High-Speed Parallel Architectures for Linear Feedback Shift Registers

Manohar Ayinala, *Student Member, IEEE*, and Keshab K. Parhi, *Fellow, IEEE*

***Abstract*—Linear feedback shift register (LFSR) is an important component of the cyclic redundancy check (CRC) operations and BCH encoders. The contribution of this paper is two fold. First, this paper presents a mathematical proof of existence of a linear transformation to transform LFSR circuits into equivalent state space formulations. This transformation achieves a full speed-up compared to the serial architecture at the cost of an increase in hardware overhead. This method applies to all generator polynomials used in CRC operations and BCH encoders. Second, a new formulation is proposed to modify the LFSR into the form of an infinite impulse response (IIR) filter. We propose a novel high speed parallel LFSR architecture based on parallel IIR filter design, pipelining and retiming algorithms. The advantage of the proposed approach over the previous architectures is that it has both feedforward and feedback paths. We further propose to apply combined parallel and pipelining techniques to eliminate the fanout effect in long generator polynomials. The proposed scheme can be applied to any generator polynomial, i.e., any LFSR in general. The proposed parallel architecture achieves better area-time product compared to the previous designs.**

***Index Terms*—BCH, cyclic redundancy check (CRC), linear feedback shift register (LFSR), look-ahead computation, parallel processing, pipelining, state space, transformation.**

I. Introduction

LINEAR Feedback Shift Registers (LFSR) are widely used in BCH encoders and CRC operations [1]–[3]. A sequential LFSR circuit cannot meet the speed requirement when high-speed data transmission is required. Because of this limitation, parallel architectures must be employed in high-speed applications such as optical communication systems where throughput of several Gigabit/s is required. LFSRs are also used in conventional Design for Test (DFT) and Built-in Self-Test (BIST) [4]. LFSRs are used to carry out response compression in BIST, while for the DFT, it is a source of pseudorandom binary test sequences.

CRC is used in almost all communication protocols as an efficient way to detect transmission errors and BCH codes are among the most extensively used codes in modern communication systems. A specialized hardware implementation can make use of higher speed clock to meet the real-time constraints. Thus in order to meet the increasing demand on processing capabilities, much research has been carried out on parallel architectures of LFSR for BCH and CRC encoders.

Many parallel architectures have been proposed in the literature for BCH and CRC encoders to increase the throughput [5]–[23]. In [5] and [6], parallel CRC implementations have been proposed based on mathematical deduction. In these papers, recursive formulations were used to derive parallel CRC architectures. High-speed architectures for BCH encoders have been proposed in [6] and [12]. These are based on multiplication and division computations on generator polynomials and can be used for any LFSR of any generator polynomial. They are efficient in terms of speeding up the LFSR but their hardware cost is high. A method to reduce the worst case delay is discussed in [14]. In [14], CRC is computed using shorter polynomials that has fewer feedback terms. Unfolding based parallel CRC implementations are proposed in [7]. Look-ahead computations are used to reduce the iteration bound before unfolding is applied. The parallel processing leads to a long critical path even though it increases the number of message bits processed. Further many approaches have been presented in the literature for software based CRC computation [24], [25].

In general, parallel architectures for LFSR proposed in the literature lead to an increase in the critical path. The longer the critical path, the speed of operation of the circuit decreases; thus the throughput rate achieved by parallel processing will be reduced. Since these parallel architectures consist of feedback loops, pipelining technique cannot be applied to reduce the critical path. Another issue with the parallel architectures is the hardware cost. A novel parallel CRC architecture based on state space representation is proposed in [9]. The main advantage of this architecture is that the complexity is shifted out of the feedback loop. The full speed-up can be achieved by pipelining the feedforward paths. A state space transformation has been proposed in [9] to reduce complexity but the existence of such a transformation was not proved in [9] and whether such a transformation is unique has been unknown so far.

This paper makes two contributions. First, we present a mathematical proof to show that such a transformation exists for all CRC and BCH generator polynomials. We also show that this transformation is non-unique. In fact, we show the existence of many such transformations and how these can be derived. Second, we present a novel formulation of the LFSR in terms of an IIR filter. We then propose novel schemes based on pipelining, retiming, and look ahead computations to reduce the critical path in the parallel architectures based on parallel and pipelined IIR filter design [26]. The proposed IIR filter based parallel architectures have both feedback and feedforward paths,

Manuscript received July 16, 2010; revised December 30, 2010 and December 30, 2010; accepted May 19, 2011. Date of publication June 13, 2011; date of current version August 10, 2011. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Ut-Va Koc.

The authors are with the Department of Electrical and Computer Engineering, University of Minnesota, Minneapolis, MN 55455 USA (e-mail: ayina004@umn.edu; parhi@umn.edu).

Digital Object Identifier 10.1109/TSP.2011.2159495

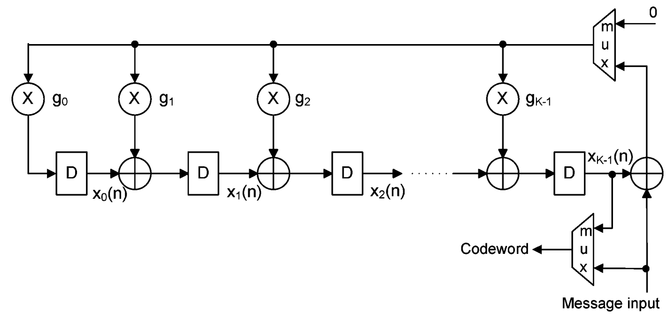

Fig. 1. Basic LFSR architecture.

and pipelining can be applied to further reduce the critical path. We show that the proposed architecture can achieve a critical path similar to previous designs with less hardware overhead. Without loss of generality, only binary codes are considered. This paper is an expanded version of [27].

This paper is organized as follows. Section II briefly reviews the LFSR architecture and the existing architectures and their effects on critical path and hardware cost. In Section III, a state space based representation of LFSR is described and the transformation for parallel LFSR is presented in Section IV. In Section V, our proposed IIR filter representation of LFSR is discussed and parallel architectures for this representation are presented in Section VI. Section VII shows how pipelining and retiming can be used to further reduce the critical path. The proposed design is compared with previous designs in Section VIII and some conclusions are drawn in Section IX.

II. LFSR Architecture

A basic LFSR architecture for K^{th} order generating polynomial in $GF(2)$ is shown in Fig. 1. K denotes the length of the LFSR, i.e., the number of delay elements and $g_0, g_1, g_2, \ldots, g_K$ represent the coefficients of the characteristic polynomial. The characteristic polynomial of this LFSR is

$$g(x) = g_0 + g_1 x + g_2 x^2 + \cdots + g_K x^K$$

where $g_0, g_1, g_2, \ldots, g_K \in GF(2)$. For CRC and BCH generator polynomials, $g_K = g_0 = 1$. In $GF(2)$, the sum of two elements is implemented using a two-input XOR gate and the multiplier elements are either open circuits or short circuits, i.e., $g_i = 1$ implies that a connection exists. On the other hand $g_i = 0$ implies that no connection exists and the corresponding XOR gate can be replaced by a direct connection from input to output.

Let $u(x) = \sum_{n=0}^{N-1} u(n)x^n, u(n) \in GF(2)$, for $n = 0, 1, \ldots, N-1$, represent an input sequence of length N. Both CRC computation and BCH encoding involve the division of the polynomial $u(x)x^K$ by $g(x)$ to obtain the remainder $Rem(u(x)x^K)_{g(x)}$. During the first N clock cycles, the N-bit message is input to the LFSR with most significant bit (MSB) first. At the same time, the message bits are also sent to the output to form the BCH encoded codeword. After N clock cycles, the feedback is reset to zero and the K registers contain the coefficients of $Rem(u(x)x^K)_{g(x)}$. In BCH encoding, the remaining bits are then shifted out bit by bit to form the remaining systematic codeword bits.

The throughput of the system is limited by the serial computation of the LFSR. We can increase the throughput by modifying the system to process some number of bits in parallel. In [15], first serial to parallel transformation of linear feedback shift register was described and was first applied to CRC computation in [16]. Several other approaches have been recently presented to parallelize LFSR computations. We review some of these approaches here.

In [5], [6], and [8], parallel CRC implementations have been proposed based on mathematical deduction. In these papers, recursive formulations were used to derive parallel CRC architectures. In [9], [21] and [22] parallel architectures for CRC based on state space representation are described. A state space transformation is proposed to reduce the complexity and an exhaustive search to find the optimal transformation matrix is presented in [22]. As it involves two matrix multiplication computations, the hardware complexity will be high. The state space approach is presented in detail in Section III. References [17] and [18] describe parallel architectures based on Chinese remainder theorem (CRT) for long BCH encoding. The critical path of the architecture in [17] is proportional to $\log_2 K$, where K is the length of the BCH code, which will be an issue in long BCH codes. In [14], [19], and [23], a two step approach is proposed to realize the CRC computation in hardware. A full programmable CRC architecture is derived in [20] based on the algorithm in [5].

High-speed architectures for BCH encoders have been proposed in [6] and [12]. These are based on multiplication and division computations on generator polynomials and can be used for any LFSR of any generator polynomial. They are efficient in terms of speeding up the LFSR but their hardware cost is high. Further, unfolding [28] technique has been used in [6], [7], [13] to implement parallel LFSR architectures. When we unfold the LFSR architectures directly, it may lead to a parallel circuit with a long critical path. Look-ahead pipelining technique is used in [7] to reduce the iteration bound in the LFSR. The LFSR architectures can also face fanout issues due to the large number of nonzero coefficients especially in longer generator polynomials. In [6] and [17] new approaches have been proposed to eliminate the fan-out bottleneck by modifying the generator polynomial. A two step approach is proposed to reduce the effect of fanout in [13].

In the next two sections, we describe the state space based representation of LFSR and prove that a transformation exists to transform the feedback loop of parallel LFSR into a simple LFSR.

III. State Space Representation of LFSR

A parallel LFSR architecture based on state space computation has been proposed in [9]. The LFSR shown in Fig. 1 can be described by

$$\mathbf{x}(n+1) = \mathbf{A}\mathbf{x}(n) + \mathbf{b}u(n); \quad n \geq 0 \qquad (1)$$

with the initial state $\mathbf{x}(0) = \mathbf{x}_o$. The K-dimensional state vector $\mathbf{x}(n)$ is given by

$$\mathbf{x}(n) = [x_0(n) \quad x_1(n) \quad \ldots \quad x_{K-1}(n)]^T$$

and $\mathbf{A}$ is the $K \times K$ matrix given by

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 & \dots & 0 & g_0 \\ 1 & 0 & 0 & \dots & 0 & g_1 \\ 0 & 1 & 0 & \dots & 0 & g_2 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 1 & g_{K-1} \end{bmatrix}. \qquad (2)$$

The $K \times 1$ matrix $\mathbf{b}$ is

$$\mathbf{b} = [g_0 \quad g_1 \quad \dots \quad g_{K-1}]^T.$$

The output of the system is the remainder of the polynomial division that it computes, which is the state vector itself. We call the output vector $\mathbf{y}(n)$ and add the output equation $\mathbf{y}(n) = \mathbf{C}\mathbf{x}(n)$ to the state equation in (1), with $\mathbf{C}$ equal to the $K \times K$ identity matrix. The coefficients of the generator polynomial $g(x)$ appear in the right-hand column of the matrix $\mathbf{A}$. Note that, this is the *companion matrix* of polynomial $g(x)$ and $g(x)$ is the *characteristic polynomial* of this matrix. The initial state $\mathbf{x}_o$ depends on the given application.

The L-parallel system can be derived so that L elements of the input sequence $u(n)$ are processed in parallel. Let the elements of $u(n)$ be grouped together, so that the input to the parallel system is a vector $\mathbf{u}_L(mL)$ where

$$\begin{aligned} \mathbf{u}_L(mL) = [&u(mL+L-1) \quad u(mL+L-2) \dots \\ &u(mL+1) \quad u(mL)]^T; \\ &m = 0, 1, \dots, \left(\frac{N}{L}\right) - 1 \end{aligned} \qquad (3)$$

where N is the length of the input sequence. Assume that N is an integral multiple of L. The state space equation can be written as

$$\begin{aligned} \mathbf{x}(mL+L) &= \mathbf{A}^L\mathbf{x}(mL) + \mathbf{B}_L\mathbf{u}_L(mL); \\ \mathbf{y}(mL) &= \mathbf{C}_L\mathbf{x}(mL) \end{aligned} \qquad (4)$$

where the index mL is incremented by L for each block of L input bits.

The matrix $\mathbf{B}_L$ is a $K \times L$ matrix given by

$$\mathbf{B}_L = [\mathbf{b} \quad \mathbf{A}\mathbf{b} \quad \mathbf{A}^2\mathbf{b} \quad \dots \quad \mathbf{A}^{L-1}\mathbf{b}]$$

and $\mathbf{C}_L = I$, the $K \times K$ identity matrix. The output vector $\mathbf{y}(mL)$ is equal to the state vector which has the remainder at $m = \frac{N}{L}$. Fig. 3 shows an L-parallel system which processes L-bits at a time. The issue in this system is that the delay in the feedback loop increases due to the complexity of $\mathbf{A}^L$.

We can compare the throughput of the original system and the modified L-parallel system described by (4) using the block diagrams shown in Figs. 2 and 3. The throughput of both systems is limited by the delay in the feedback loop. The main difference in the two systems is the computation of the products $\mathbf{A}\mathbf{x}$ and $\mathbf{A}^L\mathbf{x}$. We can observe from (2), that no row of $\mathbf{A}$ has more than two nonzero elements. Thus the time required for the computation of the product $\mathbf{A}\mathbf{x}$ in $GF(2)$ is the delay of a two input XOR gate. Similarly, the time required for the product $\mathbf{A}^L\mathbf{x}$ will depend on the number of nonzero elements in the matrix $\mathbf{A}^L$.

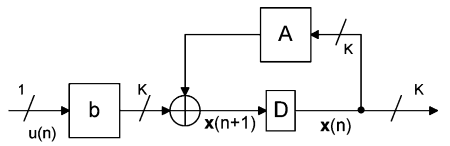

Fig. 2. Serial LFSR Architecture. The delay element represents K delays.

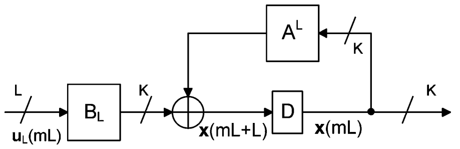

Fig. 3. L-parallel LFSR Architecture [9]. The delay element represents K delays.

Consider the standard 16-bit CRC with the generator polynomial

$$g(x) = x^{16} + x^{15} + x^2 + 1$$

with $L = 16$. The maximum number of nonzero elements in any row of $\mathbf{A}^{16}$ is 15, i.e., the time required to compute the product is the time to compute exclusive or of 15 terms. The detailed analysis for this example is presented in the Appendix. In a similar manner, the maximum number of nonzero entries within a row in $\mathbf{A}^{32}$ for the standard 32-bit CRC is 17. This computation is the bottleneck for achieving the required speed-up factor.

We can observe from Fig. 3 that a speed-up factor of L is possible if the block $\mathbf{A}^L$ can be replaced by a block whose circuit complexity is no greater than that of block $\mathbf{A}$. This is possible given the restrictions on the generator polynomial $g(x)$ that are always met for the CRC and BCH codes in general which are discussed in Section IV. The computation in the feedback loop can be simplified by applying a linear transformation to (4). The linear transformation shifts the complexity from the feedback loop to the blocks that are outside the loop. These blocks can be pipelined and their complexity will not affect the system throughput.

IV. State Space Transformation

A. Transformation

In general, $\mathbf{A}^L$ in (4) is not in the form of a companion matrix even though $\mathbf{A}$ is a companion matrix as shown in (2). It has been shown in [9] that a linear transformation $\mathbf{T}$ can be used to derive a transformed L-parallel state space description where $\mathbf{A}^L$ is in the form a companion matrix.

Consider the linear transformation of the state vector $\mathbf{x}(mL)$ through a constant non-singular matrix $\mathbf{T}$, i.e.

$$\mathbf{x}(mL) = \mathbf{T}\mathbf{x}_t(mL).$$

Given $\mathbf{T}$ and its inverse $(\mathbf{T}^{-1})$, we can express the state space (4) in terms of the transformed state vector $\mathbf{x}_t(mL)$, as follows:

$$\begin{aligned} \mathbf{x}_t(mL+L) = \mathbf{A}_{Lt}\mathbf{x}_t(mL) \\ +\mathbf{B}_{Lt}\mathbf{u}_L(mL); \mathbf{y}(mL) = \mathbf{C}_{Lt}\mathbf{x}_t(mL) \end{aligned}$$

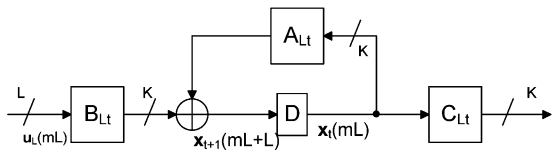

Fig. 4. Modified LFSR Architecture using state space transformation [9].

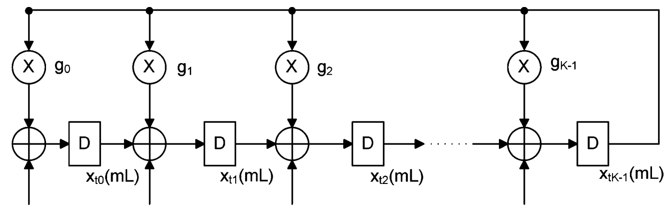

Fig. 5. Modified feedback loop of Fig. 4 [9].

where

$$\mathbf{A}_{Lt} = \mathbf{T}^{-1}\mathbf{A}^{L}\mathbf{T}; \mathbf{B}_{Lt} = \mathbf{T}^{-1}\mathbf{B}_{L}; \mathbf{C}_{Lt} = \mathbf{T}\mathbf{C}_{L} \quad (5)$$

and $\mathbf{T}$ is the transformation matrix. The parallel LFSR architecture after the transformation is shown in Fig. 4 and the modified feedback loop in Fig. 5. We can observe from the figure that if $\mathbf{A}_{Lt}$ is a companion matrix, then the complexity of the feedback loop will be same as that of the original LFSR. If there exists a $\mathbf{T}$ such that $\mathbf{A}_{Lt}$ is a companion matrix, then the complexity in the feedback loop can be reduced. It is evident from (5) that the relationship between $\mathbf{A}_{Lt}$ and $\mathbf{A}^{L}$ represents a similarity transformation.

Although a particular method of obtaining $\mathbf{T}$ was described in [9], it was not shown whether this $\mathbf{T}$ is unique. In addition, if $\mathbf{T}$ is nonunique, how one can generate an exhaustive list of all possible $\mathbf{T}$ was not known. In this paper, first we prove that a transformation exist for all CRC and BCH polynomials with distinct roots. Then we show that this $\mathbf{T}$ is not unique, and present how to construct different $\mathbf{T}$ matrices all of which guarantees $\mathbf{A}_{Lt}$ to be in the form of a companion matrix.

B. Existence of Transformation Matrix

Some of the definitions, results and theorems from [29] and [30] are presented here for completeness.

Theorem 1: If f is an irreducible polynomial in $GF(q)$ of degree k, then f has a root α in $GF(q^k)$. Furthermore, all the roots of f are simple and are given by the k distinct elements $\alpha, \alpha^{q}, \alpha^{q^2}, \ldots, \alpha^{q^{k-1}}$ of $GF(q^k)$.

The proof can be found in [29, p. 48].

BCH codes: The generator polynomial of the binary BCH code is specified in terms of its roots from the Galois field $GF(2^m)$, where $m(m \geq 3)$ is a positive integer. Let α be a primitive element in $GF(2^m)$. For all i, let $m_i(x)$ be the minimal polynomial of α^i with coefficients in $GF(2)$. The generator polynomial of the BCH code is defined as least common multiple (LCM) of $m_1(x), m_2(x), \ldots, m_{d-1}(x)$, that is

$$g(x) = LCM(m_1(x), m_2(x), \ldots, m_{d-1}(x))$$

where $2 \leq d \leq 2^m - 1$. $g(x)$ has $\alpha, \alpha^2, \alpha^4, \ldots, \alpha^{2d}$ and their conjugates as all its roots [30, p. 194]. Clearly, $g(x)$ has distinct roots in $GF(2^m)$ [29, p. 95].

CRC: The generator polynomial of CRC code is generated by either a primitive polynomial or a polynomial

$$g(x) = (x+1)p(x)$$

where $p(x)$ can be product of one or more primitive polynomials [31]. Any primitive polynomial $p(x)$ is irreducible and following theorem 1, $p(x)$ has distinct roots in its extension field. The root of $1 + x$ is 1 which shows that the generator polynomial $g(x)$ has distinct roots.

Theorem 2: Given any generator polynomial $g(x)$ with distinct roots and companion matrix $\mathbf{A}$, $\mathbf{A}^L$ is similar to a companion matrix, L is a positive integer.

Proof: A companion matrix is similar to a diagonal matrix if its characteristic polynomial has distinct roots. Since, the generator polynomial $g(x)$ has distinct roots in its extension field, we can diagonalize $\mathbf{A}$ as follows:

$$\mathbf{A} = \mathbf{V}\mathbf{D}\mathbf{V}^{-1} \quad (6)$$

where $\mathbf{D}$ is diagonal matrix with distinct roots of $g(x)$ as its elements and $\mathbf{V}$ is the Vandermonde matrix corresponding to the eigenvalues of $\mathbf{A}$. Now

$$\begin{aligned}\mathbf{A}^L &= (\mathbf{V}\mathbf{D}\mathbf{V}^{-1})^L \\ &= (\mathbf{V}\mathbf{D}\mathbf{V}^{-1})(\mathbf{V}\mathbf{D}\mathbf{V}^{-1})\ldots(\mathbf{V}\mathbf{D}\mathbf{V}^{-1}) \\ &= \mathbf{V}\mathbf{D}^L\mathbf{V}^{-1} \\ \Rightarrow \mathbf{V}^{-1}\mathbf{A}^L\mathbf{V} &= \mathbf{D}^L. \end{aligned} \quad (7)$$

Let $\lambda_1, \lambda_2, \ldots, \lambda_k$ be the distinct roots of the polynomial $g(x)$. Then

$$\mathbf{D}^L = \begin{bmatrix} \lambda_1^L & 0 & \ldots & 0 \\ 0 & \lambda_2^L & \ldots & 0 \\ \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & \ldots & \lambda_k^L \end{bmatrix}. \quad (8)$$

Since $\lambda_1, \lambda_2, \ldots, \lambda_k$ are distinct roots in the splitting field $GF(2^k)$, $\lambda_1^L, \lambda_2^L, \ldots, \lambda_k^L$ will be distinct in the same field. The characteristic polynomial of the $\mathbf{D}^L$ is

$$(x - \lambda_1^L)(x - \lambda_2^L)\ldots(x - \lambda_k^L) = 0.$$

It can be rewritten as

$$x^k + a_{k-1}x^{k-1} + \cdots + a_0$$

where $a_i, 0 \leq i \leq k-1$ are functions of λ's. Note that $a_0 = 1$.

Let $\mathbf{C}$ be the companion matrix of the above characteristic polynomial with distinct roots $\lambda_1^L, \lambda_2^L, \ldots, \lambda_k^L$. We can diagonalize $\mathbf{C}$ as follows, where $\mathbf{B}$ is the Vandermonde matrix corresponding to the eigenvalues of $\mathbf{C}$.

$$\begin{aligned} \mathbf{C} &= \mathbf{B}\mathbf{D}^L\mathbf{B}^{-1} \\ or \quad \mathbf{D}^L &= \mathbf{B}^{-1}\mathbf{C}\mathbf{B} \quad (9) \\ \therefore \quad \mathbf{V}^{-1}\mathbf{A}^L\mathbf{V} &= \mathbf{B}^{-1}\mathbf{C}\mathbf{B} \\ (\mathbf{B}\mathbf{V}^{-1})\mathbf{A}^L(\mathbf{V}\mathbf{B}^{-1}) &= \mathbf{C}. \quad (10) \end{aligned}$$

This shows that $\mathbf{A}^L$ is similar to a companion matrix. ■

C. Construction of Matrix $\mathbf{T}$

Given that $\mathbf{A}^L$ is similar to a companion matrix, it remains to construct $\mathbf{A}_{Lt}$ as the appropriate companion matrix. This can be achieved by finding a matrix $\mathbf{T}$ that represents the similarity transformation in (5). In general, the matrix $\mathbf{T}$ is not unique.

Theorem 3: Given a companion matrix $\mathbf{A}$ and a system defined as in (5), the transformation matrix $\mathbf{T}$ is not unique.

Proof: From theorem 2, a transformation matrix $\mathbf{T}$ exists for a given companion matrix $\mathbf{A}$. Let us assume $\mathbf{T}$ is unique. From (5), we have

$$\mathbf{A}_{Lt} = \mathbf{T}^{-1}\mathbf{A}^L\mathbf{T}.$$

Let $\mathbf{T}_1 = \mathbf{A}^i\mathbf{T}$, where i is any integer.

$$\begin{aligned}\mathbf{T}_1^{-1}\mathbf{A}^L\mathbf{T}_1 &= \mathbf{T}^{-1}\mathbf{A}^{-i}\mathbf{A}^L\mathbf{A}^i\mathbf{T}\\ &= \mathbf{T}^{-1}\mathbf{A}^L\mathbf{T} = \mathbf{A}_{Lt}.\end{aligned}$$

This shows that $\mathbf{T}_1$ is also a transformation matrix which leads to same similarity transformation which by contradiction proves that $\mathbf{T}$ is not unique. ■

As a counter example, we obtain two different $\mathbf{T}$ matrices for a given generator polynomial. This is illustrated by an example for the standard 16-bit CRC polynomial and is presented in the Appendix.

The construction of $\mathbf{T}$ matrix is based on cyclic vector of the transformation represented by the matrix $\mathbf{A}^L$. Select an arbitrary vector $\mathbf{b}_1$, subject only to the condition that the vectors $\mathbf{A}^{kL}\mathbf{b}_1$, for $k = 0, 1, \ldots, K-1$, are linearly independent. Now, let $\mathbf{T}$ be the matrix whose columns are these vectors, i.e.

$$\mathbf{T} = [\mathbf{b}_1 \quad \mathbf{A}^L\mathbf{b}_1 \quad \mathbf{A}^{2L}\mathbf{b}_1 \quad \ldots \quad \mathbf{A}^{(K-1)L}\mathbf{b}_1]. \tag{11}$$

Since all the columns are linearly independent, $\mathbf{T}$ is non-singular. We can show that matrix $\mathbf{T}$ implements the desired similarity relationship described in (5).

$$\mathbf{T}^{-1}\mathbf{A}^L\mathbf{T} = \mathbf{T}^{-1}[\mathbf{A}^L\mathbf{b}_1 \quad \mathbf{A}^{2L}\mathbf{b}_1 \quad \ldots \quad \mathbf{A}^{KL}\mathbf{b}_1].$$

We can observe that the first $K-1$ columns in the matrix on the right-hand side (RHS) are the same as the last $K-1$ columns in the $\mathbf{T}$ matrix. Therefore, $\mathbf{A}_{Lt}$ is similar to a companion matrix since first $K-1$ columns will be same as the first $K-1$ columns of $\mathbf{A}$ and the last column depends on the chosen vector $\mathbf{b}_1$. Then by theorem in [32, p. 230], we can show that for such a matrix $\mathbf{A}^L$, a cyclic vector exists. In [22], an exhaustive search algorithm is proposed to find the optimal $\mathbf{b}_1$ such that the number of 1's in $\mathbf{B}_{Lt}$ and $\mathbf{C}_{Lt}$ are reduced.

Using the transformation, the complexity of feedback loop will be identical to that of the original system. The feedback loop after the transformation looks as shown in Fig. 5. The complexity has been moved out of the feedback loop. Also the matrix multiplications (with $\mathbf{B}_{Lt}$ and $\mathbf{C}_{Lt}$) have to be pipelined to reduce the critical path which leads to an increase in the hardware cost. We propose a new architecture based on parallel IIR filter design to achieve high speed with less hardware cost.

V. IIR Filter Representation of LFSR

In this section, we propose a new formulation for LFSR architecture. Let's assume that the input to the LFSR is $u(n)$, and the required output, i.e., the remainder, is $y(n)$ as shown in Fig. 6. Then the LFSR can be described using the following equations:

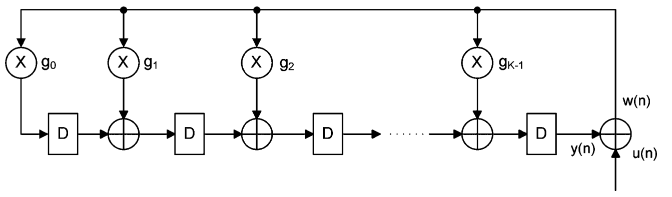

Fig. 6. General LFSR architecture.

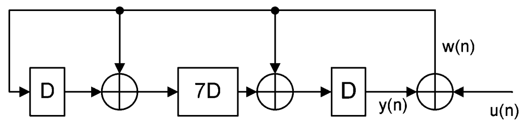

Fig. 7. LFSR architecture for $g(x) = 1 + x + x^8 + x^9$.

$$w(n) = y(n) + u(n) \tag{12}$$

$$\begin{aligned} y(n) =& g_{K-1} * w(n-1) + g_{K-2} * w(n-2) + \cdots\\ &+ g_0 * w(n-K). \end{aligned} \tag{13}$$

Substituting (12) into (13), we get

$$\begin{aligned} y(n) = g_{K-1} * y(n-1) + g_{K-2} * y(n-2) + \cdots\\ + g_0 * y(n-K) + f(n) \end{aligned} \tag{14}$$

where

$$f(n) = g_{K-1}*u(n-1)+g_{K-2}*u(n-2)+\cdots+g_0*u(n-K).$$

In the equations above, $'+'$ denotes XOR operation. We can observe that equation resembles an IIR filter with $g_0, g_1, \ldots, g_{K-1}$ as coefficients.

Consider a generator polynomial $g(x) = 1+x+x^8+x^9$. The corresponding LFSR architecture is shown in Fig. 7. By using the above formulation in (14)

$$y(n) = y(n-9) + y(n-8) + y(n-1) + f(n) \tag{15}$$

where

$$f(n) = u(n-9) + u(n-8) + u(n-1).$$

The following example illustrates the correctness of the proposed method. The CRC architecture for the above equation is shown in Fig. 8. Let the message sequence be 101011010; Table I shows the data flow at the marked points of this architecture at different time slots. In Table I, we can see that this architecture requires 17 clock cycles to compute the output. After N clock cycles (here $N = 9$), the feedback is reset to zero similar to the basic LFSR circuit shown in Fig. 1. The registers contain the intermediate value of the computation. To not affect the intermediate values, the input has to be reset to zero, until the remainder bits are shifted out.

VI. Parallel LFSR Architecture Based on IIR Filtering

We can derive parallel architectures for IIR filters using look ahead techniques [33]. We use the same look-ahead technique

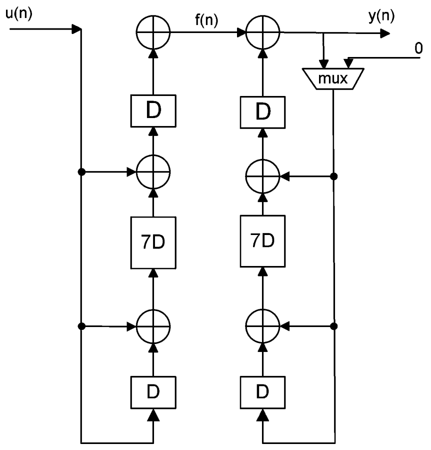

Fig. 8. LFSR architecture for $g(x) = 1 + x + x^8 + x^9$ after the proposed formulation.

TABLE I
DATA FLOW OF FIG. 8 WHEN THE INPUT MESSAGE IS 101011010

clock	$u(n)$	$f(n)$	$y(n)$
1	1	1	1
2	0	0	1
3	1	1	0
4	0	0	0
5	1	1	1
6	1	1	0
7	0	0	0
8	1	0	0
9	0	1	0
10	0	1	1
11	0	1	0
12	0	1	1
13	0	0	1
14	0	1	0
15	0	1	1
16	0	1	1
17	0	0	0

to derive parallel system for a given LFSR. Parallel architecture for a simple LFSR described in the previous section is discussed first.

Consider the design of a 3-parallel architecture for the LFSR in Fig. 7. In the parallel system, each delay element is referred to as a block delay where the clock period of the parallel system is 3 times the original sample period (bit period). Therefore, instead of (15), the loop update equation should update $y(n)$ using inputs and $y(n-3)$. The loop update process for the 3-parallel system is shown in Fig. 9, where $y(3k+3)$, $y(3k+4)$, and $y(3k+5)$ are computed using $y(3k)$, $y(3k+1)$, and $y(3k+2)$. By iterating the recursion or by applying look-ahead technique, we get

$$\begin{aligned} y(n) &= y(n-1) + y(n-8) + y(n-9) + f(n) && (16)\\ &= y(n-2) + y(n-8) + y(n-10)\\ &\quad + f(n-1) + f(n) && (17)\\ &= y(n-3) + y(n-8) + y(n-11)\\ &\quad + f(n-2) + f(n-1) + f(n). && (18) \end{aligned}$$

Substituting $n = 3k+3, 3k+4, 3k+5$ in the above equations, we have the following 3 loop update equations:

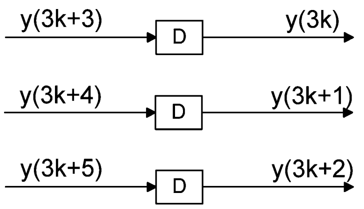

Fig. 9. Loop update equations for block size $L = 3$.

$$\begin{aligned} y(3k+3) &= y(3k+2) + y(3k-5) + y(3k-6)\\ &\quad + f(3k+3) && (19)\\ y(3k+4) &= y(3k+2) + y(3k-4) + y(3k-6)\\ &\quad + f(3k+3) + f(3k+4) && (20)\\ y(3k+5) &= y(3k+2) + y(3k-3) + y(3k-6)\\ &\quad + f(3k+3) + f(3k+4) + f(3k+5) && (21) \end{aligned}$$

where

$$\begin{aligned} f(3k+3) &= u(3k+2) + u(3k-5) + u(3k-6)\\ f(3k+4) &= u(3k+3) + u(3k-4) + u(3k-5)\\ f(3k+5) &= u(3k+4) + u(3k-3) + u(3k-4). \end{aligned}$$

The 3-parallel system is shown in Fig. 10. The input message 101011010 in the previous example leads to the data flow as shown in Table II. We can see that 6 clock cycles are needed to encode a 9-bit message. The proposed architecture requires $\frac{(N+K)}{L}$ cycles where N is the length of the message, K is the order of the generating polynomial, and L is the parallellism level. Similar to the serial architecture, the feedback path is reset to zero after the message bits have been processed. The critical path (CP) of the design is $6T_{\text{xor}}$, where T_{xor} is the two-input XOR gate delay. We can further reduce this delay by pipelining which is explained in the next section.

VII. PIPELINING AND RETIMING FOR REDUCING CRITICAL PATH

The critical path of the parallel system can be reduced by further pipelining and retiming the architecture. In this section, we show how to pipeline and retime for reducing the critical path to obtain fast parallel architecture by using the example in Fig. 7. Pipelining is possible only when there exist feed-forward cutsets [26]. When we unfold an LFSR directly, pipelining is not possible since there are no feed-forward cutsets. In the proposed method, the calculation of $f(n)$ terms is feed-forward. So, we can pipeline at this stage to reduce the critical path.

By designing the 3-parallel system, we obtain the design in Fig. 10. It is obvious that critical path of the system is $6T_{\text{xor}}$. We can reduce it to $3T_{\text{xor}}$ by applying pipelining cutset as shown in Fig. 10.

If the parallelism level (L) is high, the number of $f(n)$ terms to be added will increase leading to a large critical path. We apply tree structure to reduce the critical path while keeping the hardware overhead to a minimum. Further, we observe many common terms in the filter equations if the parallelism level is

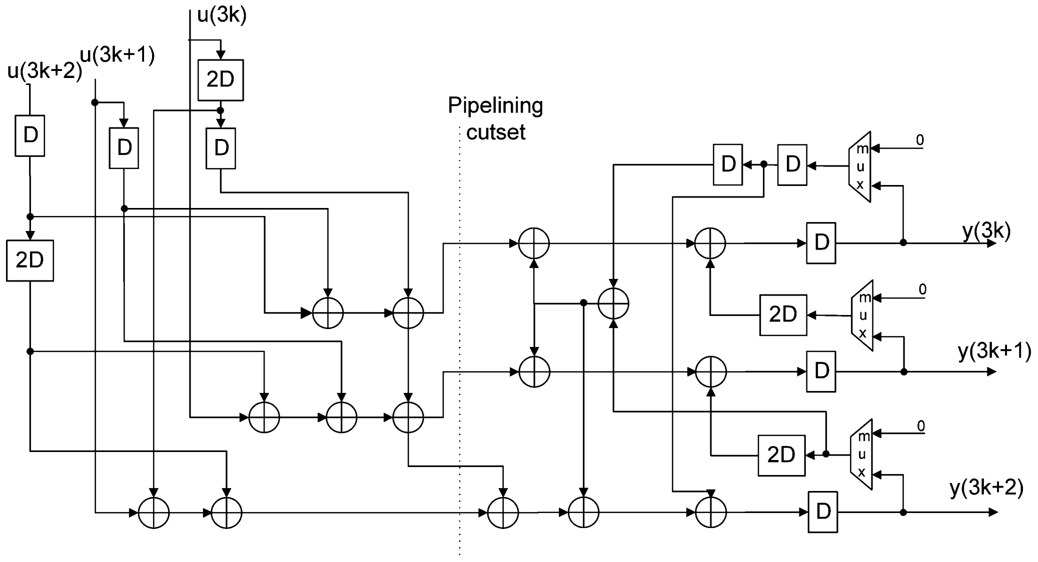

Fig. 10. Three parallel LFSR architecture after pipelining for Fig. 7.

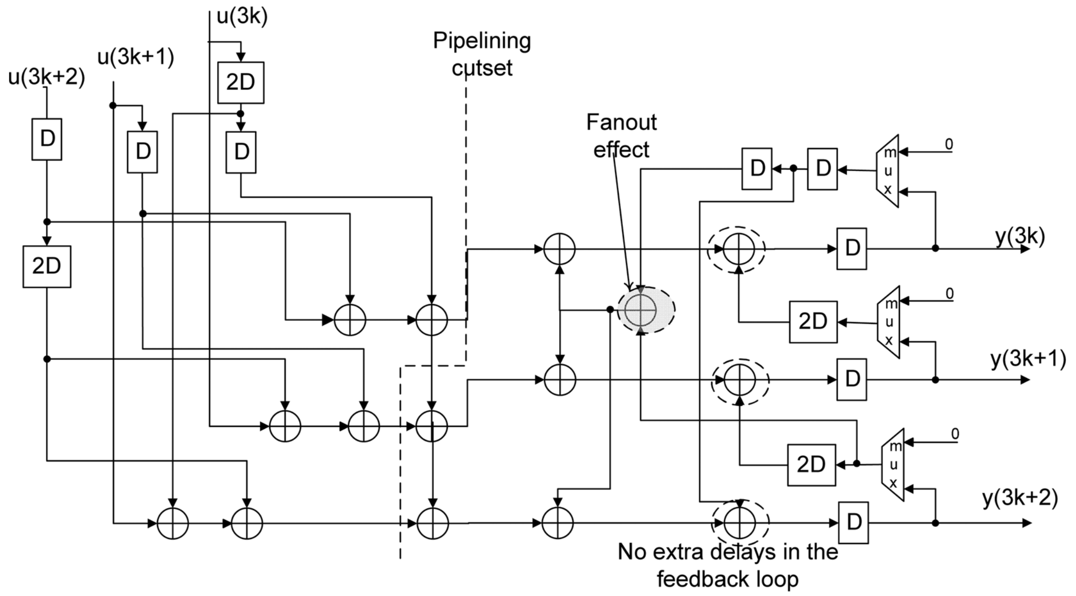

Fig. 11. Retiming and pipelining cutsets in three parallel LFSR architecture to reduce C.P to $2T_{\mathrm{xor}}$ for Fig. 7.

TABLE II
DATA FLOW OF FIG. WHEN THE INPUT MESSAGE IS 101011010

Clock#	$u(3k)$	$u(3k+1)$	$u(3k+2)$	$y(3k)$	$y(3k+1)$	$y(3k+2)$
1	1	0	1	0	1	1
2	0	1	1	0	0	1
3	0	1	0	0	0	0
4	0	0	0	0	1	0
5	0	0	0	1	1	0
6	0	0	0	1	1	0

high. To reduce the hardware complexity, subexpression sharing is applied [26]. In particular, we used multiple constant multiplication (MCM) [34] method to find the common computations across the filter equations. This algorithm uses an iterative matching process to find the common subexpressions.

Retiming can be applied to further reduce the critical path to $2T_{\mathrm{xor}}$. By using the pipelining cutsets as shown in Fig. 11 and the retiming around the circled nodes, we can reduce the critical path to $2T_{\mathrm{xor}}$. We can observe that there will be some hardware overhead to achieve this critical path. But one path around the bottom circled node has still 3 XOR gates. This is because one of the feedback paths does not have preexisting delays. This can be solved by using combined parallel and pipelining technique of IIR filter design.

A. Combined Parallel Processing and Pipelining

We can see from Fig. 11 that, when we have additional delays at the feedback loops, we can retime around the commonly shared nodes (as indicated in Fig. 11) to remove the large fanout effect. We can further reduce the critical path by combining parallel processing and pipelining concepts for IIR filter design [26]. Instead of using one-step look ahead, we use two-step look ahead computation to generate the filter equations. In this example, we need to compute $y(3k+8), y(3k+7), y(3k+6)$

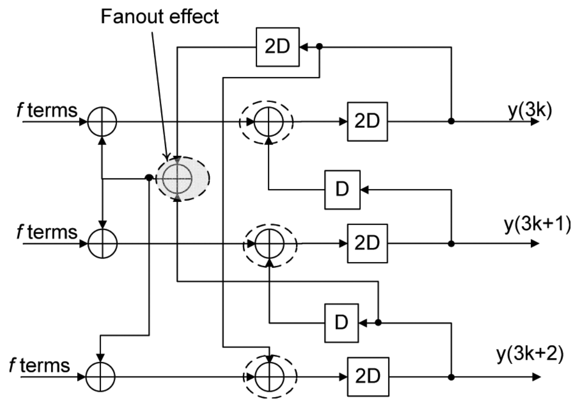

Fig. 12. Loop update for combined parallel pipelined LFSR.

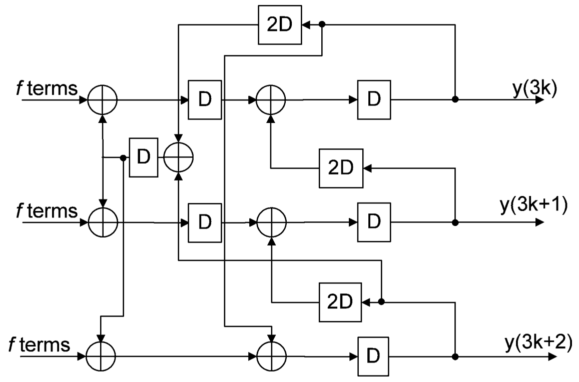

Fig. 13. Loop update for combined parallel pipelined LFSR after retiming.

instead of $y(3k+5), y(3k+4), y(3k+3)$. By doing this, we get two delays in the feedback loop. We can then retime to reduce the critical path and also remove the fanout effect on the shared node.

Now, the loop update equations are

$$\begin{aligned} y(3k+3) =& y(3k+2) + y(3k-2) + y(3k-6) \\ &+ f(3k+3) + \cdots + f(3k+6) \quad (22) \\ y(3k+4) =& y(3k+2) + y(3k-1) + y(3k-6) \\ &+ f(3k+3) + \cdots + f(3k+7) \quad (23) \\ y(3k+5) =& y(3k+2) + y(3k) + y(3k-6) \\ &+ f(3k+3) + \cdots + f(3k+8). \quad (24) \end{aligned}$$

The feedback part of the architecture is shown in Fig. 12. We can observe from this figure that retiming can be applied at all the circled nodes to reduce the critical path in the feedback section. Further, fanout bottleneck at any node can be alleviated using retiming at that particular node. The final architecture after retiming is shown in Fig. 13.

VIII. Comparison and Analysis

We limit our discussion to the commonly used generator polynomials for CRC and BCH encoders that are shown in Table III. A comparison between the previous high-speed architectures and the proposed ones is shown in Table IV for different parallelism levels of different generator polynomials. The comparison is made in terms of required number of xor gates (# XOR), required number of delay elements (# D.E.) and critical path (C.P.) (in terms of XOR gates) of the architectures for different parallelism levels (L). We define area-time product (A.T.) as (C.P.)*(#XOR + 1.5#D.E.) as an efficiency parameter, where 1.5 is the ratio between a delay element and an XOR gate as estimated in terms of number of NAND gates. This is based on the assumption that an XOR gate requires 4 2-input NAND gates and a delay element requires 6 2-input NAND gates. The values presented are normalized with respect to the results of the proposed design. The lower A.T. value represents a better implementation.

TABLE III
Commonly Used Generator Polynomials

CRC-12	$x^{12}+x^{11}+x^3+x^2+x+1$
CRC-16	$x^{16}+x^{15}+x^2+1$
SDLC	$x^{16}+x^{12}+x^5+1$
CRC-16 REVERSE	$x^{16}+x^{14}+x+1$
SDLC REVERSE	$x^{16}+x^{11}+x^4+1$
CRC-32	$x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x+1$

TABLE IV
Comparison to Previous LFSR Architectures and the Proposed One

Poly (L)	Algorithm	# XOR	# D.E.	C.P.	A.T.
CRC-12 (12)	[9]	113	35	5	1.01
	[13]*	276	47	3.15	1.70
	[5]	52	12	10	0.86
	Proposed	109	36	5	1
CRC-16 (16)	[9]	187	48	5	1.37
	[13]*	400	76	4	2.18
	[5]	72	16	15	1.53
	Proposed	113	50	5	1
SDLC (16)	[9]	207	48	5	1.31
	[13]*	400	54	5.44	2.69
	[5]	88	16	8	0.84
	Proposed	139	50	5	1
CRC-16 Reverse (16)	[9]	219	46	5	1.43
	[13]*	592	68	5.97	4.14
	[5]	154	16	15	2.66
	Proposed	126	50	5	1
SDLC Reverse (16)	[9]	217	48	5	1.43
	[13]*	233	76	7.4	2.74
	[5]	84	16	8	0.86
	Proposed	127	50	5	1
CRC-32 (32)	[9]	968	96	5	1.18
	[13]*	6496	344	16.13	25.42
	[5]	452	32	17	1.81
	Proposed	794	96	5	1
BCH (255,223) (32)	[9]	903	96	5	1.24
	[13]*	4832	276	14	14.58
	[5]	538	32	24	2.80
	Proposed	863	96	5	1

*Reported values are minimum achievable C.P

The numbers for [9] are calculated after the design is pipelined to have a critical path of $5T_{\text{xor}}$. The transformation matrix $\mathbf{T}$ is computed using $\mathbf{b}_1 = [1 \;\; 0 \;\; \ldots \;\; 0]^T$. We can further pipeline the architecture to reduce the critical path to $2T_{\text{xor}}$ but it will increase the required number of delay elements. Note that the latency of this design increases by 2. The critical path for the design in [13] shown in Table IV is the minimum achievable critical path, i.e., the iteration bound. Tree structure is applied to further reduce the C.P. to $O(\log(C.P.))$ at the cost of some additional XOR gates.

In the Table IV, we can observe that for all the listed commonly used generator polynomials, the proposed design

TABLE V
COMPARISON BETWEEN PREVIOUS LFSR ARCHITECTURES AND THE PROPOSED ONE

Poly (L)	Algorithm	# XOR	# D.E.
CRC-12 (12)	[22]*	112	91
	[22]†	96	84
	Proposed	109	36
CRC-16 (16)	[22]*	186	131
	[22]†	156	125
	Proposed	113	50
SDLC (16)	[22]*	206	152
	[22]†	194	139
	Proposed	139	50
CRC-16 Reverse (16)	[22]*	218	139
	[22]†	158	125
	Proposed	126	50
SDLC Reverse (16)	[22]*	216	155
	[22]†	194	139
	Proposed	127	50
CRC-32 (32)	[22]*	967	562
	[22]†	864	477
	Proposed	794	96

* with $\mathbf{b}_1 = [1 \quad 0 \quad \dots \quad 0]^T$
† with optimal $\mathbf{b}_1$ found through exhaustive search taken from [22]

TABLE VI
COMPARISON OF C.P AND XOR GATES OF THE PROPOSED DESIGN AND PREVIOUS PARALLEL LONG BCH(8191,7684) ENCODERS FOR L-PARALLEL ARCHITECTURE

		$L=8$	$L=16$	$L=24$	$L=32$
C.P (T_{xor})	Proposed	9	9	9	9
	[13]*	3.5	7.03	10.2	13.63
	[12]*	4.167	7.769	11.111	14.034
XOR gates	Proposed	2012	4069	6125	8229
	[13]	2360	4032	8520	10592
	[12]	2845	5469	9532	12512

*Reported values are minimum achievable C.P

achieves a critical path that is same as the previous designs without increasing the hardware overhead. The proposed design takes $\frac{(N+K)}{L}$ clock cycles to process N bits. The designs in [9] and [13] takes $\frac{N}{L}$ clock cycles to process N bits. Our design can achieve the same throughput rate at the reduced hardware cost.

In Table V, the proposed design is compared with the CRC computation using improved state space transformation [22]. The transformation matrix in [22] is chosen such that the complexity of $\mathbf{B}_{Lt}$ and $\mathbf{C}_{Lt}$ is minimized.

Comparison of critical path (C.P.) and XOR gates of the proposed design with previous parallel long BCH(8191,7684) encoders in [13] and [12] for L-parallel architecture is shown in Table VI. From the table, we can observe that the proposed design performs better in terms of hardware efficiency. The proposed design also performs better in terms of critical path for higher levels of parallelism. The values reported in [13] and [12] are iteration bounds, i.e., the minimum achievable critical path. Even for the cases $L = 8$ and $L = 16$, we can further pipeline and retime to reduce the critical path as proposed in Section VII.

We can see from Fig. 13 that critical path in the feedback loop can be reduced. This is possible because of the extra delay in the feedback path. This shows that our proposed approach of combined parallel and pipelining technique can reduce the critical path as a whole. The elimination of fanout effect on the XOR gate (shared subexpression) is another advantage of this approach. This comes at the cost of extra hardware.

IX. CONCLUSION

This paper has presented a complete mathematical proof to show that a transformation exists in state space to reduce the complexity of the parallel LFSR feedback loop. Further, a novel method for high speed parallel implementation of linear feedback shift registers based on parallel IIR filter design is proposed. Our design can reduce the critical path and the hardware cost at the same time. The design is applicable to any type of LFSR architecture. Further we show that using combined pipelining and parallel processing techniques of IIR filtering, critical path in the feedback part of the design can be reduced. The large fan-out effect problem can also be minimized with some hardware overhead by retiming around the nodes with fan-out bottleneck. Future work will be directed towards evaluating the proposed design with combined parallel processing and pipelining for long BCH codes.

APPENDIX
EXAMPLE FOR STATE SPACE TRANSFORMATION

Consider the standard 16-bit CRC with the generator polynomial

$$g_{16}(x) = x^{16} + x^{15} + x^2 + 1$$

with $L = 16$, i.e., processing 16-bits of input sequence at a time. Using (2), we can compute matrices $\mathbf{A}$ and $\mathbf{A}^L$ are as follows:

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

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

We can observe that the maximum number of nonzero elements in any row of $\mathbf{A}^L$ is 15, i.e., the time required to compute the product is the time to compute 15 exclusive or operations. By using the proposed transformation, we can transform the matrix $\mathbf{A}^L$ into a companion matrix $\mathbf{A}_{Lt}$. We can compute $\mathbf{T}$, $\mathbf{T}^{-1}$, and $\mathbf{A}_{Lt}$ from (11) and (5), which are shown here. In this case, we use $\mathbf{b}_1 = [1 \quad 0 \quad \ldots \quad 0]^T$.

$$\mathbf{T} = \begin{bmatrix}
1&1&1&1&1&1&1&1&0&1&1&1&0&1&0&1\\
0&0&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
0&1&1&0&0&0&0&0&1&1&0&0&1&1&1&1\\
0&0&0&1&0&0&0&0&0&0&1&0&0&0&1&0\\
0&0&1&1&1&0&0&0&0&1&1&1&0&1&1&0\\
0&0&0&1&0&1&0&0&0&0&1&0&1&0&1&0\\
0&0&0&1&0&1&1&0&0&0&1&0&1&1&1&0\\
0&0&0&0&0&0&0&1&0&0&0&0&0&0&1&0\\
0&0&0&0&1&0&1&1&1&0&0&1&0&1&1&0\\
0&0&0&0&0&1&0&1&0&1&0&0&1&0&1&0\\
0&0&0&0&0&1&1&1&0&1&1&0&1&1&1&0\\
0&0&0&0&0&0&0&1&0&0&0&1&0&0&1&0\\
0&0&0&0&0&0&1&1&0&0&1&1&1&1&1&0\\
0&0&0&0&0&0&0&1&0&0&0&1&0&1&1&0\\
0&0&0&0&0&0&0&1&0&0&0&1&0&1&0&0\\
0&1&1&1&1&1&1&1&0&1&1&1&0&1&0&0
\end{bmatrix}$$

$$\mathbf{T}^{-1} = \begin{bmatrix}
1&0&1&0&0&1&0&0&1&0&0&1&0&0&1&0\\
0&0&0&1&1&0&1&1&0&1&1&0&1&1&0&1\\
0&1&1&1&0&1&0&1&0&1&0&1&0&1&0&1\\
0&0&0&1&0&1&1&0&0&1&1&0&0&1&1&0\\
0&1&1&1&1&0&0&1&0&0&0&1&0&0&0&1\\
0&0&0&1&0&1&0&0&0&1&1&1&1&0&0&0\\
0&0&0&0&0&1&1&0&0&0&0&1&0&1&0&0\\
0&0&0&0&0&0&0&1&0&0&0&0&0&1&1&0\\
0&1&1&1&1&1&1&1&1&0&0&0&0&0&0&1\\
0&0&0&1&0&1&0&1&0&1&0&0&0&0&0&0\\
0&0&0&0&0&1&1&0&0&1&1&0&0&0&0&0\\
0&0&0&0&0&0&0&1&0&0&0&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&1&1&1&1&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&1&0&1&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0\\
0&0&1&0&0&1&0&0&1&0&0&1&0&0&1&1
\end{bmatrix}$$

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

We provide another matrix $\mathbf{T}_1 = \mathbf{AT}$ below to show that this transformation is not unique. We get a different $\mathbf{T}$, by just changing the cyclic vector in the $\mathbf{T}$ matrix computation. We can observe that the two $\mathbf{T}$ matrices are different and lead to the same similarity transformation. This shows that multiple solutions exist for transformation matrix $\mathbf{T}$.

$$\mathbf{T}_1 = \begin{bmatrix}
0&1&1&1&1&1&1&1&0&1&1&1&0&1&0&0\\
1&1&1&1&1&1&1&1&0&1&1&1&0&1&0&1\\
0&1&0&0&0&0&0&0&1&0&0&0&1&0&1&1\\
0&1&1&0&0&0&0&0&1&1&0&0&1&1&1&1\\
0&0&0&1&0&0&0&0&0&0&1&0&0&0&1&0\\
0&0&1&1&1&0&0&0&0&1&1&1&0&1&1&0\\
0&0&0&1&0&1&0&0&0&0&1&0&1&0&1&0\\
0&0&0&1&0&1&1&0&0&0&1&0&1&1&1&0\\
0&0&0&0&0&0&0&1&0&0&0&0&0&0&1&0\\
0&0&0&0&1&0&1&1&1&0&0&1&0&1&1&0\\
0&0&0&0&0&1&0&1&0&1&0&0&1&0&1&0\\
0&0&0&0&0&1&1&1&0&1&1&0&1&1&1&0\\
0&0&0&0&0&0&0&1&0&0&0&1&0&0&1&0\\
0&0&0&0&0&0&1&1&0&0&1&1&1&1&1&0\\
0&0&0&0&0&0&0&1&0&0&0&1&0&1&1&0\\
0&1&1&1&1&1&1&0&0&1&1&0&0&0&0&0
\end{bmatrix}$$

$$\mathbf{T}_1^{-1} = \begin{bmatrix}
1&1&0&1&0&0&1&0&0&1&0&0&1&0&0&1\\
1&0&0&0&1&1&0&1&1&0&1&1&0&1&1&0\\
0&0&1&1&1&0&1&0&1&0&1&0&1&0&1&0\\
1&0&0&0&1&0&1&1&0&0&1&1&0&0&1&1\\
0&0&1&1&1&1&0&0&1&0&0&0&1&0&0&0\\
0&0&0&0&1&0&1&0&0&0&1&1&1&1&0&0\\
0&0&0&0&0&0&1&1&0&0&0&0&1&0&1&0\\
1&0&0&0&0&0&0&0&1&0&0&0&0&0&1&1\\
0&0&1&1&1&1&1&1&1&1&0&0&0&0&0&0\\
0&0&0&0&1&0&1&0&1&0&1&0&0&0&0&0\\
0&0&0&0&0&0&1&1&0&0&1&1&0&0&0&0\\
0&0&0&0&0&0&0&0&1&0&0&0&1&0&0&0\\
0&0&0&0&0&0&0&0&0&0&1&1&1&1&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0\\
1&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1\\
0&0&0&1&0&0&1&0&0&1&0&0&1&0&0&1
\end{bmatrix}.$$

It may indeed be noted that any $\mathbf{T}_i = \mathbf{A}^i\mathbf{T}$ is a valid similarity transformation.

Acknowledgment

The authors are grateful to Dr. J. H. Derby for valuable discussions. Part of this work is inspired by his early work in [9]. The authors are also grateful to the anonymous reviewers for their numerous constructive comments.

References

[1] T. V. Ramabadran and S. S. Gaitonde, "A tutorial on CRC computations," *IEEE Micro.*, Aug. 1988.

[2] R. E. Blahut, *Theory and Practice of Error Control Codes*. Reading, MA: Addison-Wesley, 1984.

[3] W. W. Peterson and D. T. Brown, "Cyclic codes for error detection," *Proc. IRE*, vol. 49, pp. 228–235, Jan. 1961.

[4] N. Oh, R. Kapur, and T. W. Williams, "Fast speed computation for reseeding shift register in test pattern compression," *IEEE ICCAD*, pp. 76–81, 2002.

[5] T. B. Pei and C. Zukowski, "High-speed parallel CRC circuits in VLSI," *IEEE Trans. Commun.*, vol. 40, no. 4, pp. 653–657, Apr. 1992.

[6] K. K. Parhi, "Eliminating the fanout bottleneck in parallel long BCH encoders," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 51, no. 3, pp. 512–516, Mar. 2004.

[7] C. Cheng and K. K. Parhi, "High speed parallel CRC implementation based on unfolding, pipelining, retiming," *IEEE Trans. Circuits Syst. II, Expr. Briefs*, vol. 53, no. 10, pp. 1017–1021, Oct. 2006.

[8] G. Campobello, G. Patane, and M. Russo, "Parallel CRC realization," *IEEE Trans. Comput.*, vol. 52, no. 10, pp. 1312–1319, Oct. 2003.

[9] J. H. Derby, "High speed CRC computation using state-space transformation," in *Proc. Global Telecommun. Conf. (GLOBECOM'01)*, vol. 1, pp. 166–170.

[10] G. Albertengo and R. Sisto, "Parallel CRC generation," *IEEE Micro*, vol. 10, pp. 63–71, Oct. 1990.

[11] S. L. Ng and B. Dewar, "Parallel realization of the ATM cell header CRC," *Comput. Commun.*, vol. 19, pp. 257–263, Mar. 1996.

[12] X. Zhang and K. K. Parhi, "High-speed architectures for parallel long BCH encoders," in *Proc. ACM Great Lakes Symp. VLSI*, Boston, MA, Apr. 2004, pp. 1–6.

[13] C. Cheng and K. K. Parhi, "High speed VLSI architecture for general linear feedback shift register (LFSR) structures," in *Proc. 43rd Asilomar Conf. on Signals, Syst., Comput.*, Monterey, CA, Nov. 2009, pp. 713–717.

[14] R. J. Glaise, "A two-step computation of cyclic redundancy code CRC-32 for ATM networks," *IBM J. Res. Devel.*, vol. 41, pp. 705–709, Nov. 1997.

[15] M. Y. Hsiao and K. Y. Sih, "Serial-to-parallel transformation of linear feedback shift register circuits," *IEEE Trans. Electron. Comput.*, vol. EC-13, pp. 738–740, Dec. 1964.

[16] A. M. Patel, "A multi-channel CRC register," in *Proc. AFIPS Conf.* , 1971, vol. 38, pp. 11–14.

[17] H. Chen, "CRT-based high-speed parallel architecture for long BCH encoding," *IEEE Trans. Circuits Syst. II: Expr. Briefs*, vol. 56, no. 8, pp. 684–686, Aug. 2009.

[18] F. Liang and L. Pan, "A CRT-based BCH encoding and FPGA implementation," in *Proc. Int. Conf. Inf. Sci. Appl. (ICISA)*, Apr. 2010, pp. 21–23.

[19] C. Kennedy, J. Manii, and J. Gribben, "Retimed two-step CRC computation on FPGA," in *Proc. 23rd Canadian Conf. Elect. Comput. Eng. (CCECE)*, May 2–5, 2010, pp. 1–7.

[20] C. Toal, K. McLaughlin, S. Sezer, and X. Yang, "Design and implementation of a field programmable CRC circuit architecture," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 17, no. 8, pp. 1142–1147, Aug. 2009.

[21] K. Septinus, T Le, U. Mayer, and P. Pirsch, "On the design of scalable massively parallel CRC circuits," in *Proc. IEEE Int. Conf. Electron., Circuits Syst.*, Dec. 11–14, 2007, pp. 142–145.

[22] C. Kennedy and A. Reyhani-Masoleh, "High-speed CRC computations using improved state-space transformations," in *Proc. IEEE Int. Conf. Electro/Inf. Technol.*, Jun. 7–9, 2009, pp. 9–14.

[23] A. Doring, "Concepts and experiments for optimizing wide-input streaming CRC circuits," in *Proc. 23rd Int. Conf. Architect. Comput. Syst.*, Feb. 2010.

[24] Y. Do, S. R. Yoon, T. Kim, K. E. Pyun, and S. Park, "High-speed parallel architecture for software-based CRC," in *Proc. IEEE Consumer Commun. Netw. Conf.*, Jan. 10–12, 2008, pp. 74–78.

[25] M. E. Kounavis and F. L. Berry, "Novel table lookup-based algorithms for high-performance CRC generation," *IEEE Trans. Comput.*, vol. 57, no. 11, pp. 1550–1560, Nov. 2008.

[26] K. K. Parhi, *VLSI Digital Signal Processing Systems: Design and Implementation*. Hoboken, NJ: Wiley, 1999.

[27] M. Ayinala and K. K. Parhi, "Efficient parallel VLSI architecture for linear feedback shift registers," in *Proc. IEEE Workshop on SiPS*, Oct. 2010, pp. 52–57.

[28] K. K. Parhi and D. G. Messerschmitt, "Static-rate optimal scheduling of iterative data-flow programs via optimum unfolding," *IEEE Trans. Comput.*, vol. 40, no. 2, pp. 178–195, Feb. 1991.

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

[30] S. Lin and D. J. Costello, *Error Control Coding: Fundamental and Applications*, 2nd ed. Englewood Cliffs, NJ: Prentice-Hall, 2004.

[31] P. Koopman and T. Chakravarty, "Cyclic redundancy code (CRC) polynomial selection for embedded networks," in *Proc. DSN04*, Jun. 2004.

[32] K. Hoffman and R. Kunze, *Linear Algebra*. Englewood Cliffs, NJ: Prentice-Hall, 1971.

[33] K. K. Parhi and D. G. Messerschmitt, "Pipeline interleaving and parallelism in recursive digital filters—Part II: Pipelined incremental block filtering," *IEEE Trans. Acoust., Speech Signal Process.*, vol. 37, pp. 1118–1135, Jul. 1989.

[34] M. Potkonjak, M. B. Srivastava, and A. P. Chandrakasan, "Multiple constant multiplications: Efficient and versatile framework and algorithms for exploring common subexpression elimination," *IEEE Trans. Comput.-Aided Design for Integr. Circuits Syst.*, vol. 15, no. 2, pp. 151–165, 1996.

Manohar Ayinala (S'10) received the Bachelor's degree in electronics and communication engineering from the Indian Institute of Technology, Guwahati, India, in 2006 and the M.S. degree in electrical and computer engineering from the University of Minnesota, Minneapolis, in 2010 , where he is currently working towards the Ph.D. degree.

His research interests include communication systems, digital signal processing and classification algorithms, and low-power/cost-efficient VLSI architectures and implementation.

Keshab K. Parhi (S'85–M'88–SM'91–F'96) received the B.Tech., M.S.E.E., and Ph.D. degrees from the Indian Institute of Technology, Kharagpur, the University of Pennsylvania, Philadelphia, and the University of California at Berkeley, in 1982, 1984, and 1988, respectively.

He has been with the University of Minnesota, Minneapolis, since 1988, where he is currently Distinguished McKnight University Professor with the Department of Electrical and Computer Engineering. His research addresses VLSI architecture design and implementation of physical layer aspects of broadband communications systems, error control coders and cryptography architectures, high-speed transceivers, and ultrawideband systems. He is also currently working on intelligent classification of biomedical signals and images, for applications such as seizure prediction, lung sound analysis, and diabetic retinopathy screening. He has published more than 450 papers, authored *VLSI Digital Signal Processing Systems* (New York: Wiley, 1999), and coedited *Digital Signal Processing for Multimedia Systems* (New York: Marcel-Dekker, 1999).

Dr. Parhi is the recipient of numerous awards including the 2004 F. E. Terman award by the American Society of Engineering Education, the 2003 IEEE Kiyo Tomiyasu Technical Field Award, the 2001 IEEE W. R. G. Baker prize paper award, and a Golden Jubilee award from the IEEE Circuits and Systems Society in 1999. He served on the editorial boards of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMSI AND II, VLSI Systems, the IEEE TRANSACTIONS ON SIGNAL PROCESSING, the IEEE TRANSACTIONS ON SIGNAL PROCESSING LETTERS, and *Signal Processing Magazine*, and served as the Editor-in-Chief of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMSI (2004–2005 term), and currently serves on the Editorial Board of the *Journal of VLSI Signal Processing*. He has served as Technical Program Co-Chair of the 1995 IEEE VLSI Signal Processing workshop and the 1996 ASAP conference, and as the General Chair of the 2002 IEEE Workshop on Signal Processing Systems. He was a Distinguished Lecturer for the IEEE Circuits and Systems society during 1996-1998. He was an elected member of the Board of Governors of the IEEE Circuits and Systems society from 2005 to 2007.