarize our major results and set up a framework for future work.

To appreciate the material presented later in the paper, readers unfamiliar with the area should consider the review of linear finite state machines, linear feedback shift registers, and linear hybrid cellular automata which appears in the following section.

2 Preliminaries

In this section we give a brief theoretical background of linear finite state machines and their algebraic context, and show their different representations. All arithmetic is performed in the normal binary field GF(2), so that the + operation is addition modulo 2 (Exclusive-OR).

2.1 Linear Finite State Machines

For our purposes, a *linear finite state machine* M is comprised of n single-bit memory elements, and a transition function. The value *state* of the ith memory element at time t is denoted $s_i^{(t)}$. The state of M at time t is denoted $s^{(t)}$.

The transition function f determines the state of M at time $t+1$ from the state at time t; that is $s^{(t+1)} = f(s^{(t)})$. The next state function of a machine can be described graphically using a *state graph*. For M to be *linear* means that f is a linear function from n-bit vectors to n-bit vectors, so that

$$f(a+b) = f(a) + f(b)$$

for any pair of states a and b. The transition function f can be specified as n functions $f_1, f_2, \ldots, f_n$, where the ith function calculates the next state of cell i:

$$s_i^{(t+1)} = f_i(s^{(t)}).$$

The transition function f is linear if and only if each of the f_i are linear.

The LFSM described above is *autonomous*, since it has no input. In the remainder of this paper, wherever an LFSM is used, it means an autonomous LFSM, if not otherwise specified.

A basic result in linear algebra is that a linear transformation can be represented by a transition matrix and that, conversely, such a transition matrix represents a linear transformation. Thus we can use matrices to represent and analyze an LFSM.

If, for two square matrices A and B, there exists an invertible matrix P such that $B = P^{-1} \cdot A \cdot P$, then A is said to be *similar* to B [28, page 306]. Moreover, two matrices represent the same linear operator if they are similar to each other [20, page 155]. Of importance here is the result that if two matrices are similar nonsingular state transition matrices, then their state graphs have identical cycle structures and differ only in the labeling of the states [28, page 307].

From this 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. Since the matrix can represent an LFSM, the theorem gives us the freedom to select the machine of least cost in an equivalence class.

The *characteristic polynomial* of an $n \times n$ matrix A is defined as $\det(xI - A)$ [28, page 310]. A polynomial of degree n which is not divisible by any polynomial of degree k, $(0 < k < n)$, is called *irreducible* [24, page 148]. An irreducible polynomial of degree n is *primitive* if it divides $x^m - 1$ for no m less than $2^n - 1$ [24, page 161]. The important consequence of the primitivity of a characteristic polynomial is that the corresponding LFSM in its autonomous operation traverses all possible $2^n - 1$ non-zero states, before returning to its initial configuration. This property, called a maximum length cycle, is of importance in many applications (for example, signature analysis in VLSI testing).

2.2 Linear Feedback Shift Registers

A *Linear Feedback Shift Register* (LFSR) [28] is an LFSM defined as a collection of storage elements and XOR gates which perform addition over GF(2)

chained together and controlled by a synchronous clock. There are two configurations for the LFSR: a Type I LFSR, which has the exclusive-OR gates between the cells, and a Type II LFSR, which has exclusive-OR gates on the feedback path. We assume that shifting and numbering of the cells are from left to right. For convenience, the Type I and Type II LFSRs are simply named LFSR(I) and LFSR(II), respectively.

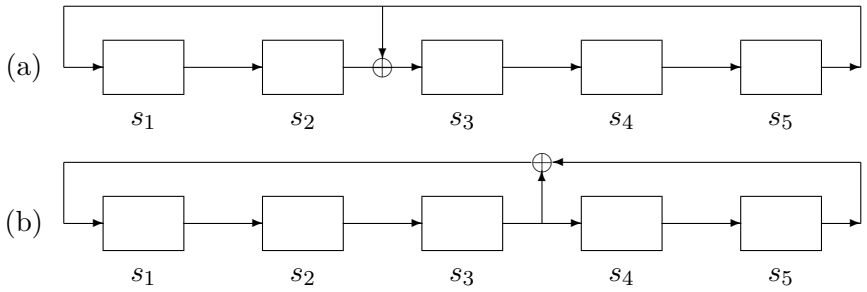


Figure 1: (a) Type I LFSR and (b) Type II LFSR with characteristic polynomial $x^5 + x^2 + 1$.

Figure 1 shows the two types of LFSRs derived from the same characteristic polynomial $x^5 + x^2 + 1$. The difference between them is the placement of the exclusive-OR gates. We can write

a set of state transition equations for any LFSR. For example, the equations for the LFSR(I) of Figure 1(a) are:

$$\begin{aligned} s_1^+ &= s_5, \\ s_2^+ &= s_1, \\ s_3^+ &= s_2 + s_5, \\ s_4^+ &= s_3, \\ s_5^+ &= s_4, \end{aligned}$$

where s_i is the present state of the cell i and s_i^+ is the next state. Simply, we can evaluate the next state as follows, with all operations being carried out over GF(2):

$$[s_1^+, s_2^+, s_3^+, s_4^+, s_5^+]^T = A \cdot [s_1, s_2, s_3, s_4, s_5]^T,$$

where A is a state transition matrix for the LFSR(I) of Figure 1(a) given by

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

Similarly, the state transition matrix for the LFSR(II) of Figure 1(b) is given by

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

The generating function $p(x)$, corresponding to an LFSR, is called the *characteristic polynomial*. This is also the characteristic polynomial of the transition matrix A. Both LFSRs in Figure 1 have $p(x) = x^5 + x^2 + 1$ as their 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 LFSR 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 cycle*. The characteristic polynomial associated with a maximum length LFSM is primitive [3, 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 [3, page 74]. Sequences generated by LFSMs are called *pseudorandom sequences,* since they are periodic and deterministic, but they have many of the properties of random sequences.

2.3 One-Dimensional Linear Hybrid Cellular Automata

Cellular automata are LFSMs, defined as uniform arrays of identical cells in an n-dimensional space, where cells are restricted to local neighborhood interaction and have no global communication. There are $2^{2^3} = 256$ possible distinct cellular automata rules in one dimension with a three-cell neighborhood. However, only the combination of *linear* rules 90 and 150 (see below), can yield an LHCA with a maximum length cycle. This is a *hybrid* LHCA since all cells do not use the same rule.

Figure 2 is an example of the type of machine being described. Each cell, labeled s_i, $1 \leq i \leq n$, 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 cells at the boundary of the array always receive a 0.

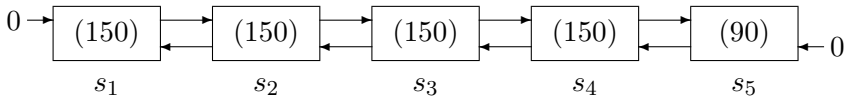


Figure 2: A 5-cell one-dimensional LHCA with characteristic polynomial $x^5 + x^2 + 1$.

The computation rules 90 and 150 are defined as follows:

$$\begin{aligned} \textit{Rule } 90: &\quad s_i^+ = s_{i-1} + s_{i+1}, \\ \textit{Rule } 150: &\quad s_i^+ = s_{i-1} + s_i + s_{i+1}. \end{aligned}$$

According to rule 90, the value of a particular cell i is the sum modulo 2 of the values of its two neighboring cells on the previous time step t. Rule 150 also includes the value of cell i at time step t. In general, we use a rule vector $[d_1, d_2, \ldots, d_n]$ to represent an n-cell LHCA, where d_i is either 0, if cell i uses rule 90, or 1, if cell i uses rule 150 for $1 \leq i \leq n$. For the example of Figure 2 the rule vector has the form $[1, 1, 1, 1, 0]$.

For any LHCA, one can 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 LHCA. For the example of Figure 2 the next state equations are

$$\begin{aligned} s_1^+ &= s_1 + s_2, \\ s_2^+ &= s_1 + s_2 + s_3, \\ s_3^+ &= s_2 + s_3 + s_4, \\ s_4^+ &= s_3 + s_4 + s_5, \\ s_5^+ &= s_4, \end{aligned}$$

and the corresponding state transition matrix is

$$\begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix},$$

which has characteristic polynomial $x^5 + x^2 + 1$. This is primitive and hence the LHCA of Figure 2 produces a maximum length cycle.

2.4 Remarks

Both LHCA and LFSRs can be represented by transition matrices, for which characteristic polynomials can be computed. The relationship between LFSRs and LHCA is stated in [26] as follows. A one-dimensional LHCA and an LFSR with the same irreducible (or primitive) characteristic polynomial are isomorphic, and the corresponding transition matrices are similar.

The consequence is that an LHCA and an LFSR, which are based on the same irreducible or primitive polynomial, have the same behavior as LFSMs up to permutation of the order in which the states appear, and *the cycle structure* of the states [28, page 307] is identical.

Figure 3 shows an LHCA and an LFSR with their corresponding transition matrices and their characteristic polynomial. The cycle structure is also 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 easily be transformed to either of the other two. A polynomial

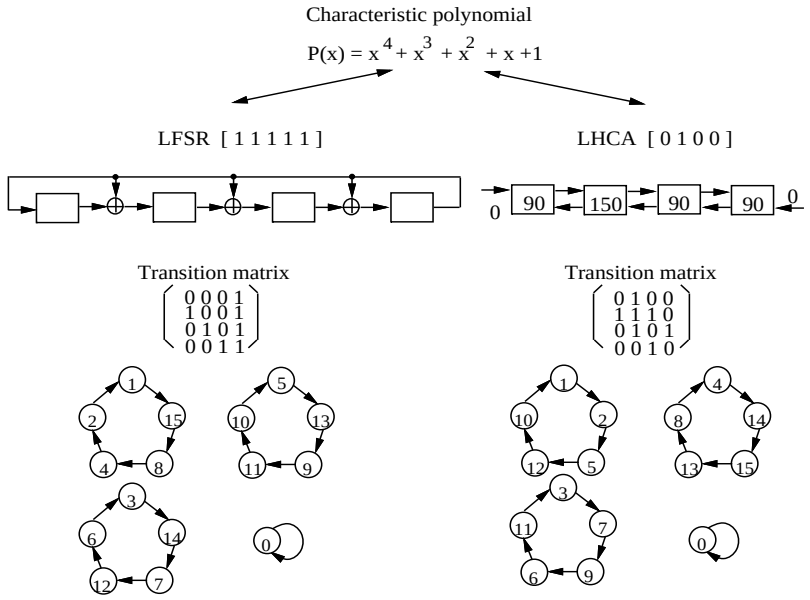


Figure 3: Transition matrices of an LFSR(I) and an LHCA.

can be directly mapped into an 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. The reverse transformations hold as well. In Figure 3, these transformations are shown on the left side.

For any given LHCA, it is easy to calculate the characteristic polynomial of its transition matrix [26]. The mapping between an LHCA 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.

Given a characteristic polynomial, three algorithms have been developed and implemented to find its corresponding LHCA [26, 8, 25]. The most recent one in [8] applies the Euclidean algorithm to compute the LHCA. It has a polynomial running time, which is sufficiently fast to generate LHCA for polynomials of very large degree. In the following section, we introduce this algorithm.

3 Synthesis of Linear Hybrid Cellular Automata

With the increasing use of LHCA in practical environments, as well as the need for longer machines, it has become a major problem to find the corresponding LHCA for a given characteristic polynomial. This situation does not arise for LFSRs, since the polynomial leads directly to the implementation. In this section we give a brief outline of the new synthesis algorithm which provides a method to find the corresponding LHCA for any irreducible polynomial very quickly. This effectively solves the synthesis problem for all practical purposes (for example, it took 10 minutes on a SPARC 10 to find a length 1600 LHCA). The complete details of the synthesis algorithm are contained in [8].

Initially, we show that the characteristic polynomial of an LHCA can be calculated efficiently, using a recurrence relation. We discuss correspondences between the characteristic polynomials of LFSRs and LHCA. It is interesting that one of the keys to the algorithm is the deduction of a relation between the Euclidean algorithm (well-known in other areas, but not often relevant to VLSI) and the computation of the characteristic polynomial of an LHCA. This provides the framework for the synthesis and proof of existence of LHCA with irreducible characteristic polynomials.

3.1 Correspondences between LHCA and LFSR Characteristic Polynomials

To find correspondences between the characteristic polynomials of LFSRs and LHCA, it is necessary to know how to calculate the characteristic polynomial for a given LFSR (or LHCA). As discussed in section 2, there is a 1-to-1 correspondence between LFSRs and polynomials of degree n. In other words, we can easily determine a unique characteristic polynomial for a given LFSR and *vice versa.* For a given n-cell LHCA $[d_1, d_2, ..., d_n]$, Serra, *et al.* [26] demonstrate that the characteristic polynomial $\Delta_n(x)$ for the LHCA can be calculated by an LHCA recurrence relation

$$\Delta_k(x) \quad = \quad (x + d_k)\Delta_{k-1}(x) + \Delta_{k-2}(x), \qquad\qquad k \geq 1 \tag{1}$$

with initial conditions $\Delta_0(x) = 1$ and $\Delta_{-1}(x) = 0$. It is easy to understand the calculation procedure from the following example.

Example 3.1 *For the LHCA* $[1, 1, 1, 1, 0]$ *in Figure 2, the characteristic polynomial is computed by equation* (1) *as follows.*

$$\begin{aligned}
\Delta_{-1}(x) &= 0, \\
\Delta_0(x) &= 1, \\
\Delta_1(x) &= (x + d_1)\Delta_0(x) + \Delta_{-1}(x) \\
&= (x + 1)1 + 0 \\
&= x + 1, \\
\Delta_2(x) &= (x + d_2)\Delta_1(x) + \Delta_0(x) \\
&= (x + 1)(x + 1) + 1 \\
&= x^2, \\
\Delta_3(x) &= (x + d_3)\Delta_2(x) + \Delta_1(x) \\
&= (x + 1)x^2 + (x + 1) \\
&= x^3 + x^2 + x + 1, \\
\Delta_4(x) &= (x + d_4)\Delta_3(x) + \Delta_2(x) \\
&= (x + 1)(x^3 + x^2 + x + 1) + x^2 \\
&= x^4 + x^2 + 1, \\
\Delta_5(x) &= (x + d_5)\Delta_4(x) + \Delta_3(x) \\
&= (x + 0)(x^4 + x^2 + 1) + (x^3 + x^2 + x + 1) \\
&= x^5 + x^2 + 1.
\end{aligned}$$

Therefore, the characteristic polynomial of the LHCA is $x^5 + x^2 + 1$. □

It can be seen that for a given n-cell LHCA, the calculation of the characteristic polynomial requires n applications of equation (1). Each application involves the multiplication of a polynomial by a degree 1 polynomial, and a polynomial addition. Since the multiplication can be performed with a "shift" and an addition, the total number of operations required is ($2n$ additions + n shifts). In short, for a given n-cell LHCA, the corresponding characteristic polynomial can be calculated in linear time.

To see whether there exists a 1-to-1 correspondence between LHCA and polynomials of degree n, our task now is to determine whether there exists an n-cell LHCA for any given polynomial of degree n. Before doing so, it is instructive to work out the numbers of different LHCA and polynomials of degree n. In general, we have 2^n different polynomials since each of the coefficients can be 0 or 1. For the n-cell LHCA, we have 2^n different choices because each of the n cells can be selected in two ways: either rule 90 or rule 150. Using equation (1), we find that an LHCA $[0, 1, 1, 1, 1]$ has the characteristic polynomial $x^5 + x^2 + 1$ which is the same as that of the

LHCA $[1,1,1,1,0]$. Moreover, there is no LHCA with a characteristic polynomial x^2+x, and there are four LHCA all having a characteristic polynomial $x^6+x^5+x^4+x^3+1$ (the LHCA are $[0,0,1,0,0,0]$, $[0,0,0,1,0,0]$, $[1,1,0,1,1,1]$ and $[1,1,1,0,1,1]$). Thus, we have enough evidence to state that in general

$$\text{LHCA} \overset{\text{not1:1}}{\longleftrightarrow} \text{polynomials} \overset{\text{1:1}}{\longleftrightarrow} \text{LFSRs}.$$

Fortunately, if we restrict our attention to irreducible polynomials and machines with irreducible polynomials, some structure becomes apparent. In this restricted setting, the situation is

$$\text{irreducible LHCA} \overset{\text{2:1}}{\longleftrightarrow} \text{irreducible polynomials} \overset{\text{1:1}}{\longleftrightarrow} \text{irreducible LFSRs}.$$

The 2-to-1 correspondence between LHCA and LFSRs with irreducible characteristic polynomials is found by exploring the connection between the LHCA and the Euclidean algorithm. The following sections show this connection, and translate the results from the domain of the Euclidean algorithm to the domain of LHCA characteristic polynomials.

3.2 Synthesis Algorithm

The *synthesis of an LHCA for a polynomial* $\Delta(x)$ is the process of obtaining an LHCA that has $\Delta(x)$ as its characteristic polynomial. The approach by Serra and Slater [25], based on a modification for finite fields of the Lanczos tridiagonalization algorithm, is tractable for irreducible polynomials with degree up to approximately 50. We describe a completely different approach to the synthesis based on the Euclidean algorithm, which relies on the division algorithm. This method is much faster and can be used to solve the synthesis problem for all practical purposes.

The division algorithm for polynomials states that for given polynomials $a(x)$ and $b(x)$ with $b(x) \neq 0$, there exist unique polynomials $q(x)$ and $r(x)$ such that

$$a(x) \quad = \quad q(x)b(x) + r(x)$$

where $\deg(r(x)) < \deg(b(x))$ or $r(x) = 0$. The polynomial $a(x)$ is the *dividend*, $b(x)$ is the *divisor*, $q(x)$ is the *quotient*, and $r(x)$ is the *remainder*.

For example, if $a(x) = x^5 + x^2 + 1$ is divided by $b(x) = x^4 + x^2 + 1$, we have

$$x^5 + x^2 + 1 \quad = \quad x(x^4 + x^2 + 1) + (x^3 + x^2 + x + 1).$$

That is, $q(x) = x$ and $r(x) = x^3 + x^2 + x + 1$.

The essential observation is that the LHCA recurrence relation in equation (1) satisfies the division algorithm, if we identify $a(x)$ with $\Delta_k(x)$ and $b(x)$ with $\Delta_{k-1}(x)$. That is, if we know $\Delta_n(x)$ and $\Delta_{n-1}(x)$, we can uniquely determine $x + d_n$ and $\Delta_{n-2}(x)$. Then, applying the division algorithm to $\Delta_{n-1}(x)$ and $\Delta_{n-2}(x)$, we can calculate $x + d_{n-1}$ and $\Delta_{n-3}(x)$. We may continue this process until we have $x + d_1$ and $\Delta_{-1}(x) = 0$.

In short, for a given irreducible polynomial $\Delta_n(x)$, assuming we are able to find $\Delta_{n-1}(x)$ (later we discuss how to do this), by applying the division algorithm we can obtain a sequence of degree 1 quotient polynomials

$$x + d_n,\ x + d_{n-1},\ \ldots,\ x + d_2,\ x + d_1,$$

where d_k $(1 \leq k \leq n)$ is either 0 or 1. By taking the constant terms of these quotient polynomials and reversing, we get

$$[d_1, d_2, \ldots, d_n],$$

which is a rule vector for an LHCA with the characteristic polynomial $\Delta_n(x)$. The whole procedure is illustrated in the following example.

Example 3.2 *Given a polynomial* $\Delta_5(x) = x^5 + x^2 + 1$*, we want the rule vector* $[d_1, d_2, d_3, d_4, d_5]$ *for an LHCA with characteristic polynomial* $\Delta_5(x)$*. Assume that we know* $\Delta_4(x) = x^4 + x^2 + 1$*. First, we apply the division algorithm five times as follows.*

$$\begin{aligned}
\Delta_5(x) &= x^5 + x^2 + 1 \\
&= (x+0)(x^4 + x^2 + 1) + (x^3 + x^2 + x + 1) \\
&= (x + d_5)\Delta_4(x) + \Delta_3(x), \\
\Delta_4(x) &= x^4 + x^2 + 1 \\
&= (x+1)(x^3 + x^2 + x + 1) + x^2 \\
&= (x + d_4)\Delta_3(x) + \Delta_2(x), \\
\Delta_3(x) &= x^3 + x^2 + x + 1 \\
&= (x+1)x^2 + (x+1) \\
&= (x + d_3)\Delta_2(x) + \Delta_1(x), \\
\Delta_2(x) &= x^2 \\
&= (x+1)(x+1) + 1 \\
&= (x + d_2)\Delta_1(x) + \Delta_0(x), \\
\Delta_1(x) &= x + 1 \\
&= (x+1)1 + 0 \\
&= (x + d_1)\Delta_0(x) + \Delta_{-1}(x).
\end{aligned}$$

Now, we have got a sequence of degree 1 *quotient polynomials*

$$x + d_5,\ x + d_4,\ x + d_3,\ x + d_2,\ x + d_1$$

which is

$$x + 0,\ x + 1,\ x + 1,\ x + 1,\ x + 1.$$

By taking the constant terms and reversing, we get

$$[d_1, d_2, d_3, d_4, d_5] = [1, 1, 1, 1, 0]$$

which is the rule vector for the LHCA used in Example 3.1. The correctness of the calculation is illustrated by Examples 3.1 and 3.2. □

Note that we calculate $\Delta_k(x)$ by applying the LHCA recurrence relation to $x + d_k$, $\Delta_{k-1}(x)$, and $\Delta_{k-2}(x)$ for k from 1 up to n, whereas we determine unique $x + d_k$ and $\Delta_{k-2}(x)$ by applying the division algorithm to $\Delta_k(x)$ and $\Delta_{k-1}(x)$ for k from n down to 1. That is, the orderings of the calculation are different.

In general, a sequence of applications of the division algorithm is summarized by

$$\begin{aligned}
\Delta_n(x) &= (x + d_n)\Delta_{n-1}(x) + \Delta_{n-2}(x), \\
\Delta_{n-1}(x) &= (x + d_{n-1})\Delta_{n-2}(x) + \Delta_{n-3}(x), \\
&\vdots \\
\Delta_2(x) &= (x + d_2)\Delta_1(x) + 1, \\
\Delta_1(x) &= (x + d_1)1 + 0.
\end{aligned}$$

The computation as a whole is the Euclidean algorithm applied to the polynomials $\Delta_n(x)$ and $\Delta_{n-1}(x)$, and the division algorithm ensures that this computation is unique. Since the division algorithm only uses one polynomial division operation and we need at most n applications of the division algorithm in the Euclidean algorithm, then clearly, we can determine the quotient polynomials in linear time, given that we know the $\Delta_{n-1}(x)$ for the irreducible polynomial $\Delta_n(x)$.

It remains to discuss how to find the $\Delta_{n-1}(x)$ for a given irreducible polynomial $\Delta_n(x)$. Cattell and Muzio [8] show that $g(x) = \Delta_{n-1}(x)$ satisfies the following equation

$$g^2(x) + (x^2 + x)\Delta'_n(x)g(x) + 1 \equiv 0 \bmod \Delta_n(x), \tag{2}$$

where the polynomial $\Delta'_n(x)$ is the formal derivative of $\Delta_n(x)$, which is calculated in the same manner as the derivative for integer-coefficient polynomials, but with modulo 2. For example, given $\Delta_5(x) = x^5 + x^2 + 1$, we have

$$\Delta'_5(x) \;=\; (x^5 + x^2 + 1)' \;=\; 5x^4 + 2x \;=\; x^4.$$

Finding solutions to equation (2) involves solving a quadratic equation in the finite field GF(2^n). Though the formula for solving rational quadratic equations can not be used, there are well-known techniques for this problem ([19, **?**]). Once we compute the solution, we can apply the Euclidean algorithm to derive the rule vector for the LHCA. In summary, the Euclidean algorithm and equation (2) provide a synthesis algorithm which produces LHCA for given irreducible polynomials.

The synthesis algorithm has been used in [7] to generate LHCA for each of the primitive polynomials given in [3] (one of each degree up to 300). Table 1 contains a small subset of the results, where the time column gives the total CPU time in seconds on a SPARC 10 workstation. The largest LHCA synthesized consists of 1600 cells, which was calculated in 10 CPU minutes.

Polynomial	*Time*	*LHCA*
$x^{20} + x^3 + 1$	0.2	`[01101011100001010110]`
$x^{40} + x^{21} + x^{19} + x^2 + 1$	0.4	`[1100110000011000000100010100000100110011]`
$x^{60} + x + 1$	0.6	`[111001111010010111010000101111001101000010110110010111100111]`
$x^{80} + x^{38} + x^{37} + x + 1$	0.8	`[01010110010000100000101000110011101111011110101011011101111000000100001001101010]`

Table 1: Example running time of LHCA synthesis program.

Note that for any given irreducible polynomial, we can get two distinct solutions, both satisfying equation (2). Referring to the work of Mesirov and Sweet in [22], Cattell and Muzio in [8] show that for any given irreducible polynomial of degree n, there exist exactly two corresponding LHCA. The rule vectors of the two LHCA are distinct, but reversals of each other.

3.3 Similarity Transforms between LHCA and LFSRs

We now give an explicit form for a similarity transform between the transition matrices of an LHCA and an LFSR (The details can be found in [6]). To this end, the *trace* function of a polynomial $f(x)$, with respect to $\Delta(x)$ of degree n, is defined as

$$\mathrm{Tr}(f(x)) = (f(x) + f^2(x) + f^4(x) + \cdots + f^{2^{n-1}}(x)) \bmod \Delta(x).$$

The trace of $f(x)$ is always either 0 or 1 for the irreducible polynomial $\Delta(x)$. For example, given $f(x) = x$ and $\Delta(x) = x^5 + x^2 + 1$, we have

$$\begin{aligned}
\mathrm{Tr}(f(x)) &= (f(x) + f^2(x) + f^4(x) + f^8(x) + f^{16}(x)) \bmod \Delta(x) \\
&= (x + x^2 + x^4 + x^8 + x^{16}) \bmod (x^5 + x^2 + 1) \\
&= 0 \bmod (x^5 + x^2 + 1) \\
&= 0.
\end{aligned}$$

On the basis of the *trace* function and materials discussed previously, we have the following result. For any given irreducible polynomial $\Delta(x)$ of degree n, let T_{ca} and T_{sr} be the corresponding

transition matrices for an LHCA and an LFSR (Type I) respectively. The similarity transform between $T_{\rm ca}$ and $T_{\rm sr}$ is

$$T_{\rm ca} \;=\; P \cdot T_{\rm sr} \cdot P^{-1},$$

where the matrix P is defined as

$$P = \begin{bmatrix} \mathrm{Tr}(x\Delta_0) & \mathrm{Tr}(x^2\Delta_0) & \mathrm{Tr}(x^3\Delta_0) & \cdots & \mathrm{Tr}(x^n\Delta_0) \\ \mathrm{Tr}(x\Delta_1) & \mathrm{Tr}(x^2\Delta_1) & \mathrm{Tr}(x^3\Delta_1) & \cdots & \mathrm{Tr}(x^n\Delta_1) \\ \mathrm{Tr}(x\Delta_2) & \mathrm{Tr}(x^2\Delta_2) & \mathrm{Tr}(x^3\Delta_2) & \cdots & \mathrm{Tr}(x^n\Delta_2) \\ \vdots & \vdots & \vdots & & \vdots \\ \mathrm{Tr}(x\Delta_{n-1}) & \mathrm{Tr}(x^2\Delta_{n-1}) & \mathrm{Tr}(x^3\Delta_{n-1}) & \cdots & \mathrm{Tr}(x^n\Delta_{n-1}) \end{bmatrix}$$

where Δ_k $(0 \le k < n)$ means a polynomial $\Delta_k(x)$ of degree k, which is evaluated by the LHCA recurrence relation or the Euclidean algorithm mentioned previously.

The theory provided in this section is a key to deriving an explicit form for a similarity transform between the transition matrices of an LHCA and an LFSR. For example, for the LHCA $[1, 1, 1, 1, 0]$ in Figure 2, we have the following transition matrix

$$T_{\rm ca} \;=\; \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

with characteristic polynomial $x^5 + x^2 + 1$. By the theory in this section and the extension work in [6], we can calculate the corresponding matrices P and P^{-1} of the LHCA as follows.

$$P \;=\; \begin{bmatrix} 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \end{bmatrix}, \qquad P^{-1} \;=\; \begin{bmatrix} 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \end{bmatrix}.$$

We can easily verify the similarity transform

$$T_{\rm ca} \;=\; P \cdot T_{\rm sr} \cdot P^{-1},$$

where

$$T_{sr} \;=\; \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

which is the transition matrix of the LFSR(I) with characteristic polynomial $x^5 + x^2 + 1$ in Figure 1(a). □

4 Concatenations and Partitions

In order to provide the required flexibility to designers working with Built-In Self-Test (BIST), there is some interest in having a maximal length LFSR or LHCA which can be easily split into

smaller LFSRs or LHCA which are also maximal length. An investigation of these properties is the primary thrust of this section. Our main interest lies in finding LFSRs and LHCA which can be partitioned and/or concatenated and still maintain their primitivity. That is, we are interested in maximum length LFSRs and LHCA which can be easily concatenated into another maximum length machine, and in those which can be easily partitioned into two or more maximum length machines. While we only discuss maximum length machines here, a comprehensive analysis of the concatenation and partitioning properties of general linear machines can be found in [18, 29]. Rather than giving the formal, theoretical definitions of partitioning and concatenation, we explain the concepts with the aid of Figures 4 and 5.

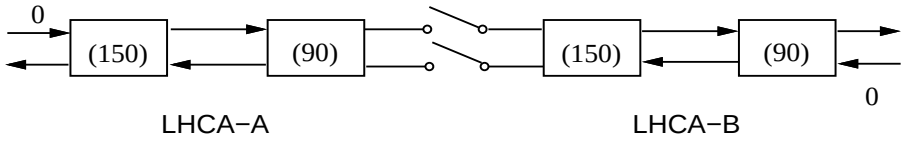


Figure 4: LHCA concatenation.

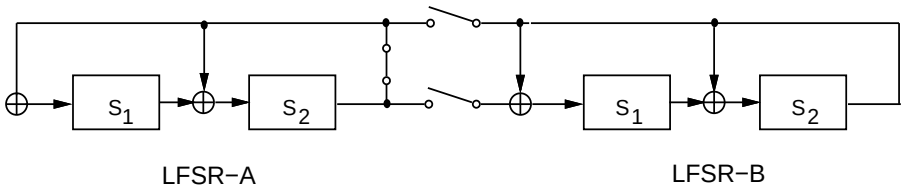


Figure 5: LFSR concatenation.

Figure 4 shows the concatenation of two smaller LHCA with a controllable link between them. In Figure 5 we have a similar concatenation of two smaller LFSRs with a controllable link between them. Note the presence of the extra XOR at the join (in theory, this can be omitted, but in that case the resulting machines which we discuss below would all be reducible, and so of little interest). Here we concentrate solely on the *primitive concatenation* of primitive machines, and on the *primitive partitioning* of a primitive machine (that is, all the partitions are primitive). This means that there is a certain symmetry about the results which is not true in general - in that two non-primitive machines may be concatenated to form a primitive machine, and a non-primitive machine may be partitioned into two primitive partitions. The concatenation of a polynomial of degree s with itself n times ($n > 1$) to form a polynomial of degree $n \times s$ is called *self-concatenation*. It is called *non-self-concatenation* if the polynomial concatenates with one or more different polynomial(s).

Example 4.1 *The LFSR based on the polynomial* $x^{16}+x^{14}+x^{13}+x^{10}+x^8+x^6+x^5+x^2+1$ *(which is primitive) can be partitioned into two length eight primitive LFSRs, both of which implement the same polynomial* $x^8+x^6+x^5+x^2+1$. *Alternatively, we can think of this as the self-concatenation twice of the latter polynomial.*

Obviously, we can either consider two smaller machines and concatenate them, or discuss the longer machine and a partition of it into two smaller machines. If an LFSM of length n is partitioned into two bit-slices of length r and s respectively, such that $r + s = n$, $r = 1$ or $s = 1$, then it is called a *degenerate partitioning*. If an LFSM of length n is partitioned into two bit-slices of length r and s respectively, such that $r + s = n$, $1 < r, s < n - 1$, then it is called a *proper partitioning*. Hence, an LFSM of length n has $(n - 1)$ possible partitionings, two degenerate partitionings, and $(n - 3)$ proper partitionings.

Degree	*Polynomial*	*LFSR Concatenation*	*LHCA*	*LHCA Concatenation*
2	[111]	-	[10]	2,3,5,6,9,11,14
3	[1011]	2,3,9	[110]	4,16
	[1101]	2,3,9	[001]	4
4	[10011]	-	[1010]	3,7,13,15
	[11001]	-	[1101]	17
5	[100101]	3,12	[11110]	-
	[101001]	3,12	[01100]	3,9
	[110111]	5	[10000]	-
	[111011]	5	[11100]	6,7
	[101111]	-	[11000]	6,7
	[111101]	-	[11001]	3,8,9
6	[1000011]	3,5	[011000]	10
	[1100001]	3,5	[011010]	2,6,8
	[1100111]	2	[101001]	2,4,6
	[1110011]	2	[100000]	3
	[1011011]	-	[101110]	9
	[1101101]	-	[101010]	2,3,10

Table 2: Self-concatenation of degree 3 to 6 machines.

4.1 Concatenation

In general, concatenation can be examined from three different perspectives: the method of concatenation (self or non-self), the attributes of the resulting machines (primitive or not), and the attributes of the participating machines (primitive or not).

Our only interest is the concatenation of the primitive machines. Only a comparatively small number of primitive machines can be self-concatenated into a primitive machine, and in Table 2 we list *the primitive self-concatenations* of primitive LFSRs and LHCA of degree 3 to degree 6, forming machines of much longer lengths. The second column of each table lists all the primitive polynomials of these degrees, which are used as the initial polynomial for concatenation (the polynomials are represented by their coefficients, in decreasing degree order, so that [1011] is $x^3 + x + 1$. The third column, labeled "LFSR Concatenation", gives the number of primitive self-concatenations of the corresponding initial LFSR which result in primitive LFSRs. The fourth column gives the LHCA realization for each of the polynomials (the polynomial itself immediately gives the LFSR realization). The fifth column, "LHCA Concatenation", gives the information of the self-concatenations for LHCA. For example, for polynomial [101001] of degree 5, with an LHCA implementation of [01100], the table shows that both three and twelve copies of the LFSR when self-concatenated give primitive machines, while for the LHCA, three copies and nine copies, when self-concatenated, give primitive machines. In all cases listed, both the initial and the resulting polynomials are primitive. The information in column 3 comes from Bhavsar in [4]. The table itself is part of a much larger set of results that are contained in [29].

4.2 Partitioning

The partitioning behavior of LFSMs, for all degree 2 to 16 primitive and irreducible polynomials, are examined, with extrapolation of the results to higher degrees. The study is restricted to proper partitionings. There are two properties of interest in evaluating the partitioning properties, namely

(1)) *ATLOP:* the number of degree n primitive polynomials which have AT Least One Partitioning with both parts being primitive.

(2) *PEPP:* the PErcentage of the total number of Partitions where both parts are primitive.

Degree	*Number of Polynomials*	*LFSR*		*LHCA*	
		PEPP	*ATLOP*	*PEPP*	*ATLOP*
4	2	0	0	50.00	1
5	6	33.33	4	16.66	2
6	6	22.22	4	27.77	4
7	18	16.66	10	18.05	11
8	16	12.50	10	12.50	8
9	48	9.72	22	12.15	27
10	60	8.57	26	11.42	40
11	176	6.81	80	8.59	91
12	144	6.32	70	7.40	78
13	630	5.42	276	6.38	311
14	756	4.93	336	5.73	378
15	1800	3.94	704	5.53	910
16	2048	4.15	904	4.81	1002

Table 3: The partitioning behavior of LFSRs and LHCA - primitive.

The partitioning behavior of LFSRs and LHCA of length up to 16 which implement primitive polynomials is given in Table 3. Table 3 shows that, for increasing values of n up to 16, the percentage of primitive partitions of both primitive LFSRs and LHCA decreases gradually as n increases.

A comparative graph for primitive machines is shown in the two lower graphs of Figure 6. 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 the LHCA almost always exceeds the performance of the LFSRs. Note that although primitive LFSRs and LHCA have primitive partitions, the number of such partitions is quite small.

4.3 Modifications

Since there are a comparatively small number of primitive partitions of primitive LFSRs and LHCA, we consider ways to improve the partitioning (or concatenation) behavior and show that better performance for both LFSRs and LHCA can be achieved by allowing minimal hardware modifications, which have very low cost. We define *one modification* to an LFSR as the introduction or the elimination of a nonzero term in its characteristic polynomial. Alternatively, we define *one modification* in an LHCA as the reconfiguration of a rule 90 cell to a rule 150 cell or vice versa. All primitive machines up to length 16 have been investigated (for similar results for irreducible machines, see [18]). For each length ($n = 4, 5, \ldots, 16$) the values of *ATLOP* and *PEPP* have been found, allowing one modification. The partitioning behavior of primitive LFSRs and LHCA up to degree 16, when minimum modifications are allowed, is shown in Table 4.

Example 4.2 *The LFSR with characteristic polynomial* $x^{16} + x^{10} + x^9 + x^7 + x^6 + x + 1$ *can be partitioned into two length eight LFSRs with characteristic polynomials* $x^8 + x^7 + x^6 + x + 1$ *and* $x^8 + x^2 + x + 1$*, respectively. Only the first LFSR 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.*

The better partitioning behavior with the introduction of one change is evident from the values of *PEPP* and *ATLOP* shown in these tables. It is clearly illustrated in the two graphs of Figure 6. 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.

We can draw the following conclusions from these results:

Degree	*Number of Prim. Polynomials*	*PEPP LFSR*	*PEPP LHCA*	*ATLOP LFSR*	*ATLOP LHCA*
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 4: The partitioning behavior of LFSRs and LHCA (primitive) with one modification.

(1) LFSRs and LHCA 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 purpose.

(2) Without allowing modifications, LHCA behave slightly better than LFSRs. After the introduction of minimum modifications LHCA are always superior to LFSRs. Intuitively, this is because we can try more changes in LHCA. More specifically, in an LHCA of length n we are able to try n changes, i.e., each of the n cells can be reconfigured either in rule 90 cell or in rule 150 cell. However, in an LFSR of length n there are only $n-1$ possible changes.

(3) The percentage of primitive partitions of primitive machines (*PEPP*) exhibits consistent behavior. It decreases as the machine length increases. It is anticipated that this behavior can be extrapolated to longer length machines. The extrapolation is clearer if separate plots are made of even and odd degree polynomials. In these cases, *PEPP* decreases linearly with the increase in the machine length.

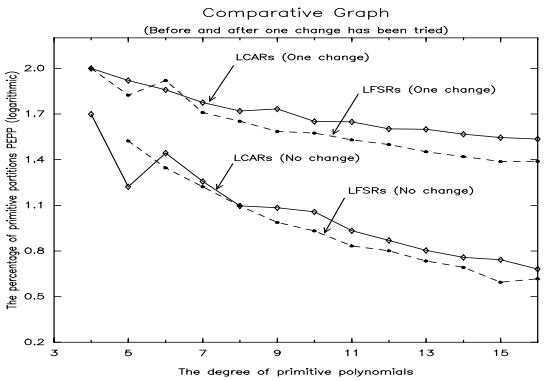
4.4 Applications

Given the possibility of concatenation and partitioning, as well as some potential advantages of LHCA versus LFSRs, we summarize some of the applications in the area of VLSI design and test.

(1) The principle of concatenation allows efficient use of BIST resources, such as pseudo-random pattern generator (PRPG), signature analyzer and CALBO/BILBO at function, chip, printed circuit board or system level. It includes dynamic reconfiguration of the devices into different lengths or for different functionalities. One such recent design can be found in [21].

(2) In a boundary scan environment, a chip may consist of several functional blocks with different numbers of inputs and outputs. Concatenation can be used to reconfigure the boundary scan cells into BIST circuitry to facilitate on-chip test of the functions.

(3) New ways to merge boundary scan with BIST of printed circuit boards (PCB) have been proposed in many papers, *e.g.,* [13]. This approach makes use of the hardware resource for boundary scan as part of BIST circuitry, such that the total cost in boundary scan and BIST is reduced. Using the principle of LHCA concatenation, the scheme in [13] can be improved



Note: $PEPP = \frac{P_{prim} \times 100}{(n-3) \times prim(n)}$. For the LFSRs, the value of PEPP for $n = 4$ is zero excluded from the graph.

Figure 6: LFSRs - LHCA: The improved percentage of primitive partitions.

by means of dynamic reconfiguration of the source and interior machines, avoiding the m-bit delay in the original design.

(4) A recent application involves merging on-line checking, off-line BIST and boundary scan circuits as one unit and provide all three kinds of testability [30]. In the on-line checking mode, small 2 or 3-bit machine cells comprise the code generator for a self-checking circuit designed with cyclic codes. When the system enters a test mode in order to execute off-line testing with signature analysis, the machine cells are reconfigured by concatenation into longer ones, providing a PRPG and a data compactor. Thus resources are shared between two different testing schemes, one operating during the normal operation of the circuit, with considerable saving in area overhead when compared with a design providing on-line and off-line testing separately.

5 Two-Vector Transition Property

Numbers *chosen at random* are useful in a very wide variety of applications such as simulation, sampling, numerical analysis, computer programming, decision making, and recreation [17]. In this section, we consider *pseudorandom* number sequences, generated in a deterministic way using an LFSR or LHCA, to be used as a BIST generator to apply to a circuit under test. It is important to investigate the *randomness* of the sequences produced by the generators because it affects the number of detected faults for a given fault model.

Knuth [17] introduces nine kinds of specific tests that have been applied to sequences. Unfortunately, those tests are based on real or integer numbers. Therefore, it is not easy to decide on the relationship between those tests and the testability for the given fault model in BIST.

When an LFSM is used as the generator in BIST, it is sequenced through a number of states with each state serving as a test vector. Table 5 shows the test vectors produced by the LFSR(I) and LFSR(II) of Figure 1, and LHCA of Figure 2. The test vectors produced cover all possible nonzero states, beginning from a nonzero state. Thus, they are all maximum length cycles (with

Time	*LFSR(I)*	*LFSR(II)*	*LHCA*	*Time*	*LFSR(I)*	*LFSR(II)*	*LHCA*
0	0̲0001	0̲0001	00001	16	00110	00111	1̲0001
1	1̲0100	1̲0000	00̲010	17	00011	00011	11010
2	01010	01̲000	00111	18	10101	10001	00011
3	00101	001̲00	0̲1011	19	11110	11000	00101
4	10110	1001̲0	11001	20	01111	01100	01100
5	01011	01001	00110	21	10011	10110	10010
6	10001	10100	01001	22	11101	11011	11111
7	11100	11010	11110	23	11010	11101	01111
8	01110	01101	01101	24	01101	01110	10111
9	00111	00110	10000	25	10010	10111	10011
10	10111	10011	11000	26	01001	01011	11101
11	11111	11001	001̲00	27	10000	10101	01000
12	11011	11100	0111̲0	28	01̲000	01010	11100
13	11001	11110	10101	29	001̲00	00101	01010
14	11000	11111	10100	30	0001̲0	00010	11011
15	01100	01111	10110	31	00001	00001	00001

Table 5: Test vectors produced by the LFSR(I), LFSR(II), and LHCA.

length $2^n - 1 = 2^5 - 1 = 31$). Moreover, since the LFSR(I), LFSR(II), and LHCA have the same primitive characteristic polynomial, they produce the same output stream in each bit position, flowing from each single cell of the generators [2], *e.g.,* starting at 0̲ marked for state s_1 in Table 5, we can see that three sequences on s_1 for the LFSR(I), LFSR(II) and LHCA are identical. Such a property is called the correlation between the outputs of an LFSR (or LHCA), which is important in understanding the *pseudorandom* behavior of the machine. These correlations are known as *structural dependencies.*

In this section, we discuss two-vector transition property by observing a sequence of states with each state encoding serving as a test vector produced by an LFSM generator. In the following section, we show that the transition property is a useful measure for testing faults requiring a pair of vectors. When testing for certain types of faults, appropriate pairs of vectors are required. Consequently, the percentage of all possible transition pairs which are generated by subsets of the LFSM is directly relevant to the likely fault coverage that results. This is examined further in section 6. We start by defining the k-cell substate vector, and the corresponding transitions.

5.1 Transitions and Bounds

For a given n-cell LFSM state vector $\mathbf{s} = (s_1, s_2, \ldots, s_n)$, $s_p \in \{0, 1\}$, $1 \leq p \leq n$, a k-cell substate vector $\mathbf{w}$ of $\mathbf{s}$ is defined by

$$\mathbf{w} \quad = \quad (s_{i_1}, s_{i_2}, \ldots, s_{i_k}),$$

and a transition corresponding to $\mathbf{w}$ is defined as

$$\langle (s_{i_1}, s_{i_2}, \ldots, s_{i_k}), (s^+_{i_1}, s^+_{i_2}, \ldots, s^+_{i_k}) \rangle,$$

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

For notational convenience, $\overline{\mathbf{w}}$ is used to denote a substate vector of $\mathbf{s}$ with cells which are not in $\mathbf{w}$. We count one transition even if $(s_{i_1}, s_{i_2}, \ldots, s_{i_k}) = (s^+_{i_1}, s^+_{i_2}, \ldots, s^+_{i_k})$ because it simplifies the derivation of a general equation to evaluate the number of transitions for a given substate vector.

Example 5.1 *For the three LFSMs in Table 5, if* $\mathbf{s} = (s_1, s_2, s_3, s_4, s_5) = (00111)$ *and* $\mathbf{w} = (s_1, s_3, s_4)$*, then the transition corresponding to* $\mathbf{w}$ *is* $\langle (s_2, s_3, s_4), (s^+_2, s^+_3, s^+_4) \rangle$*, which is*

$$\langle (011), (011) \rangle \qquad \textit{for the LFSR(I) because } s^+ = (10111),$$

Substate Vector	$LFSR(I)$	$LFSR(II)$	$LHCA$
(s_1, s_2)	8	8	8
(s_1, s_3)	16	16	16
(s_1, s_4)	16	16	16
(s_1, s_5)	8	16	16
(s_2, s_3)	16	8	16
(s_2, s_4)	16	16	16
(s_2, s_5)	16	16	16
(s_3, s_4)	8	8	16
(s_3, s_5)	16	16	16
(s_4, s_5)	8	8	8

Table 6: An example of the number of the transitions.

$\langle(011),(001)\rangle$ *for the* $LFSR(II)$ *because* $s^+ = (00011)$,
$\langle(011),(101)\rangle$ *for the* $LHCA$ *because* $s^+ = (01011)$.

For any particular substate vector, we can count the total number of transitions for the given substate vector for the LFSM. For example, for $\mathbf{w} = (s_3, s_4)$ *of the LFSR(I) in Table 5, we have the following transitions:* $\langle(00),(10)\rangle$, $\langle(10),(01)\rangle$, $\langle(01),(10)\rangle$, $\langle(10),(11)\rangle$, $\langle(11),(01)\rangle$, $\langle(01),(00)\rangle$, $\langle(11),(11)\rangle$, $\langle(00),(00)\rangle$, *giving a total of* 8 *transitions. Obviously, the maximum number of transitions in this case is* 16 *(four possible choices for the first vector of the pair, and four for the second), so this generator only produces half of the maximum possible number of transitions for this substate. The complete list of all the transitions for* 2*-cell substate vectors for the LFSMs from Table 5 is given in Table 6.* □

As shown in Table 6, if we consider different k-cell substate vectors, there are some differences between the numbers of distinct transitions generated by the LHCA and the LFSR. Before discussing details of transitions, it is instructive to know the following upper and lower bounds for transitions [34].

Consider any n-cell LFSM vector generator with a maximum length cycle. Let $\mathcal{F}(k)$ be the maximal number of distinct transitions corresponding to a k-cell substate vector $\mathbf{w}$. We have

$$\textit{upper bound:} \quad \mathcal{F}(k) \le \begin{cases} 2^{2k}, & 1 \le k < \lceil n/2 \rceil \\ 2^n - 1, & \lceil n/2 \rceil \le k \le n \end{cases}$$

$$\textit{lower bound:} \quad \mathcal{F}(k) \ge \begin{cases} 2^k, & 1 \le k < n \\ 2^n - 1, & k = n. \end{cases}$$

It can be seen that for any k-cell substate vector $\mathbf{w}$ of an n-cell LFSM state vector $\mathbf{s} = (s_1, s_2, \ldots, s_n)$ with $k = \lfloor n/2 \rfloor$, if n is odd, the maximum possible number of distinct transitions produced by $\mathbf{w}$ is 2^{2k}, and if n is even, the maximum possible number of distinct transitions produced by $\mathbf{w}$ is $2^{2k} - 1 = 2^n - 1$ because the all zeros state is not included in the sequence produced by the LFSM. Without loss of generality, when n is even and $k = \lfloor n/2 \rfloor$, we assume that $\mathbf{w}$ can produce 2^{2k} transitions in the best case.

It is proposed in [12] that transition coverage for $k = \lfloor n/2 \rfloor$ could be used as a universal metric of transition capability of an LFSM for the following reasons. Assume a k-cell substate vector $\mathbf{w}$ of $\mathbf{s} = (s_1, s_2, \ldots, s_n)$ can produce 2^{2k} transitions. Then it follows that

(a) Any m-cell substate vector of $\mathbf{w}$ can produce 2^{2m} transitions;

(b) Any m-cell substate vector, which includes $\mathbf{w}$, of $\mathbf{s}$ can produce at least 2^{2k} transitions.

5.2 Partners and Partner Set

In the next subsection, to compare the performance of the LHCA and LFSR concerning testing faults requiring a pair of vectors, we derive the number of different k-cell substate vectors, which have 2^{2k} transition capability, with $k \leq \lfloor n/2 \rfloor$ for the LHCA and LFSR. A key to determining if a given k-cell substate vector $\mathbf{w}$ has 2^{2k} transition capability is to check whether the corresponding $T_{\overline{w}}$ has rank k (*see* [12]). However, to avoid computing the rank of $T_{\overline{w}}$, we introduce the idea of a partner set and show that the cardinality of this set gives us the rank of $T_{\overline{w}}$.

Let $\mathbf{w}$ be a substate vector of a state vector $\mathbf{s} = (s_1, s_2, \ldots, s_n)$ for an n-cell LFSM. The next state s_i^+ corresponding to s_i in $\mathbf{w}$ is given by

$$s_i^+ = \sum_{s_j \in K_i} s_j, \qquad \textit{for some subset } K_i \subseteq \{s_1, s_2, \ldots, s_n\}.$$

(Note that the subset K_i for a specific s_i depends on the transition matrix for the given LFSM.) All such s_j, which are not in $\mathbf{w}$, are eligible partners for s_i.

In other words, partners are those states that are not in $\mathbf{w}$, but have an immediate effect on the next state function corresponding to the state in $\mathbf{w}$.

In order to define the partner set, we need to make use of a maximal matching in a bipartite graph [10], so we discuss this briefly. Let $G = (V, E)$ be a bipartite graph with V partitioned as $X \cup Y$ (Each edge of E has the form (x, y) with $x \in X$ and $y \in Y$).

(a) A *matching* of G is a subset of E such that no two edges share a common vertex in X or Y.

(b) A *maximal matching* in G is one that matches as many vertices in X as possible with vertices in Y.

We use the concept of the maximal matching to define the partner set as follows.

Let $\mathbf{w}$ be a k-cell substate vector of a state vector and $G = (V, E)$ be a bipartite graph with V partitioned as $X \cup Y$, where

$$\begin{aligned} V &= \{s_1, s_2, \ldots, s_n\}, \\ X &= \{\ s_i \mid s_i \textit{ is in } \mathbf{w}\ \}, \\ Y &= \{\ s_j \mid s_j \textit{ is in } \overline{\mathbf{w}}\ \}, \\ E &= \{\ (s_i, s_j) \mid s_i \in X, s_j \in Y, \textit{ and } s_j \textit{ is an eligible partner of } s_i\ \}. \end{aligned}$$

If M, $M \subseteq E$, is a maximal matching of G, then $ps(\mathbf{w}) = \{\ s_j \mid (s_i, s_j) \in M\ \}$ is a partner set of $\mathbf{w}$ and $|ps(\mathbf{w})| = |M|$.

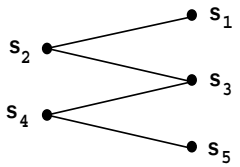


Figure 7: An example for the maximal matching and partner set.

Example 5.2 *For the LHCA shown in Figure 2, let* $\mathbf{w} = (s_2, s_4)$. *The next state functions for* s_2 *and* s_4 *are given by*

$$\begin{aligned} s_2^+ &= s_1 + s_2 + s_3, \\ s_4^+ &= s_3 + s_4 + s_5. \end{aligned}$$

That is, $K_2 = \{s_1, s_2, s_3\}$ *and* $K_4 = \{s_3, s_4, s_5\}$*. By definition of the partner, the eligible partners are* s_1 *and* s_3 *for* s_2*, and* s_3 *and* s_5 *for* s_4*.*

A bipartite graph G *for* w *is shown in Figure 7 with* $X = \{s_2, s_4\}$*,* $Y = \{s_1, s_3, s_5\}$*, and* $E = \{(s_2, s_1), (s_2, s_3), (s_4, s_3), (s_4, s_5)\}$*. The possible maximal matchings are* $\{(s_2, s_1), (s_4, s_3)\}$*,* $\{(s_2, s_1), (s_4, s_5)\}$*, and* $\{(s_2, s_3), (s_4, s_5)\}$*. Thus, the corresponding partner sets are* $\{s_1, s_3\}, \{s_1, s_5\}$*, and* $\{s_3, s_5\}$*.* □

It is shown in [36] that given a k-cell substate vector $\mathbf{w}$ of a state vector $\mathbf{s} = (s_1, s_2, \ldots, s_n)$ for an n-cell LFSM (LHCA or LFSR) with a maximum length cycle, $|ps(\mathbf{w})| = r \leq k$ if and only if $rank(T_{\overline{w}}) = r$, *i.e.,* $\mathbf{w}$ can produce $2^k \cdot 2^r = 2^{k+r}$ distinct transitions because for each value of $\mathbf{w}$ ($\mathbf{w}$ takes 2^k distinct values) there is 2^r transitions. In particular, $\mathbf{w}$ can produce 2^{2k} distinct transitions when $r = k$.

Example 5.3 *For a* 5*-cell LFSR(II), as shown in Figure 1(b), by the concept of the partner set, we select all* 2*-cell substate vectors, reported in Table 7, which produce* $2^{2\times 2} = 16$ *transitions. By referring to Tables 6 and 7, we can see that the selections in Table 7 are correct and no other* 2*-cell substate vector can be chosen to achieve* 16 *transitions.* □

Substate Vector $\mathbf{w}$	*Partner Set* $ps(\mathbf{w})$	*Notes*
(s_1, s_3)	$\{s_5, s_2\}$	s_5 is a partner of s_1 and s_2 is a partner of s_3
(s_1, s_4)	$\{s_5, s_3\}$	s_5 is a partner of s_1 and s_3 is a partner of s_4
(s_1, s_5)	$\{s_3, s_4\}$	s_3 is a partner of s_1 and s_4 is a partner of s_5
(s_2, s_4)	$\{s_1, s_3\}$	s_1 is a partner of s_2 and s_3 is a partner of s_4
(s_2, s_5)	$\{s_1, s_4\}$	s_1 is a partner of s_2 and s_4 is a partner of s_5
(s_3, s_5)	$\{s_2, s_4\}$	s_2 is a partner of s_3 and s_4 is a partner of s_5

Table 7: An example of selecting cells and partners.

Based on the concept of the partner set, if we want to evaluate the number of transitions for a given k-cell substate vector $\mathbf{w}$ of a state vector $\mathbf{s} = (s_1, s_2, \ldots, s_n)$ for an n-cell LFSM with a maximum length cycle, we only need to construct a partner set $ps(\mathbf{w})$, and then evaluate the number of transitions 2^{k+r}, where $r = |ps(\mathbf{w})|$. Since the LHCA and LFSR have straightforward next state functions, it is easy to determine the partner set for the given substate vector $\mathbf{w}$ for them. As a result, we can give general formulas for LHCA and LFSR.

5.3 Transition Properties for LHCA and LFSR

On the basis of the partner set, we can easily derive the number of different k-cell substate vectors, which have 2^{2k} transition capability, with $k \leq \lfloor n/2 \rfloor$ for the LHCA and LFSR. Here we state the two key results for the LHCA and the simplified results for the LFSR. The comprehensive results and the proofs of all theorems can be found in [36]

Consider an n-cell LHCA with a maximum length cycle. Let $f_1(n)$ be the total number of distinct k-cell substate vectors which produce 2^{2k} transitions with $k = \lfloor n/2 \rfloor$ for the LHCA. We have

$$f_1(n) = \begin{cases} 2f_1(n-2), & \text{even } n \text{ and } n \geq 4, \\ 2f_1(n-2) + f_1(n-3), & \text{odd } n \text{ and } n \geq 5, \end{cases} \quad (1)$$
$$f_1(1) = 1, \quad f_1(2) = 2, \quad f_1(3) = 3.$$

The key to the derivation is to determine the number of selections for $\mathbf{w}$ such that each element in $\mathbf{w}$ has its own partner. For $n = 1, 2$, and 3, it is obvious.

When n is even and $n \geq 4$, if we select k cells, each of the rest of the k cells must be used as a partner of an element in $\mathbf{w}$. When n is odd and $n \geq 5$, if we select k cells for $\mathbf{w}$, we need k cells as partners of $\mathbf{w}$ and have one cell which is neither included in $\mathbf{w}$ nor a partner.

It is easy to solve the recurrence relation in equation (1) as follows.

$$f_1(n) = \begin{cases} 2^{\lfloor n/2 \rfloor}, & \textit{even } n, \\ (\lfloor n/2 \rfloor + 2)2^{(n-3)/2}, & \textit{odd } n. \end{cases}$$

For the general case, we have the following result. Let $f_2(n,k)$ be the total number of distinct k-cell substate vectors which produce 2^{2k} transitions with $k \leq \lfloor n/2 \rfloor$ for an n-cell LHCA with a maximum length cycle. We have

$$\begin{aligned} f_2(n,k) &= \begin{cases} 2f_2(n-2,k-1) + f_2(n-1,k) - f_2(n-3,k-1), & k < \lfloor n/2 \rfloor, \\ f_1(n), & k = \lfloor n/2 \rfloor, \end{cases} \\ f_2(n,1) &= n. \end{aligned}$$

In a similar fashion, we can derive the numbers for the LFSR. Table 8 gives the general formulae of the total number of distinct k-cell substate vectors achieving 2^{2k} transitions for the n-cell LHCA and LFSR, where $k = \lfloor n/2 \rfloor$. Note that the number of selections for the LFSR is the upper bound because we have no general formula for any LFSR with a maximum length cycle.

LFSM	*Even n*	*Odd n*
LHCA	$2^{n/2}$	$(\lfloor n/2 \rfloor + 2)2^{(n-3)/2}$
LFSR(I)†	$n/2+1$	$(n^2+3)/4$
LFSR(II)†	$n/2+1$	$(n+3)(n+1)/8$

†The selections are upper bounds.

Table 8: General formulae for the number of selections for LHCA and LFSR with $k = \lfloor n/2 \rfloor$.

n	$k=\lfloor n/2 \rfloor$	*LHCA*	*LFSR(I)*†	*LFSR(II)*†	n	$k=\lfloor n/2 \rfloor$	*LHCA*	*LFSR(I)*†	*LFSR(II)*†
5	2	8	7	6	6	3	8	4	4
7	3	20	13	10	8	4	16	5	5
9	4	48	21	15	10	5	32	6	6
11	5	112	31	21	12	6	64	7	7
13	6	256	43	28	14	7	128	8	8
15	7	576	57	36	16	8	256	9	9
17	8	1280	73	45	18	9	512	10	10
19	9	2816	91	55	20	10	1024	11	11
21	10	6144	111	66	22	11	2048	12	12
23	11	13312	133	78	24	12	4096	13	13
25	12	28672	157	91	26	13	8192	14	14
27	13	61440	183	105	28	14	16384	15	15
29	14	131072	211	120	30	15	32768	16	16
31	15	278528	241	136	32	16	65536	17	17
33	16	589824	273	153	34	17	131072	18	18
35	17	1245184	307	171	36	18	262144	19	19
37	18	2621440	343	190	38	19	524288	20	20
39	19	5505024	381	210	40	20	1048576	21	21
41	20	11534336	421	231	42	21	2097152	22	22
43	21	24117248	463	253	44	22	4194304	23	23
45	22	50331648	507	276	46	23	8388608	24	24
47	23	104857600	553	300	48	24	16777216	25	25
49	24	218103808	601	325	50	25	33554432	26	26

† The numbers are upper bounds on the number of selections because there is no general equation for all LFSR concerning the number of different selections.

Table 9: Number of selections for the k-cell substate vector with 2^{2k} transition capability.

In Table 9, we list the number of different k-cell substate vectors, which have 2^{2k} transition capability, with $k = \lfloor n/2 \rfloor$ for the LHCA and LFSR from degree 5 to 50. It can be seen that for the LHCA, the number of possible selections increases exponentially while for the LFSR the number increases much more slowly. It is very clear from the theoretical results that the LHCA have a much higher transition space than the LFSRs, and should consequently perform much better as generators for stimulating faults requiring a pair of vectors. To investigate whether this is true in practice, we consider experiments with the ISCAS85 benchmark circuits in the next section.

6 Applications in Built-In Self-Test

Built-in self-test(BIST) refers to those techniques where additional hardware is added to a design so that testing is accomplished without the need for external special purpose testing hardware. A widely accepted approach to BIST is to use a *pseudorandom* vector generator and a data compactor. The generator produces the test vectors to be applied to a circuit under test and the compactor reduces the response to these vectors to a single value (*e.g.,* 16 or 32 bits) known as the signature.

LFSR are the predominant choice in the BIST literature, both for the test vector generator and for the data compactor[3]. Recently, alternative test vector generators based on LHCA have been considered and shown to be superior to the LFSR based generators, especially, for stimulating faults with *sequential behavior*. LHCA have also been proposed as the data compactor in BIST. We examine practical issues concerning the use of the LHCA, comparing with the LFSR.

6.1 Test Vector Generators

In BIST, to determine whether a given test vector generator is good, a practical measure is to see how many faults considered can be *stimulated* when the given test vector is applied to a circuit under test. In general, the LFSR and LHCA test vector generators offer virtually identical single *stuck-at* fault coverage. However, the LHCA have substantial promise in BIST particularly for the more complex fault models.

We review two classes of fault models, *stuck-at* and *delay* faults, and detection requirements for these models. This is useful in understanding the importance of the transition property for the test vector generator discussed in the previous section. Then, we provide empirical comparisons to show that our analysis of the transition property presented in the previous section is a reasonable metric of the effectiveness of the test vector generator.

6.1.1 Fault Models and Detection Requirements

Fault models allow us to define the types of faults considered and their behavior. In addition, fault models allow us to represent the behavior of physical failures/defects. Fault models attempt to cover the types of faults that can occur although they are not completely accurate in practice.

A single *stuck-at* fault assumes a circuit failure corresponds to one line of the circuit being permanently fixed at 0 or at 1. A circuit with p lines has $2p$ possible single *stuck-at* faults. We do simple fault collapsing on single gates, *e.g.*, an input of an AND gate *stuck-at* 0 fault is equivalent to the output of the gate *stuck-at* 0 fault.

A delay test of a combinational circuit in a clocked environment is defined to be a test of the ability of the combinational logic to propagate data in time for clocking into the next stage of latches [27]. Two different delay fault models, called the *gate delay fault* model and *path delay fault* model, respectively, have been proposed and are frequently used. We are concentrating on the *gate delay fault* model in this paper.

The *delay* faults considered are termed *slow-to-rise* and *slow-to-fall* faults. Let $c_1, c_2, ..., c_q$ be the correct (*expected*) bit sequence for a line of a circuit over some time period. The *delay* faults yield the sequence $d_i, 0 < i \leq q$, defined as follows:

(a) *slow-to-rise*

$$d_i = \begin{cases} 1, & c_{i-1} = 1 \text{ and } c_i = 1 \\ 0, & \text{otherwise} \end{cases}$$

(b) *slow-to-fall*

$$d_i = \begin{cases} 0, & c_{i-1} = 0 \text{ and } c_i = 0 \\ 1, & \text{otherwise} \end{cases}$$

where $c_0 = c_1$. A circuit with p lines has $2p$ potential single *delay* faults. *Delay* fault equivalence rules have been partly considered in [32]. We remove *slow-to-rise* and *slow-to-fall* faults for the NOT gate output and also remove *slow-to-rise* faults for NOR and AND gate outputs and *slow-to-fall* faults for NAND and OR gate outputs since these can be shown to be equivalent to input *delay* faults.

Delay faults cause combinational circuits to behave sequentially, and they can not be modeled as classical (*e.g., stuck-at*) faults. Hence, they are called *sequential* faults.

A single *delay* fault on a given line g requires a pair of vectors for detection; the first to 'set-up' the value, and the second to 'propagate' the fault effect to a circuit output. When a pair of vectors is applied, it produces one of four possible different sequences on the line g: 00, 01, 10, and 11. The sequences 01 and 10 are used to detect *slow-to-rise* and *slow-to-fall* faults on line g since *slow-to-rise* changes 01 to 00 and *slow-to-fall* changes 10 to 11 on the given line g. Note that only the second vector is used to propagate the fault effect to the circuit output because the fault does not change the first value produced by the first vector.

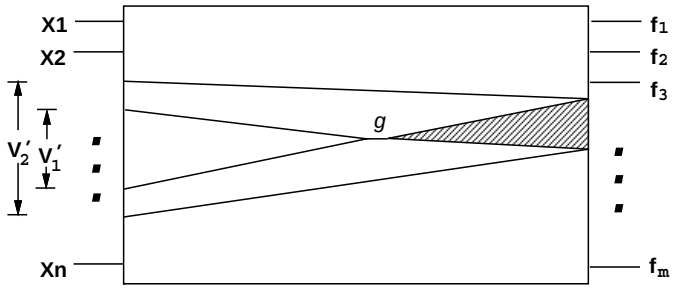


Figure 8: Simplified example for detecting delay fault.

Consider a circuit illustrated diagrammatically in Figure 8, where we are interested in detecting a *slow-to-rise* fault on line g. First we need vector $\mathbf{V_1}$ to set 0 on line g, and then use vector $\mathbf{V_2}$ to propagate the effect of the fault through the shadow area to at least one of the m outputs. This vector $\mathbf{V_2}$, of course, also detects a *stuck-at* 0 fault on line g. However, while both $\mathbf{V_1}$ and $\mathbf{V_2}$ are n-cell vectors, that is, they assign values to all the input variables, in general not all of those inputs are relevant (or even connected) to g. That is, only a subvector $\mathbf{V_1'}$ of $\mathbf{V_1}$ and a subvector $\mathbf{V_2'}$ of $\mathbf{V_2}$ are actually used (it is obvious that $|\mathbf{V_1'}| \leq |\mathbf{V_2'}|$). From the above discussion, we can see that the transition property presented in the previous section is an appropriate metric for comparing generators when stimulating faults requiring a pair of vectors.

6.1.2 Some Experiments and Discussions

Given that an LHCA has a much larger number of distinct k-cell substate vectors, $k \leq \lfloor n/2 \rfloor$, which have 2^{2k} transition capability, than an LFSR, we would expect a higher fault coverage. We use simulation experiments on the ISCAS85 benchmark circuits in [5] to observe the effectiveness of test vectors generated by the LHCA and LFSR. The experiments are based on gate *delay*, *i.e., slow-to-rise* and *slow-to-fall*, fault simulation [32]. Comprehensive *delay* fault equivalence rules are applied. A fault is detected if the output data stream in the presence of the fault differs from the

output data stream in the fault-free case. All redundancies in the circuits have been removed by the approach in [31], *i.e.,* there are no untestable *delay* faults in the circuits since all *stuck-at* faults are testable.

Circuit	*Number*	*Undetected Faults for LHCA*				*Undetected Faults for LFSR(II)*			
Name	*of Faults*	*best* (times)	*worst* (times)	*median*	*average*	*best* (times)	*worst* (times)	*median*	*average*
C432nr	658	0 (73)	4 (2)	0	0.38	0 (2)	16 (1)	5	5.25
C499nr	844	0 (99)	1 (1)	0	0.01	0 (67)	2 (2)	0	0.35
C880	1305	0 (56)	5 (3)	0	0.74	1 (1)	30 (1)	12	13.61
C1355nr	2076	14 (1)	28 (2)	22	22.18	17 (1)	36 (1)	26	26.02
C1908nr	2468	0 (71)	18 (1)	0	0.99	1 (3)	35 (1)	11	11.81
C2670nr	2532	54 (2)	119 (2)	85	84.91	115 (1)	191 (1)	146	145.16
C3540nr	4290	0 (27)	155 (1)	2	10.51	21 (1)	250 (1)	70	88.01
C5315nr	7222	0 (91)	3 (1)	0	0.12	4 (2)	73 (1)	14	17.48
C6288nr	9987	0 (40)	11 (1)	1	1.21	9 (1)	28 (1)	18	17.82
C7552nr	9104	85 (2)	132 (1)	109	109.01	65 (1)	115 (1)	91	90.82

Table 10: A summary of *delay* fault simulation results for 100 different random connections.

The simulation results for *delay* faults are shown in Table 10. The results are collected by performing the fault simulation using test sequences of length 102,000. We simulated each circuit by using the LHCA ([37, 9]) and LFSR ([3]) test vector generators with degree corresponding to the number of the circuit inputs. For each circuit, we used 100 different random connections between the circuit inputs and the test vector generator outputs. In the table, we report the best (and the number of times the best result is achieved in 100 random connections), worst (and the number of times the worst result is achieved in 100 random connections), as well as the median and average results with respect to the number of undetected faults. It can be seen that the LHCA have better fault coverage than the LFSRs and have greater chances to cover all faults considered. For example, for the LHCA, more than half of the 100 connections can detect all faults for circuit $C880$, whereas for the LFSR, none of the 100 connections can achieve 100% fault coverage for circuit $C880$. Hence, the simulation results provide evidence that our analysis of transition properties presented in the preceding section is indeed a proper metric of the effectiveness of LHCA and LFSR test vector generators.

From our analysis of the transition properties and the empirical results, it can be seen that an LHCA has more potential to achieve a higher fault coverage for stimulating faults requiring a pair of vectors. However, an LFSR still has the possibility of catching all faults considered, and infrequently, it performs better, *e.g.,* for circuit $C7552nr$ (with 207 inputs) in Table 10, resulting from the circuit topology and only a portion of the total cycle being generated. Thus, specific knowledge of a circuit under test is useful while choosing a test vector generator.

6.2 Data Compactors

When we apply a particular test vector sequence to a circuit, we can get an output sequence produced by the circuit. To determine whether a circuit under test is correct, an easy way is to check whether the output sequence produced by the circuit is the same as that produced by the *fault-free* circuit, when the same test vector sequence is applied. An advantage of this approach is that any fault can be detected as long as it can be stimulated by the test vector sequence applied. However, such an approach needs too much extra hardware (*e.g.,* ROM) to store the "gold" output sequence. An alternative way to store the "gold" output sequence is compaction of the "gold" sequence into a relatively short binary sequence called a *signature*.

In general, an LFSR is used as a compactor to produce the signature [3]. Corresponding to a circuit with a single output and a circuit with multiple outputs, there are a single-input LFSR compactor and a multiple-input LFSR compactor, simply called MISR, respectively. Similarly, for the LHCA, we have the single-input LHCA compactor and the multiple-input LHCA compactor,

called MICA, respectively. Since the compactor has a fixed length (*i.e,.* the number of bits in the LF/MISR or LH/MICA), the signature produced by the compactor for a faulty circuit may be the same as the "gold" signature, *i.e., aliasing* is possible. It has been conjectured that a k-bit signature register will exhibit a probability of aliasing on the order of $1/2^k$. Several papers have appeared showing this to be the case under various error assumptions. The most general result to date is due to Kameda, Pilarski, and Ivanov [16]. They show that the probability of aliasing is asymptotic to $1/2^k$ for a very general fault model. Their result is applicable to any linear data compactor including an MISR and an MICA.

In our experiments with the ISCAS85 circuits[5] and others we find that either no aliasing or that aliasing is indeed bounded by $1/2^k$. Aliasing is found to be less likely as the number of circuit outputs increases. This is not surprising, since in this case, 2^k is much greater than the size of the fault set. Care must be taken with this observation since the ISCAS85 benchmark circuits are rather small. However, the empirical evidence indicates that aliasing behaves as predicted in the theoretical work.

6.3 Remarks

The most important observation from our experiments is that, while we noted significantly different performance of LHCA and LFSR test vector generators in BIST with respect to sequential faults, we found that no significant difference in the aliasing behavior of MISR or MICA data compactors, regardless of the fault model. Our experiments thus indicate that fault coverage is a more significant issue than aliasing in BIST and that LHCAs provide better coverage if sequential faults are of concern.

7 Conclusions

We have outlined a solution to the open problem as to whether there exists an LHCA for any given polynomial and how to find it. In particular, we have presented an elegant synthesis algorithm that can easily produce the corresponding LHCA given any irreducible polynomial for all practical orders. The similarity transform between the transition matrices of LHCA and LFSRs is an important concept since it allows us to systematically analyze LHCA making use of the depth of knowledge that exists about LFSRs. This approach has been used quite successfully in the VLSI test domain and is of course well known in the area of coding.

The analysis of partitioning and concatenation provides an overview of how LHCAs (or LFSRs) can be easily split or combined without loss of the maximum length cycle property. It is instructive to have such knowledge when designing a circuit using LHCA or LFSR as testing stimulus sources. The designer has considerable freedom in the choice of partition or concatenation. In this paper, we only listed a few. Hopefully, these will not limit designers' creativity in using the concepts of the partitioning and concatenation.

From our experimental results on the ISCAS85 benchmark circuits, we saw that, in general, the LHCA based test vector generator provides better fault coverage than the LFSR based generator for *sequential-type* faults, *e.g., delay* and *transistor stuck-open* faults. Our analysis of transition properties brings a general theoretical answer to the question as to why LHCA are better than LFSR as BIST test vector generators for *sequential-type* faults. Our work on the choice of substate vectors for LHCA or LFSR to achieve maximum transition count provides a necessary information to determine the appropriate connection of the circuit inputs to the LFSM generator outputs in order to maximize the fault coverage for *sequential* faults. The work of determining the proper connection is a promising area for further research.

However, when doing this, we must understand that any method proposed which works in principle for a set of small circuits does not necessarily mean that it works for the general case.

Perhaps the choice based on random connections and simulation can easily achieve the expected result (at least our preliminary studies indicate this to be so).

The main purpose of this paper has been to review the current state of knowledge regarding LHCA and to relate same to the better known and, at least to date, more extensively studies LFSR. We are not proposing that LHCA replace LFSRs, but rather suggest them as an alternative that should be considered, with evidence that, at least in certain applications, there is some benefit to be gained.

References

[1] P. Bardell, "Analysis of cellular automata used as pseudorandom pattern generators," in *Proceedings of IEEE International Test Conference*, 1990, pp. 762–767.

[2] P. Bardell, "Discrete logarithms: a parallel pseudorandom pattern generator analysis method," *Journal of Electronic Testing: Theory and Applications*, vol. 3, no. 1, pp. 17–31, 1992.

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

[4] D. Bhavsar, "Concatenable polydividers: bit-sliced LFSR chips for board self-test," in *Proceedings of IEEE International Test Conference*, 1985, pp. 88–93.

[5] F. Brglez and H. Fujiwara, "A neutral netlist of 10 combinational benchmark circuits and a target translator in FORTRAN," in *Proceedings of IEEE International Symposium on Circuits and Systems*, 1985, pp. 663–698.

[6] K. Cattell, "Characteristic polynomials of one-dimensional linear hybrid cellular automata," *Ph.D. Dissertation, Department of Computer Science, University of Victoria, Victoria, BC, Canada*, May 1995.

[7] K. Cattell and J. Muzio, "Table of linear cellular automata for minimal weight primitive polynomials of degree up to 300," Department of Computer Science, University of Victoria, Victoria, BC, Canada, Tech. Rep. DCS-163-IR, June 1991.

[8] K. Cattell and J. Muzio, "Synthesis of one-dimensional linear hybrid cellular automata," *To appear in IEEE Transactions on Computer-Aided Design*, 1996.

[9] K. Cattell and S. Zhang, "Minimal cost one-dimensional linear hybrid cellular automata of degree through 500," *Journal of Electronic Testing: Theory and Applications*, vol. 6, no. 2, pp. 255–258, April 1995.

[10] T. Cormen, C. Leiserson, and R. Rivest, *Introduction to Algorithms*. The MIT Press, 1990.

[11] A. Das, A. Ganguly, A. Dasgupta, S. Bhawmik, and P. Chaudhuri, "Efficient characterisation of cellular automata," *IEE Proceedings – Computers and Digital Techniques*, vol. 137, no. 1, pp. 81–87, January 1990.

[12] K. Furuya and E. McCluskey, "Two-pattern test capabilities of autonomous TPG circuits," in *Proceedings of IEEE International Test Conference*, 1991, pp. 704–711.

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

[14] P. Hortensius, R. McLeod, and H. Card, "Parallel random number generation for VLSI systems using cellular automata," *IEEE Transactions on Computers*, vol. 38, no. 10, pp. 1466–1472, October 1989.

[15] P. Hortensius, R. McLeod, W. Pries, D. Miller, and H. Card, "Cellular automata-based pseudorandom number generators for built-in self-test," *IEEE Transactions on Computer-Aided Design*, vol. 8, no. 8, pp. 842–859, August 1989.

[16] T. Kameda, S. Pilarski, and A. Ivanov, "Notes on multiple input signature analysis," Simon Fraser University, Tech. Rep. CSS/LCCR TR90-16, 1990.

[17] K. Knuth, *The Art of Computer Programming: Seminumerical Algorithms.* Addison-Wesley, Reading, Mass., 1981.

[18] E. Kontopidi and J. Muzio, "The partitioning of linear registers for testing applications," *Microelectronics Journal*, vol. 24, pp. 533–546, 1993.

[19] R. Lidl and H. Niederreiter, *Introduction to Finite Fields and Their Applications.* Cambridge: Cambridge University Press, 1986.

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

[21] M. Martinez and S. Bracho, "Concatenated LFSR makes a weighted built-in logic block observation," in *Proceedings of Conference on Circuits and Computers*, 1993, pp. 113–121.

[22] J. Mesirov and M. Sweet, "Continued fraction expansions of rational expressions with irreducible denominators in characteristic 2," *Journal of Number Theory*, vol. 27, pp. 144–148, 1987.

[23] S. Nandi, B. Vamsi, S. Chakraborty, and P. Chaudhuri, "Cellular automata as a BIST structure for testing CMOS circuits," *IEE Proceedings – Computers and Digital Techniques*, vol. 141, no. 1, pp. 41–47, January 1994.

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

[25] M. Serra and T. Slater, "A Lanczos algorithm in a finite field and its application," *Journal of Combinatorial Mathematics and Combinatorial Computing*, vol. 7, pp. 11–32, April 1990.

[26] M. Serra, T. Slater, J. Muzio, and D. Miller, "The analysis of one-dimensional linear cellular automata and their aliasing properties," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 9, no. 7, pp. 767–778, July 1990.

[27] G. Smith, "Model for delay faults based upon paths," in *Proceedings of IEEE International Test Conference*, 1985, pp. 342–349.

[28] H. Stone, *Discrete Mathematics Structures and Their Applications.* Science Research Associates Inc., 1973.

[29] X. Sun, E. Kontopidi, M. Serra, and J. C. Muzio, "The concatenation and partitioning of linear finite state machines," *International Journal of Electronics*, vol. 78, no. 5, pp. 809–839, 1995.

[30] X. Sun and M. Serra, "Merging concurrent checking and off-line BIST," in *Proceedings of IEEE International Test Conference*, September 1992, pp. 958–967.

[31] G. Tromp and A. van de Goor, "Logic synthesis of 100-percent testable logic networks," in *Proceedings of IEEE International Conference on Computer Design*, 1991, pp. 428–431.

[32] J. Waicukauski, E. Lindbloom, B. Rosen, and V. Iyengar, "Transition fault simulation," *IEEE Design and Test of Computers*, pp. 32–38, April 1987.

[33] S. Wolfram, "Random sequence generation by cellular automata," *Advances in Applied Mathematics*, vol. 7, pp. 123–169, 1986.

[34] S. Zhang, R. Byrne, and D. Miller, "BIST generators for sequential faults," in *Proceedings of IEEE International Conference on Computer Design*, October 1992, pp. 260–263.

[35] S. Zhang, R. Byrne, J. Muzio, and D. Miller, "Why cellular automata are better than LFSRs as Built-In Self-Test generators for sequential-type faults," in *Proceedings of IEEE International Symposium on Circuits and Systems*, May 1994, pp. 69–72.

[36] S. Zhang, R. Byrne, J. Muzio, and D. Miller, "Quantitative analysis for linear hybrid cellular automata and LFSR as Built-In Self-Test generators for sequential faults," *Journal of Electronic Testing: Theory and Applications*, vol. 7, no. 3, pp. 209–221, December 1995.

[37] S. Zhang, D. Miller, and J. Muzio, "Determination of minimal cost one-dimensional linear hybrid cellular automata," *IEE Electronics Letters*, vol. 27, no. 18, pp. 1625–1627, August 1991.