This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCSII.2018.2860934, IEEE Transactions on Circuits and Systems II: Express Briefs

A Low-Power Parallel Architecture for Linear Feedback Shift Registers

Xinmiao Zhang, *Senior Member, IEEE,*

***Abstract*—Linear feedback shift registers (LFSRs) are used to implement BCH encoders and cyclic redundancy check (CRC), which are broadly used in digital communication systems. Previous parallel LFSR designs adopt a state-space transformation that shortens the feedback data path and reduces the gate count. Transformations have been designed to minimize the total gate count of the three involved matrix multiplications. However, the transformation matrix multiplication is only active for one clock cycle at the end. In this paper, we propose an alternative transformation matrix construction that effectively shifts the complexity from the other two matrices, which are active in every clock cycle, to the transformation matrix without increasing the critical path or the total gate count. For an example CRC-32, the proposed design achieves 33% power and 8% gate count reductions without compromising the achievable clock frequency.**

***Index Terms*—BCH encoder, cyclic redundancy check (CRC), linear feedback shift register (LFSR), low-power, parallel architecture, substructure sharing**

I. Introduction

BCH codes and cyclic redundancy check (CRC) are broadly used to ensure the reliability and integrity of data transmission. The basic function in BCH encoding and CRC en/decoding is to compute remainders of polynomial divisions, which is implemented by linear feedback shift registers (LFSRs). The high-throughput requirements of modern digital communications demand LFSRs with high level of parallelism.

To achieve p-parallel processing, the states of the registers in the LFSR after p clock cycles are derived by look-ahead computations [1], which result in a matrix multiplication in the feedback loop that limits the achievable clock frequency. A state-space transformation approach was introduced in [2] to modify the feedback matrix at the cost of a pre-processing and a transformation matrix multiplications. A certain companion-matrix-like transformation was used in this work to make the critical path of the feedback matrix multiplication one XOR gate, although the other two matrix multiplications have longer data paths. In [3], exhaustive search is carried out over the same type of transformations to identify the one leading to the smallest overall gate count. Triangular transformation matrices are adopted in [4]. From exhaustive search, such transformations lead to LFSR designs with lower overall gate count without sacrificing the critical path. Another line of work interprets the LFSR function as recursive filtering, and parallel processing techniques for recursive filters are applied [5]. The complexity of such parallel LFSRs is reduced by adopting a transposed format of the recursive filter [6]. Nevertheless, its gate count is larger than the state-space transformation-based design in [4]. Parallel LFSRs can be also derived by applying the unfolding technique to serial LFSRs. Although the achievable clock frequency can be increased by manipulating the divisor polynomial [7] or applying alternative look-ahead pipelining to the critical loop [8], their area is larger.

In the recursive filter approaches [5], [6], all logic gates are active in every clock cycle. In those based on state-space transformations [2]–[4], the pre-processing and feedback matrix multiplications are active in every clock cycle, whereas the transformation matrix multiplication is only running in the last clock cycle.

This paper proposes a low-power parallel LFSR architecture that effectively shifts the complexities of pre-processing and feedback matrix multiplications to transformation matrix multiplication without increasing the overall gate count or critical path. Our design is achieved by exploring different state-space transformations. Instead of searching for a transformation matrix of certain format leading to minimized total gate count, our construction starts with the inverse transformation matrix and searches for a matrix that minimizes the gate count of the pre-processing matrix multiplication. The proposed matrix also leads to reduced number of nonzero entries in the feedback matrix. To better evaluate the logic complexity, a simplified critical path delay (CPD) computation method for systems adopting substructure sharing (SS) is also presented. Compared to the design in [4], the proposed design achieves 8% gate count reduction and 33% power reduction based on the number of logic gates active in each clock cycle for an example CRC-32.

This paper is organized as follows. Section II introduces parallel LFSRs based on state-space transformation. The proposed transformation construction is detailed in Section III. Section IV presents the modified SS scheme for given critical path constraints. Complexity comparisons are provided in Section V and conclusions follow in Section VI.

II. Parallel LFSRs by State-Space Transformation

The serial LFSR shown in Fig. 1 is configured using a generator polynomial $g(x) = x^{n-k} + g_{n-k-1}x^{n-k-1} + \cdots + g_1x + g_0$. For binary BCH codes and CRC, the coefficients are binary. When the input $u(x)$ is added to the most significant tap, this LFSR implements the division of $u(x)x^{n-k}$ by $g(x)$. Only the remainder $r(x)$ is of interest to BCH encoding and CRC. The coefficients of $u(x)$ are input serially starting with the most significant one, and $r(x)$ is located in the registers after the last coefficient of $u(x)$ is sent in.

Xinmiao Zhang is with the Department of Electrical and Computer Engineering, The Ohio State University, Columbus, OH 43210, USA. Email: zhang.8952@osu.edu.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCSII.2018.2860934, IEEE Transactions on Circuits and Systems II: Express Briefs

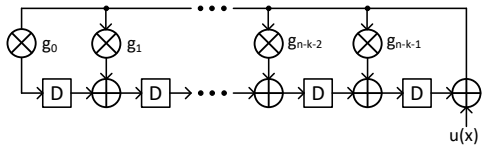

Fig. 1. Serial LFSR architecture

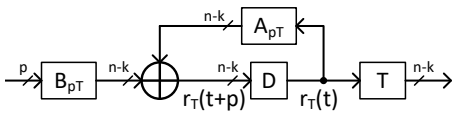

Fig. 2. Transformed p-parallel LFSR architecture

Denote the register states at clock cycle t by $\mathbf{r}(t) = [r_{n-k-1}(t), r_{n-k-2}(t), \cdots, r_0(t)]'$, where '$'$' represents transpose. Let $u(t)$ be the input at clock cycle t. Then

$$\mathbf{r}(t+1) = \mathbf{A} \times \mathbf{r}(t) + \mathbf{b} \times u(t) \tag{1}$$

where $\mathbf{A}$ is a companion matrix

$$\mathbf{A} = \begin{bmatrix} g_{n-k-1} & 1 & 0 & \cdots & 0 \\ g_{n-k-2} & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & 0 \\ g_1 & 0 & 0 & \cdots & 1 \\ g_0 & 0 & 0 & \cdots & 0 \end{bmatrix}$$

and $\mathbf{b} = [g_{n-k-1}, \cdots, g_1, g_0]'$. Substituting (1) back to itself p times, it can be derived that

$$\mathbf{r}(t+p) = \mathbf{A}^p \times \mathbf{r}(t) + \mathbf{B}_p \times \mathbf{u}_p(t), \tag{2}$$

where $\mathbf{B}_p = [\mathbf{A}^{p-1}\mathbf{b}, \cdots, \mathbf{A}\mathbf{b}, \mathbf{b}]$ and $\mathbf{u}_p(t) = [u(t), \cdots, u(t+p-2), u(t+p-1)]'$. A p-parallel LFSR that processes p bits in each clock cycle can be implemented according to (2).

The multiplication of $\mathbf{A}^p$ is in a feedback loop, and its data path limits the achievable clock frequency of the overall LFSR. To address this issue, it was proposed in [2] to transform the state vector as $\mathbf{r}(t) = \mathbf{T} \times \mathbf{r}_T(t)$, where $\mathbf{T}$ is non-singular and is referred to as the transformation matrix. Accordingly, the state equation in (2) for p-parallel processing becomes

$$\mathbf{r}_T(t+p) = \mathbf{A}_{pT} \times \mathbf{r}_T(t) + \mathbf{B}_{pT} \times \mathbf{u}_p(t), \tag{3}$$

where

$$\begin{aligned} \mathbf{A}_{pT} &= \mathbf{T}^{-1} \times \mathbf{A}^p \times \mathbf{T} \\ \mathbf{B}_{pT} &= \mathbf{T}^{-1} \times \mathbf{B}_p \end{aligned} \tag{4}$$

$\mathbf{B}_{pT}$ and $\mathbf{A}_{pT}$ are also referred to as the pre-processing and feedback matrices, respectively, in this paper. A block diagram for implementing such a transformed p-parallel LFSR is shown in Fig. 2. The complexities of the three matrix multiplications vary with $\mathbf{T}$, whose dimension is $(n-k) \times (n-k)$.

The $\mathbf{T}$ matrix considered in [2] is in the format of $\mathbf{T} = [\mathbf{b}_1, \mathbf{A}^p\mathbf{b}_1, \cdots, \mathbf{A}^{(n-k-1)p}\mathbf{b}_1]$, so that $\mathbf{A}_{pT}$ is a companion matrix, which has only one XOR gate in the data path. It turns out that the overall gate count is also reduced by the transformation, and exhaustive search was done in [3] to find the $\mathbf{b}_1$ leading to minimal gate count. In modern CMOS process, 5 or 6 XOR gates can easily fit into $1ns$ delay, and it becomes unnecessary to keep an extremely short 1-XOR-gate data path. To further reduce the gate count, [4] adopts an upper triangular matrix for $\mathbf{T}$, which has all '1's in the diagonal, and the nonzero entries in row $i+1$ equal to those in row i shifted to the right by one bit with the last bit eliminated.

III. Low-power parallel LFSRs

In existing designs, [4] has the lowest gate count. Nevertheless, this design spends a higher portion of the gates on the pre-processing ($\mathbf{B}_{pT}$) and feedback matrix ($\mathbf{A}_{pT}$) multiplications. These multiplications are active in every clock cycle, while the $\mathbf{T}$ matrix multiplication is only done for one clock cycle at the end. The dynamic power consumption can be compared by the number of logic gates switching in each clock cycle. One effective way to lower the power consumption is to reduce the gate counts of $\mathbf{B}_{pT}$ and $\mathbf{A}_{pT}$, even though $\mathbf{T}$ ends up having more gates.

$\mathbf{A}^{j+1}$ for $j \neq -1$ is a collection of column vectors that are binary representations of $\alpha^{n-k+j}, \cdots, \alpha^{2+j}, \alpha^{1+j}$, where α is a root of the LFSR generator polynomial. Also, $\mathbf{b}$ is the vector representation of α^{n-k}, and $\mathbf{A}^j\mathbf{b}$ is the vector representation of α^{n-k+j}. Following prior work, it is assumed that $p = n-k$. In this case, $\mathbf{A}^p = \mathbf{B}_p$. From (4), exhaustive search can be carried out to find a $\mathbf{T}^{-1}$ that minimizes the gate count of $\mathbf{B}_{pT}$. $\mathbf{T}$ is determined from $\mathbf{T}^{-1}$. Although the gate count of $\mathbf{A}_{pT}$ may not be minimized, having the lowest gate count in the $\mathbf{T}^{-1} \times \mathbf{A}^p$ part also helps to reduce the complexity of $\mathbf{A}_{pT}$ multiplication.

To define a valid transformation, the only requirement on $\mathbf{T}^{-1}$ or $\mathbf{T}$ is that it must be invertible. $n-k$ can be 16, 32 or higher in CRC and BCH codes. Searching over every possible $(n-k) \times (n-k)$ binary matrix for the optimal $\mathbf{T}^{-1}$ or $\mathbf{T}$ has prohibitive complexity. In [3] and [4], $\mathbf{T}$ of special format as explained in Section II are considered. The search is reduced to the first column or row, and the other columns or rows are derived by shifting and modulo reduction. Our search is done on $\mathbf{T}^{-1}$ instead. Also the later rows of $\mathbf{T}^{-1}$ are not derived from the first row just in order to reduce the search complexity. Instead, the goal is to minimize the weight of each row in $\mathbf{B}_{pT}$. The number of XOR gates needed to implement a matrix multiplication can be reduced by SS, which means common intermediate results among multiple outputs are computed once and shared. Nevertheless, comparing the row weights often gives a good indication of which matrix multiplication requires less logic gates.

Our proposed $\mathbf{T}^{-1}$ is in the following format

$$\mathbf{T}^{-1} = \begin{bmatrix} 0 & 0 & \cdots & 0 & 0 & 1 \\ 0 & 0 & \cdots & 0 & 1 & t_{1,0} \\ 0 & 0 & \cdots & 1 & t_{2,1} & t_{2,0} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 1 & \cdots & t_{n-k-2,2} & t_{n-k-2,1} & t_{n-k-2,0} \\ 1 & t_{n-k-1,n-k-2} \cdots & & t_{n-k-1,2} & t_{n-k-1,1} & t_{n-k-1,0} \end{bmatrix}.$$

Such a lower anti-triangular matrix with the anti-diagonal set to all '1's guarantees that $\mathbf{T}^{-1}$ is invertible. It is also possible to use a lower triangular, upper triangular, or upper anti-triangular format for $\mathbf{T}^{-1}$. Denote the nonzero entries in

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCSII.2018.2860934, IEEE Transactions on Circuits and Systems II: Express Briefs

row i by $\mathbf{t}_i = [1, t_{i,i-1}, \cdots, t_{i,1}, t_{i,0}]$. Exhaustive search is done to find the $\mathbf{t}_i$ that minimizes the weight of row i in $\mathbf{B}_{pT}$, and the search for different $\mathbf{t}_i$ is done independently. The overall search is carried out on $2^{n-k-1}+2^{n-k-2}+\cdots+2^1 = (1+2^{n-k-2})(n-k-1)$ instead of 2^{n-k-1} or 2^{n-k} vectors as in [3] and [4], respectively. Nevertheless, for $n-k=16$ the search takes less than a second to finish by MATLAB on a laptop. For larger $n-k$, such as 32, carrying out the full search over a single vector already takes extremely long time as also has been noted in [3], [4]. An approximate tree search method with early truncation was adopted in [4]. In our design, the search for $[t_{i,i-1}, \cdots, t_{i,1}, t_{i,0}]$ is limited to the vectors whose decimal value is in the range of $[0, 2^{min(i,m)}-1]$, where m is an upper bound. For $n-k=32$, the search is finished in less than a minute when m is set to 19, and a design with lower gate count than that in [4] is already found from this shortened search. Search has also been done by setting m=20, 21, and 22. The additional reduction on the row weights of $\mathbf{B}_{pT}$ is becoming smaller for larger m.

$\mathbf{A}_{pT} = \mathbf{B}_{pT} \times \mathbf{T}$ and $\mathbf{T}$ is computed from the $\mathbf{T}^{-1}$ found from the search. There might be multiple vectors for $\mathbf{t}_i$ that produce the same minimized weight in the ith row of $\mathbf{B}_{pT}$. These candidates lead to variations on $\mathbf{T}$ and hence $\mathbf{A}_{pT}$. They are recorded, and search over combinations of the candidate vectors can be carried out to reduce the complexity of $\mathbf{A}_{pT}$ and $\mathbf{T}$ multiplications. When $n-k$ is larger, there are more $\mathbf{t}_i$ with multiple candidates, and they have greater numbers of candidates. Trying every possible combination of the candidates also leads to overwhelming search complexity. To shorten the search, an upper bound, cap, can be set on the number of candidates to try for each t_i. Our proposed search method is summarized in Algorithm A.

Algorithm A: Proposed search method for $\mathbf{T}^{-1}$

for $i = 1$ to $n-k$
  search for $\mathbf{t}_i$ that minimize the weight of $[0, \cdots, 0, \mathbf{t}_i] \times \mathbf{B}_p$
  record every vector that leads to the minimum weight
  denote the vector number by n_i
if $\prod_{i=1}^{n-k} n_i$ is an overwhelming number
  set $n_i = min(n_i, cap)$
try n_i candidates of $\mathbf{t}_i$ and hence $\prod_{i=1}^{n-k} n_i$ combinations of $\mathbf{t}_i$ to find the one leading to minimal total number of '1's in $\mathbf{B}_{pT}$, $\mathbf{A}_{pT}$, and $\mathbf{T}$

From our simulations, using a small value for cap, such as 2 or 3, leads to noticeable improvement on the total number of '1's compared to not trying any alternative candidates for $\mathbf{t}_i$. Further increasing cap does not necessarily generates a better design.

IV. Substructure Sharing with Critical Path Constraints

If a substructure, also called a common term, appears in multiple output formulas, then this substructure only needs to be computed once and it can be shared in the computations of multiple outputs. The complexity of constant matrix multiplications is more accurately estimated by applying SS. The SS for achieving optimal gate count reduction is an NP-complete problem. Also different SS schemes lead to trade-offs on the gate count and CPD. To achieve high clock frequency in parallel LFSRs, SS needs to be applied with constraints on the CPD. Although many SS schemes targeting at gate count reduction have been proposed, only a few of them address the critical path issue associated with sharing substructures. In [9], the CPD is computed by constructing a restriction graph describing the SS, and a newly identified substructure is only adopted if it does not lead to CPD violation. In [10], the CPD is computed through updating a delay matrix along with identifying substructions for sharing. Matrices are more friendly to computer execution than restriction graphs.

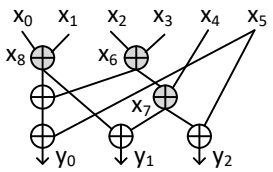

Fig. 3. Example of substructure sharing

In this section, a simplified vector-based CPD computation method for SS is proposed, and considerations in substructure selection that help to reduce the CPD are discussed. In our SS scheme, first consider that the 2-term pattern appearing most in the outputs is shared each time, since an XOR gate has two inputs. If there is a tie, pick a pattern randomly. Also substructures participate in the pattern matching recursively. Consider an example that computes $y_0 = x_0+x_1+x_2+x_3+x_5$, $y_1 = x_0+x_1+x_2+x_3+x_4$, and $y_2 = x_2+x_3+x_4+x_5$. The substructures identified for sharing in iteration 1, 2, and 3 are $x_6 = x_2+x_3$, $x_7 = x_6+x_4$ and $x_8 = x_0+x_1$, respectively. The SS can be expressed by a graph as shown in Fig. 3. Instead of converting this graph to a restriction graph and then apply a recursive process to compute the CPD, a depth vector, d, can be kept and updated along with the identifications of the substructures. d is initially an all-zero vector, when a substruction $x_l = x_i + x_j$ is adopted, an entry $d_l = \max\{x_i, x_j\} + 1$ is added to the vector. In the above example, d = [0000001], [00000012], [000000121] in iteration 1, 2, and 3, respectively. In the substructure-shared output formula, a term is either an original input or a shared substructure. Denote the numbers of terms whose values in the final depth vector are $0, 1, 2, \cdots$ by $s_0, s_1, s_2, \cdots$. Then the CPD for computing the output can be derived using Algorithm B. In this algorithm, s_{max} is the last nonzero s_i. The num in iteration i is the number of intermediate items available to be added up after i levels of XOR gates, assuming 2-input XORs are used to add up inputs, substructures, and intermediate results as early as possible in a tree structure. This algorithm generates the same result as that in [9], [10] with simpler interpretations and computations. In the above example, y_1 is computed as $x_7 + x_8$ using SS. $d_7 = 2$ and $d_8 = 1$ in the final depth vector. Hence $s_0 = 0, s_1 = 1$, and $s_2 = 1$. Applying Algorithm B, num is initialized to $s_0 = 0$, and becomes 1, 2 and 1 in the iterations that $i = 1, 2$ and 3, respectively. The iteration breaks when $i = 3$ and $CPD = 3$.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCSII.2018.2860934, IEEE Transactions on Circuits and Systems II: Express Briefs

TABLE I
GENERATOR POLYNOMIALS OF CRCS

	Generator polynomial
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 II
OPTIMAL $\mathbf{T}^{-1}$ FROM PROPOSED CONSTRUCTION

	$\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_3, \cdots$ in hexadecimal vectors
CRC-12	2, 5, A, 15, 2A, 55, AA, 155, 2AA, 555, AAB
CRC-16	3, 7, F, 1F, 3F, 7F, FF, 1FF, 3FF, 7FF, FFF, 1FFF 3FFF, 7FFF, FFFE
SDLC	2, 4, 8, 11, 23, 46, 8C, 108, 231, 463, 853, 118D 210A, 4298, 8C6B
CRC-16 reverse	3, 6, C, 18, 30, 60, C0, 180, 300, 600, C00, 1800 3000, 7FFE, FFFD
SDLC reverse	2, 4, 8, 11, 23, 44, 88, 111, 233, 466, 8CD, 1113 2226, 44C0, 8981
CRC-32	1, 3, 6, D, 1A, 35, 6A, D5, 1AA, 355, 6AA, D55 1409, 246B, 594A, FAA8, 1F551, 3EAA3, 7D546 FAA8C, 19F323, 2CA50B, 594A16, 9CDFC7 139BF8E, 20735A1, 40E6B42, 81CD684, 1039AD08 201384AD, 4018B734, 80225381

Algorithm B: Critical path delay computation method for systems with shared substructures

input: $s_0, s_1, \cdots, s_{max}$
$num = s_0, i = 0$
while $(num \neq 1)$ or $(i < max)$
  $i = i + 1$
  $num = \lceil (num/2) \rceil + s_i$ (set s_i=0 if $i > max$)
$CPD = i$

Similar to that in [9], for a possible shareable substructure, the CPD of each output adopting this substructure is computed. If the CPD exceeds the limit, then the substructure is not adopted in that output. As shown by the x_6 and x_7 nodes in Fig. 3, using a substructure as the input of another adds one more level of logic in the data path. When there are tight constraints on the CPD, such iterative SS may saturate the CPD early and prevent more substructures from being shared. One example for this case is the multiplication of a triangular matrix that is all '1' under or above the diagonal. To prevent this from happening, priorities can be given to those substructures built by using the original inputs first before recursive SS is allowed.

V. COMPARISONS WITH PRIOR DESIGNS

To compare with prior work, the CRCs listed in Table I are considered. Algorithm A is applied to find the $\mathbf{T}^{-1}$ for each CRC. For CRCs of length 12 and 16, all possible vectors for each $\mathbf{t}_i$ are searched, and all possible combinations of the $\mathbf{t}_i$ candidates are tried. For CRC-32, searching has been done by setting the upper bound, m, to 22, and up to $cap = 3$ candidates are tried for each $\mathbf{t}_i$ to form the $\mathbf{T}^{-1}$ matrix. The entire search is finished in less than 50 minutes in MATLAB on a laptop. The optimal $\mathbf{T}^{-1}$ found are listed in Table II. Each hexadecimal vector represents a $\mathbf{t}_i$. $\mathbf{t}_0 = 1$ and is not included. The complexities of the matrix multiplications are summarized in Table III. In this table, the numbers of '1's are directly counted from the matrices. The numbers of XOR gates are derived by applying the SS described in the previous section. The limit of the CPD is set to 4 for CRCs of length 12 and 16, and is set to 5 for CRC-32. As shown in Fig. 2, $n-k$ XOR gates are used to add up the outputs of the $\mathbf{A}_{pT}$ and $\mathbf{B}_{pT}$ multiplications. These XOR gates are included in the total gate counts in Table III. They also contribute to one more XOR in the data path and need to be included in the calculation of the CPD of the overall architecture. In state-of-the-art CMOS process, a data path of 5 or 6 XOR gates can easily fit into $< 1ns$ timing budget, and hence no pipelining is applied to the transformed LFSRs.

Various m and cap have been tried in our search for CRC-32. The numbers of '1's in $\mathbf{B}_{pT}$ are 193, 186, 180, 174, 173 when m is set to 18, 19, 20, 21, 22, respectively. When $m = 22$, the total number of '1's in the three matrices is reduced from 840 to 782 and futher to 771 when the value of cap is increased from 1 to 2 and to 3.

The complexities of the designs in [3], [4] are listed Table III for comparison. It turns out that for CRC-12 and CRC-16, the $\mathbf{T}$ matrices computed from the optimal $\mathbf{T}^{-1}$ targeting at minimizing the row weights of $\mathbf{T}^{-1} \times \mathbf{B}_p$ in our proposed approach are the same as the optimal $\mathbf{T}$ found in [4] for minimizing the total number of '1's in the three matrices. The overall CPDs provided in [4] do not include the contribution of the XOR adding up the $\mathbf{A}_{pT}$ and $\mathbf{B}_{pT}$ multiplication results. Also there are typos in the numbers of XOR gates provided in [4], and no detail was disclosed about the adopted SS scheme. The gate count and CPD vary with the SS scheme. Our proposed SS scheme is applied to the matrices specified in [4] to derive the corresponding XOR gate counts and CPDs listed in Table III. The gate counts from our SS are smaller than those provided in [4] for most of the matrices. In [3], the XOR gate count is derived as the sum of the row weights minus one without applying SS. The XOR gate counts listed in Table III for [3] are derived by applying our SS scheme.

The XOR gate counts are also normalized with regard to those of the design in [4] in Table III. Our design has the same CPD, same or just one more gate than that in [4] for CRCs with length 12 and 16. For CRC-32, our approach is able to reduce the gate count by 8%. More importantly, our design enables substantial power reduction compared to [4]. In modern communication systems, the size of a data sector is usually at least a few kilo bits long. Hence, the $\mathbf{T}$ matrix multiplication in Fig. 2 is only active for at most a few percent of the clock cycles. On the other hand, the $\mathbf{B}_{pT}$ and $\mathbf{A}_{pT}$ multiplications are running in almost every clock cycle. Hence, the gate counts in these components provide a good comparison on the average power consumption. As calculated

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCSII.2018.2860934, IEEE Transactions on Circuits and Systems II: Express Briefs

TABLE III
COMPARISONS OF PARALLEL LFSRS BASED ON STATE-SPACE TRANSFORMATIONS

	design	$\mathbf{A}_{pT}$			$\mathbf{B}_{pT}$			$\mathbf{T}$			Total			XORs active every clock (normalized)
		'1'	XOR	CPD	'1'	XOR	CPD	'1'	XOR	CPD	'1'	XOR (normalized)	CPD	
CRC-12	[3]	20	8	1	54	25	3	46	23	3	120	68 (1.36)	4	1.16
	[4]	29	15	3	25	12	4	23	11	2	77	50 (1)	5	1
	proposed	29	15	3	25	12	4	23	11	2	77	50 (1)	5	1
CRC-16	[3]	18	2	1	80	40	4	90	40	4	188	98 (1.48)	5	1.16
	[4]	35	18	2	33	16	4	32	16	2	100	66 (1)	5	1
	proposed	35	18	2	33	16	4	32	16	2	100	66 (1)	5	1
SDLC	[3]	18	2	1	106	53	4	102	52	4	226	123 (1.37)	5	0.95
	[4]	88	42	3	45	17	3	31	15	2	164	90 (1)	4	1
	proposed	67	33	3	38	17	3	52	24	3	157	90 (1)	4	0.88
CRC-16	[3]	18	2	1	80	40	4	92	40	4	190	98 (0.99)	5	0.71
Reverse	[4]	154	45	4	73	21	4	33	17	2	260	99 (1)	5	1
	proposed	109	36	4	32	15	4	117	33	4	258	100 (1.01)	5	0.82
SDLC	[3]	18	2	1	106	50	4	102	49	4	226	117 (1.34)	5	0.97
Reverse	[4]	84	39	4	38	15	3	33	17	2	155	87 (1)	5	1
	proposed	68	34	4	35	15	3	47	23	3	150	88 (1.01)	5	0.93
CRC-32	[3]	45	13	1	447	218	5	436	221	5	928	484(1.02)	6	0.57
	[4]	414	211	5	425	216	5	49	17	2	888	476 (1)	6	1
	proposed	332	169	5	173	107	4	266	128	5	771	436 (0.92)	6	0.67

TABLE IV
COMPARISONS OF PARALLEL LFSR ARCHITECTURES

	design	XOR	DE	CPD	ATP (normalized)	active ATP (normalized)
CRC-12	[6]	103	24	4	1	1
	proposed	50	12	5	0.61	0.51
CRC-16	[6]	94	32	5	1	1
	proposed	66	16	6	0.76	0.63
SDLC	[6]	97	32	3	1	1
	proposed	90	16	4	1.05	0.83
CRC-16	[6]	82	32	4	1	1
reverse	proposed	100	16	5	1.19	0.88
SDLC	[6]	90	32	4	1	1
reverse	proposed	88	16	5	1.01	0.81
CRC-32	[6]	675	64	5	1	1
	proposed	436	32	6	0.75	0.55

in Table III, our proposed design achieves significant power reduction for SDLC, CRC-16 reverse, and CRC-32.

Between the recursive filter-interpreted LFSR designs [5], [6], the one in [6] has lower gate count and shorter CPD. It is compared with our design in Table IV. In this design, the register inputs are generated by more complex logic involving register feedbacks and data inputs. A pipelining stage of width p is inserted to shortern the data path. Hence, it needs p more delay elements (DEs) compared to our design. To compare the efficiency of architectures with different CPDs, the normalized area-time product (ATP) calculated as $(1.5 \cdot DE + XOR) \cdot CPD$ [6] is adopted. In the architecture from [6], almost every logic gate is active in each clock cycle. The active ATP defined according to the area of the XORs and DEs active in each clock cycle is also compared in Table IV. It is evident that our proposed LFSRs achieve substantial improvements on power consumption compared to those filter-interpreted designs and have much higher efficiency for many CRCs.

VI. CONCLUSION

This paper proposed a new transformation for parallel LFSRs. The proposed design effectively shifts the complexity to the transformation matrix multiplication, which is only active in the last clock cycle. As a result, the proposed design achieves substantial reduction on the power consumption without increasing the CPD or total gate count compared to other transformed designs. In addition, a simplified method for computing the CPD of systems adopting SS is developed to better evaluate the complexity of matrix multiplications with CPD constraints. Future work will address efficient parallel design of long LFSRs.

REFERENCES

[1] T.-B. Pei, and C. Zukowski, "High-speed parallel CRC circuits in VLSI," *IEEE Trans. on Commun.*, vol. 40, no. 4, pp. 653-657, Apr. 1992.
[2] J. H. Derby, "High-speed CRC computation using state-space transformations," *Proc. IEEE Global Commun. Conf.*, pp. 166-170, Nov. 2001.
[3] C. Kennedy and A. Reyhani-Masoleh, "High-speed CRC computations using improved state-space transformation," *Proc. IEEE Intl. Conf. Electro/Info. Tech.*, pp. 9-14, 2009.
[4] G. Hu, J. Sha, and Z. Wang, "High-speed parallel LFSR architectures based on improved state-space transformations," *IEEE Trans. on VLSI Syst.* vol. 25, no. 3, pp. 1159-1163, Mar. 2017.
[5] M. Ayinala and K. K. Parhi, "High-speed parallel architectures for linear feedback shift registers," *IEEE Trans. on Signal Process.*, vol. 59, no. 9, pp. 4459-4469, Sep. 2011
[6] J. Jung, *et. al.*, "Efficient parallel architecture for linear feedback shift registers," *IEEE Trans. on Circuits and Syst.-II*, vol. 62, no. 11, pp. 1068-1072, Nov. 2015.
[7] X. Zhang and K. K. Parhi, "High-speed architectures for parallel long BCH encoders," *IEEE Trans. on VLSI Syst.*, vol. 13, no. 7, pp. 872-877, Jul. 2005.
[8] C. Cheng and K. K. Parhi, "High-speed parallel CRC implementation based on unfolding, pipelining, and retiming," *IEEE Trans. on Circuits and Syst.-II*, vol. 53, no. 10, pp. 1017 - 1021, Oct. 2006.
[9] N. Chen and Z. Yan, "High-performance designs of AES transformations," *Proc. IEEE Intl. Symp. Circuits and Syst.*, pp. 2906-2909, May 2009.
[10] N. Wu, *et. al.*, "Improving common subexpression elimination algorithm with a new gate-level delay computing method," *Proc. World Congress on Engr. and Computer Science*, Oct. 2013.