High-Speed CRC Computation Using State-Space Transformations

Jeff H. Derby
Communications Research & Development Center
IBM Corporation
Research Triangle Park, NC 27709

Abstract – Parallelization of the linear-feedback shift register used to compute the CRC has long been recognized as a way to increase throughput. In all applications of this technique reported previously, the achievable increase in throughput is limited by an increase in the circuit complexity within the feedback loop; for a circuit that processes *M* bits of the input sequence in parallel, the throughput increase, or speed-up, appears to be asymptotically limited to *M*/2.

In this paper, we develop a state-space transformation for the *M*-bits-at-a-time CRC system that reduces the complexity of its feedback loop to exactly that of the original bit-at-a-time system. This simplification comes at the cost of increased circuit complexity outside the feedback loop; however, these blocks can be pipelined. Thus the transformed system can achieve a full speed-up factor of *M* compared to the bit-at-a-time system.

The transformation introduced in this paper is general in that it is valid in any field. It can thus be applied to encoders for cyclic codes over arbitrary finite fields that process *M* elements of an input sequence in parallel.

I. Introduction

Cyclic codes have found wide use in communication systems because of their error detection and correction capabilities. Perhaps the simplest and most common application of a cyclic code is the so-called cyclic redundancy check, or CRC. With binary data sequences represented by polynomials over the finite field *GF*(2), the field of integers modulo 2, the CRC is obtained essentially as the remainder resulting from the division of the polynomial representing the input data sequence by the generator polynomial of the CRC. The error detection properties of the CRC depend on the characteristics of polynomials over *GF*(2) and are well understood (see [1], for example).

The usual reference systems for polynomial division over *GF*(2), and thus for computing the CRC, employ shift registers with feedback (see, for example, Sec. 6.2 of [2]). Fig. 1 shows one such system, generalized to be applicable to division in any field. Let $u(n)$, for $n = 0, 1, \ldots, N-1$, be a sequence of length *N*. The polynomial representation of $u(n)$ is

$$U(z) = \sum_{n=0}^{N-1} u(n) z^{N-1-n} \qquad (1)$$

Because the CRC code is systematic, what is needed for the standard *K*-bit CRC, for example as defined in ISO 3309 [10], is the remainder resulting from the division of $z^K U(z)$ by the generator polynomial. The generator polynomial is

$$G(z) = \sum_{k=0}^{K} g_k z^k \qquad (2)$$

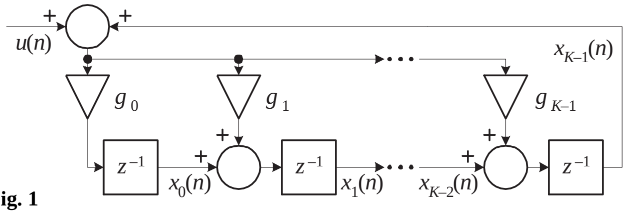

Fig. 1

The blocks in the figure labeled z^{-1} are the delay elements that make up the shift register; for, say, the *k*th delay element (or register element), the output is $x_k(n)$ while the input is $x_k(n+1)$. The *K*-dimensional vector $\mathbf{x}(n)$ with

$$\mathbf{x}(n) = [x_0(n)\ x_1(n)\ \ldots\ x_{K-1}(n)]^T \qquad (3)$$

is the state vector of the system; as usual, the superscript '*T*' indicates transpose. It is well known that at time $n = N$ the elements of this vector are the coefficients of the remainder polynomial resulting from the division of $z^K U(z)$ by $G(z)$. In other words, writing $z^K U(z) = Q(z)G(z) + R(z)$, where $Q(z)$ is the quotient polynomial with degree no greater than $N-K-1$, and $R(z)$ is the remainder polynomial with degree no greater than $K-1$, we have

$$R(z) = \sum_{k=0}^{K-1} r_k z^k; \quad r_k = x_k(N); \quad k = 0, \ldots, K-1 \qquad (4)$$

Of course, for computation of the CRC in *GF*(2), the summing elements are exclusive-or (XOR) gates performing modulo-2 addition, the negative signs are superfluous (because any element in *GF*(2) is its own additive inverse), and the gain elements are either open circuits or short circuits, e.g. for g_k equal to 0 or 1, respectively; and clearly, the elements of the input data sequence and the state vector are taken from *GF*(2).

In computing the CRC, the system in Fig. 1 processes the input data sequence one bit at a time. The throughput of this system is limited by the net propagation delay around the feedback loop, which consists of two XORs plus the register; if the net effective delay is t_d, then the maximum throughput is $1/t_d$. It was realized early on that increased throughput could be achieved by modifying the system in Fig. 1 to process some number of bits (i.e. elements of the input sequence $u(n)$) in parallel. The appropriate serial-to-parallel conversion was apparently first described by Hsiao and Sih [3] and was first applied to CRC calculation by Patel [4], with a number of variations reported more recently, for example in [6]–[9]; software implementations of this technique have also been described, e.g. in [5]. If the net propagation delay around the feedback loop in the system modified to compute the CRC by processing *M* bits at a time were to remain at t_d, then the modified system would have a maximum throughput of M/t_d; i.e., the modified system would have a *speed-up* factor of *M*. In [7], Pei and Zukowski noted that the maximum speed-up factor achievable for the usual *M*-bits-at-a-time systems for CRC computation is only about 0.5*M*, because of the increased circuit complexity in the feedback loop compared to the system in Fig. 1. The results in [7] apply to all of the referenced techniques, with the possible exception of that described by Glaise in [9]. The technique in [9] differs from others in that it employs division by a very high degree polynomial that is itself divisible by the CRC's generator polyno-

0-7803-7206-9/01/$17.00 © 2001 IEEE

mial as an intermediate step. The speed-up achievable using this technique is not completely clear from its description in [9]; however, indications are that it is closer to the bound of 0.5*M* given in [7] than it would be to *M*. In fact, all the referenced techniques, including that in [9], share a common property, namely that the state vector contains the coefficients of the remainder polynomial. This property limits the achievable speed-up because it results in an increase with *M* of the circuit complexity inside the feedback loop.

The speed-up that is achievable with parallel CRC computation remains an important issue, especially with speeds of both local-area network and wide-area network interfaces reaching and even exceeding 10Gbits/s. Thus the question of whether it is possible to realize a speed-up factor greater that 0.5*M* or perhaps even equal to *M* remains significant. This paper shows that by applying an appropriate linear transformation to the state vector in the usual parallel CRC formulation, the circuit complexity inside the feedback loop can be reduced to that of the bit-at-a-time system in Fig. 1. Simplification within the feedback loop comes at the expense of complexity outside the feedback loop. However, the blocks that are required outside the feedback loop can be pipelined, and the pipelining can be dimensioned such that the time required for each stage is no greater than that required within the bit-at-a-time feedback loop in Fig. 1. Thus, a speed-up factor of *M* is in fact achievable.

Section II of the paper provides state-space descriptions of the bit-at-a-time system in Fig. 1 and of the parallel CRC system described in [7] and related references. In Section III, the transformation is developed that reduces the complexity of the feedback loop in the parallel CRC system to that of the bit-at-a-time system. Section IV describes the pipelining used for the blocks introduced by the state-space transformation outside the feedback loop. Section V presents an example, and Section VI then discusses the overall circuit complexity of the transformed system.

Finally, while the focus of this paper is CRC computation, the state-space transformation is developed in the context of operation in an arbitrary finite field *GF*(*q*). Thus the transformation can be applied to cyclic encoders and other systems like that in Fig. 1 operating over any finite field *GF*(*q*).

II. State-Space Description of the CRC

The system shown in Fig. 1 can be described by the vector state equation

$$\mathbf{x}(n+1) = \mathbf{A}\mathbf{x}(n) + \mathbf{b}\mathbf{u}(n); \qquad n \ge 0 \qquad (5)$$

with initial state $\mathbf{x}(0) = \mathbf{x}_o$. The *K*-dimensional state vector $\mathbf{x}(n)$ is given in (3), **A** is the $K \times K$ matrix

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 & \dots & 0 & -g_0 \\ 1 & 0 & 0 & \dots & 0 & -g_1 \\ 0 & 1 & 0 & \dots & 0 & -g_2 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 1 & -g_{K-1} \end{bmatrix} \qquad (6)$$

and the $K \times 1$ matrix **b** is

$$\mathbf{b} = [-g_0 \ -g_1 \ \dots \ -g_{K-1}]^T \qquad (7)$$

The desired output of the system is the remainder of the polynomial division that it computes, i.e. the remainder resulting from the division of $z^K U(z)$ by $G(z)$. Because this remainder is contained in the state vector itself at $n = N$, as in (4), the system output is identical to the state vector. One could call the output vector $\mathbf{y}(n)$ and add the output equation

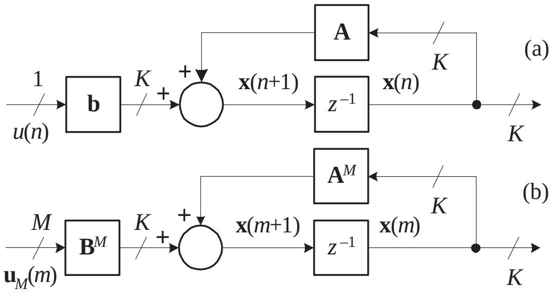

Fig. 2

$\mathbf{y}(n) = \mathbf{C}\mathbf{x}(n)$ to the state equation (5), with **C** equal to the $K \times K$ identity matrix. Note also that the coefficients of the generator polynomial $G(z)$ appear in the right-hand column of the matrix **A**. In fact, this matrix is the *companion matrix* of the polynomial $G(z)$, and $G(z)$ is the characteristic polynomial of this matrix.

The initial state $\mathbf{x}_o$ depends on the specific definition of the CRC for a given application. For example, ISO 3309 [10] specifies in effect that an initialization polynomial

$$U_o(z) = z^N \sum_{k=0}^{K-1} u_{ok} z^k \qquad (8)$$

be added to $z^K U(z)$ before the division by $G(z)$ is carried out, with $u_{ok} = 1$ for $k = 0, \dots, K-1$. It can be shown that, for the system in Fig. 1, addition of the initialization polynomial is exactly equivalent to setting the initial state $\mathbf{x}_o$ such that its elements satisfy $x_{ok} = u_{ok}$ for each *k*.

A straightforward block-processing approach has generally been employed, for example in [6]–[9], in modifying the system in Fig. 1 so that *M* elements of the input sequence $u(n)$ are handled in parallel. Let the elements of the input sequence $u(n)$ be grouped into blocks of length *M*, so that the input to the block-oriented system is now a vector $\mathbf{u}_M(m)$ with

$$\mathbf{u}_M(m) = [u(mM+M-1) \ u(mM+M-2) \ \dots$$
$$\dots \ u(mM+1) \ u(mM)]^T; \ m = 0, 1, \dots, (N/M)-1 \qquad (9)$$

assuming that *N* is an integral multiple of *M*.[1] It is well known that the state equation (5) can be rewritten as

$$\mathbf{x}(m+1) = \mathbf{A}^M \mathbf{x}(m) + \mathbf{B}_M \mathbf{u}_M(m); \qquad m \ge 0 \qquad (10)$$

where the index *m* is incremented by one for each block of *M* input elements. The matrix $\mathbf{B}_M$ in (10) is a $K \times M$ matrix given by

$$\mathbf{B}_M = \left[\mathbf{b} \ \mathbf{A}\mathbf{b} \ \mathbf{A}^2\mathbf{b} \ \dots \ \mathbf{A}^{M-1}\mathbf{b}\right] \qquad (11)$$

whose columns are found by multiplying the vector **b** in (5) by successively higher powers of **A**. The initial state $\mathbf{x}_o$ is identical to that for the original system of Fig. 1. The output vector remains identical to the state vector, and the coefficients of the desired remainder polynomial are equal to the elements of the state vector at $m = N/M$.

The maximum throughput of the original system in Fig. 1 and of the modified system described by (10) can be compared using the block diagrams of the bit-at-a-time system (5) in Fig. 2(a) and the *M*-bits-at-a-time system (10) in Fig. 2(b). For both systems, the throughput is limited by the delay around the feedback loop. The only difference between the two systems is that between computing the products $\mathbf{A}\mathbf{x}$ and $\mathbf{A}^M\mathbf{x}$. From (6), no row of **A** has more than two non-zero entries; thus, for computation in *GF*(2), the time required for the product $\mathbf{A}\mathbf{x}$ is that for a two-input XOR. Similarly, the time required for the product $\mathbf{A}^M\mathbf{x}$ will depend on the pattern

[1] There are several possible techniques to deal with cases where *N* is not an integral multiple of *M*.

of non-zero entries in the matrix $\mathbf{A}^M$, with the computation of the product implemented using a tree of exclusive-or gates.

As an example, consider using the standard 32-bit CRC with generator polynomial

$$G_{32}(z) = z^{32}+z^{26}+z^{23}+z^{22}+z^{16}+z^{12} \\ + z^{11}+z^{10}+z^{8}+z^{7}+z^{5}+z^{4}+z^{2}+z+1 \qquad (12)$$

with $M = 32$. The maximum number of non-zero entries in $\mathbf{A}^{32}$, starting with the appropriate matrix $\mathbf{A}$, is 17; i.e., the time required would appear to be that for the exclusive-or of 17 terms. According to Pei and Zukowski [7], it is possible to combine terms in the product $\mathbf{A}^M\mathbf{x}$ for this case such that the net delay around the feedback loop is only about twice that for the bit-at-a-time system, with the speed-up factor thus being about $0.5M$. In fact, the results in [7] imply that for the standard generator polynomials such as $G_{32}(z)$ in (12) and values of M up to 32, optimum factorings of the product $\mathbf{A}^M\mathbf{x}$ can be found but with the maximum achievable speed-up factor always limited to about $0.5M$.

It is evident from Figs. 2 that a speed-up factor of M is possible with the structure in Fig. 2(b) if the block labeled $\mathbf{A}^M$ can be replaced by a block whose circuit complexity is no greater than that of the block labeled $\mathbf{A}$ in Fig. 2(a), i.e. if the product $\mathbf{A}^M\mathbf{x}$ can be replaced with a product whose computational complexity is no greater than that of the product $\mathbf{Ax}$ with $\mathbf{A}$ as in (6). In fact, this is indeed possible given certain restrictions on the polynomial $G(z)$ that are always met for the CRC and for cyclic codes in general. The computation and associated circuitry within the feedback loop are simplified by applying a linear transformation to (10). In the transformed system, the state vector no longer contains the coefficients of the remainder polynomial; however, these are readily obtained as the elements of an appropriately defined output vector. Essentially, the linear transformation moves computational and circuit complexity from within the feedback loop to blocks that are outside the loop. Because these blocks are outside the feedback loop, they can be pipelined, and so their added complexity need not be a limiting factor on system throughput.

III. State-Space Transformation for CRC

The theoretical development of the state-space transformation to be presented is general in that it is applicable to the system described by the block state equation (10) over any finite field $GF(q)$; the applications that will then be addressed are specific to $GF(2)$. We start with the block state equation (10) and add an explicit output equation:

$$\mathbf{x}(m+1) = \mathbf{A}^M\mathbf{x}(m)+\mathbf{B}_M\mathbf{u}_M(m); \quad \mathbf{y}(m) = \mathbf{C}_M\mathbf{x}(m) \qquad (13)$$

where $\mathbf{C}_M = \mathbf{I}$, the $K \times K$ identity matrix. The output vector $\mathbf{y}(m)$ is equal to the state vector and contains the remainder coefficients at $m = N/M$.

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

$$\mathbf{x}(m) = \mathbf{T}\mathbf{x}_t(m) \qquad (14)$$

Note that $\mathbf{T}$ and its inverse are both matrices over the appropriate field $GF(q)$. Given $\mathbf{T}$ and its inverse, the block state equations (13) are rewritten so that the evolution of the output sequence $\mathbf{y}(m)$ based on the block input sequence $\mathbf{u}_M(m)$ is expressed in terms of the evolution of the transformed state vector $\mathbf{x}_t(m)$, as follows:

$$\mathbf{x}_t(m+1) = \mathbf{A}_{Mt}\mathbf{x}_t(m)+\mathbf{B}_{Mt}\mathbf{u}_M(m); \; \mathbf{y}(m) = \mathbf{C}_{Mt}\mathbf{x}_t(m) \qquad (15)$$

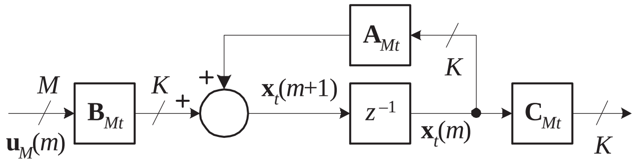

Fig. 3

where

$$\mathbf{A}_{Mt} = \mathbf{T}^{-1}\mathbf{A}^M\mathbf{T}\;; \quad \mathbf{B}_{Mt} = \mathbf{T}^{-1}\mathbf{B}_M\;; \quad \mathbf{C}_{Mt} = \mathbf{T} \qquad (16)$$

and the initial state is $\mathbf{x}_t(0) = \mathbf{T}^{-1}\mathbf{x}_o$. The coefficients of the desired remainder polynomial are now equal to $y_k(N/M)$, the elements of the output vector at $m = N/M$. A block diagram of the system described by (15) is shown in Fig. 3. It is essentially the same as what is shown in Fig. 2(b), except that the block labeled $\mathbf{C}_{Mt}$ associated with the output equation is now included explicitly. One can see from the figure that if $\mathbf{T}$ can be chosen such that $\mathbf{A}_{Mt}$ is a companion matrix, then the delay around the feedback loop for the transformed system will be the same as that for the original bit-at-a-time system, and a speed-up factor of M should be realized. From the relationship between $\mathbf{A}^M$ and $\mathbf{A}_{Mt}$ in (16), there exists $\mathbf{T}$ such that $\mathbf{A}_{Mt}$ is a companion matrix if and only if $\mathbf{A}^M$ is similar to a companion matrix. It can be shown [13] that $\mathbf{A}^M$ is in fact similar to a companion matrix, except for certain specific values of M, if the roots of the characteristic polynomial $G(z)$ of $\mathbf{A}$ are distinct in its splitting field, a condition that is met for all polynomials that are useful as generator polynomials of cyclic codes. For example, for $G_{32}(z)$ in (12), the smallest value of M for which $\mathbf{A}^M$ is not similar to a companion matrix is $M = 2^{16}+1$.

Given that $\mathbf{A}^M$ is similar to a companion matrix, it remains to construct $\mathbf{A}_{Mt}$ as the appropriate companion matrix and then to find a matrix $\mathbf{T}$ that represents the similarity transformation in (16). Construction of $\mathbf{A}_{Mt}$ is simple, given the structure of companion matrices and the fact that similar matrices have the same characteristic polynomial. Let $H(z)$ be the characteristic polynomial of $\mathbf{A}^M$, with

$$H(z) = \sum_{k=0}^{K} h_k z^k \qquad (17)$$

Then $\mathbf{A}_{Mt}$ has exactly the same form as $\mathbf{A}$ in (6), but with the coefficients of $H(z)$ replacing those of $G(z)$. Given that $\mathbf{A}^M$ and $\mathbf{A}_{Mt}$ are similar, there must exist a nonsingular matrix $\mathbf{T}$ such that $\mathbf{A}_{Mt} = \mathbf{T}^{-1}\mathbf{A}^M\mathbf{T}$. In general, the matrix $\mathbf{T}$ satisfying this relation is not unique, and there are several ways to find an appropriate matrix. The approach employed here, which is similar to one sometimes used in the design of control systems (see, e.g., Sec. 5-12 of [12]), employs a cyclic vector of the transformation represented by the matrix $\mathbf{A}^M$. Select an arbitrary vector $\mathbf{b}_1$, subject only to the condition that the vectors $\mathbf{A}^{kM}\mathbf{b}_1$, for $k = 0, 1, \ldots K-1$, are linearly independent; this condition is met if there is no prime factor $H_i(z)$ of $H(z)$ such that $H_i(\mathbf{A})\mathbf{b}_1 = 0$ (see [11], p.202). Now, let $\mathbf{T}$ be the matrix whose columns are these vectors, i.e.

$$\mathbf{T} = \left[\mathbf{b}_1 \; \mathbf{A}^M\mathbf{b}_1 \; \mathbf{A}^{2M}\mathbf{b}_1 \; \ldots \; \mathbf{A}^{(K-1)M}\mathbf{b}_1\right] \qquad (18)$$

Clearly, $\mathbf{T}$ is nonsingular. Moreover, this matrix $\mathbf{T}$ implements the desired similarity relationship[2], i.e. that $\mathbf{A}_{Mt} = \mathbf{T}^{-1}\mathbf{A}^M\mathbf{T}$ for the matrix $\mathbf{T}$ defined in (18).

Having completed construction of a similarity transforma-

[2] This follows from a constructive proof that a matrix representing a transformation that has a cyclic vector is similar to a companion matrix (for example, see Theorem 2 in Chapter 7 of [11]).

tion between $\mathbf{A}^M$ and $\mathbf{A}_{Mt}$, with $\mathbf{A}_{Mt}$ a companion matrix, we can look at the detailed structure of the feedback loop in Fig. 3. For convenience, define $\mathbf{v}(m) \equiv \mathbf{B}_{Mt}\mathbf{u}_M(m)$. Using this definition in (15), we have an implementation of the core of the feedback loop for the transformed system shown in Fig.

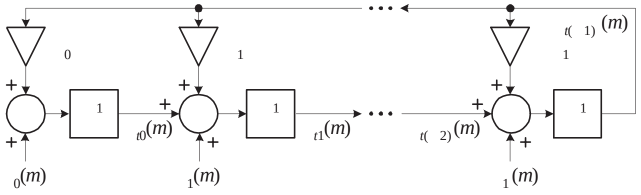

Fig. 4

4. Comparison of Fig. 4 with Fig. 1 confirms the intent of the transformation, namely that the circuit complexity within the feedback loop for the transformed *M*-at-a-time system is equivalent to that for the original one-at-a-time system. In other words, the full speed-up by a factor of *M* is indeed achieved.

IV. Pipelining Outside the Feedback Loop

As has been seen, the transformation developed in the preceding section permits operation with the maximum speed-up factor by shifting computational and circuit complexity from inside the feedback loop to outside the loop. The blocks outside the loop, associated with the products $\mathbf{v} = \mathbf{B}_{Mt}\mathbf{u}_M$ and $\mathbf{y} = \mathbf{C}_{Mt}\mathbf{x}_t$ can be pipelined. Pipelining is a well known technique. This section will clarify how pipelining can be applied in the system shown in Fig. 3, to confirm that the desired throughput is achieved and also as a means of assessing the circuit complexity of the overall system.

For the speed-up factor of *M* to be achieved throughout the system in Fig. 3, the blocks that implement the two matrix products outside the feedback loop must be pipelined into stages with the per-stage circuit complexity no greater than that inside the feedback loop. For the general case of operation in *GF*(*q*) shown in the figures, the processing time around the loop is associated with scaling by a constant, three-input addition, and setup of the delay element; for operation in *GF*(2), e.g. for CRC computation, the scaling by a constant becomes a null function and the addition is an exclusive-or. Consider first the pipelining of a generic multiplication of a $J \times 1$ matrix $\mathbf{s}(m)$ by an $I \times J$ matrix whose elements are β_{ij}, with the result being a $I \times 1$ matrix $\mathbf{w}(m)$; here the indices *i* and *j* run from 0 to $(I\;\;\;1)$ and $(J\;\;\;1)$, respectively. The matrix product is written out as

$$w_i(m) = \sum_{j=0}^{J\;\;\;1} \beta_{ij}s_j(m)\,; \qquad i = 0, \ldots, I\;\;\;1 \tag{19}$$

Note that the pipeline delay is ignored in (19). If there are *L* pipeline stages, then $\mathbf{w}(m)$ appears *L* sample times after $\mathbf{s}(m)$. A straightforward pipelining of this computation that meets the limit on per-stage processing can be constructed by partitioning the sum of products in (19) so that at each stage the contributions of two inputs are added for each output. More precisely:

$$\begin{aligned} w_{1i}(m) &= \beta_{i0}s_0(m) + \beta_{i1}s_1(m) + \beta_{i2}s_2(m) \\ w_{li}(m) &= \beta_{i(2l\;\;\;1)}s_{2l\;\;\;1}(m) + \beta_{i(2l)}s_{2l}(m) + w_{(l\;\;\;1)i}(m) \\ w_i(m) &= \beta_{i(J\;\;\;1)}s_{2L\;\;\;1}(m) + w_{(L\;\;\;1)i}(m) \end{aligned} \tag{20}$$

where the middle equation in (20) runs for $l = 2, \ldots, L\;\;\;1$, and $\mathbf{w}_l$ is the output of the *l*th stage. The computation for each of the *L* stages consists of no more than a scaling by a constant, a three-input addition, and a delay element. It is assumed for purposes of using this partitioning of the matrix product as a reference that all the entries in the matrix are nonzero and that *J* is even, with $L = J/2$. The presence of only a single addition in the *L*th stage results from taking *J* to be even.

Now, let L_B be the number of stages required for the computation of $\mathbf{B}_{Mt}\mathbf{u}_M$, and let L_C be the number of stages required for the computation of $\mathbf{C}_{Mt}\mathbf{x}_t$; note that L_B need be no greater than $M/2$ and L_C need be no greater than $\;\;/2$ for the pipelined structure described above. This generic structure can be used for each of these matrix products, with the following identifications in (19). First, for the product $\mathbf{v} = \mathbf{B}_{Mt}\mathbf{u}_M$, identify $\mathbf{s} = \mathbf{u}_M$, $\mathbf{w}_L = \mathbf{v}$, $I = \;\;$, $J = M$, $L = L_B$, and β_{ij} is the *ij*th element of $\mathbf{B}_{Mt}$. Then, for the product $\mathbf{y} = \mathbf{C}_{Mt}\mathbf{x}_t$, identify $\mathbf{s} = \mathbf{x}_t$, $\mathbf{w}_L = \mathbf{y}$, $I = \;\;$, $J = \;\;$, $L = L_C$, and β_{ij} is the *ij*th element of $\mathbf{C}_{Mt}$.

Clearly, the approach described above is just one of several possible partitions of the matrix multiplication in (19) such that the per-stage processing time is within the desired limit. For example, there are structures that require fewer pipeline stages, although with no net reduction in hardware complexity. Alternatively, when the computation is in *GF*(2) so that all the coefficients β_{ij} are either 0 or 1, there are simple ways to group terms in (19) such that the number of additions (i.e., exclusive-ors) required for each element of the output vector is minimized; this technique was applied to the basic block state equation (10) without pipelining in [7], and could be applied here in the pipelined case.

V. An Example

Consider the 32-bit CRC defined by the generator polynomial $G_{32}(\;\;)$ given in (12), to be computed taking 32 bits of the input sequence at a time (i.e. with $M = 32$). From (6), (7), and (12) we have **A** and **b** for this example, and $\mathbf{A}^M$ is easily computed. With $\;\;=M$ as is the case in this example, $\mathbf{B}_M = \mathbf{A}^M$. Also in this example *M* is a power of 2, and for any polynomial $f(\;\;)$ over *GF*(2) we have $f(\;\;^2) = [f(\;\;)]^2$. Thus **A** and $\mathbf{A}^M$ here have the same characteristic polynomial, and so $\mathbf{A}_{Mt} = \mathbf{A}$. Now, for simplicity take $\mathbf{b}_1 = [1\;0\;0\;\ldots\;0]^T$. Note that because $G_{32}(\;\;)$ is irreducible, we know from the comment immediately preceding (18) that any non-zero vector may be selected for $\mathbf{b}_1$ in this case. The resulting matrix **T** and its inverse are shown in Fig. 5.

VI. Discussion

The circuit complexity for the transformed system in Fig. 3 is concentrated in the blocks associated with the products $\mathbf{v} = \mathbf{B}_{Mt}\mathbf{u}_M$ and $\mathbf{y} = \mathbf{C}_{Mt}\mathbf{x}_t$. In Section IV we described a generic approach, valid in any field, to the pipelined implementation of these blocks. We will now look at the complexity of these blocks again, specifically with respect to CRC computation in *GF*(2). Recall that for arithmetic in *GF*(2) the gain elements are null functions (each is either a short circuit if the coefficient is 1 or an open circuit if the coefficient is 0), the summing elements are XOR gates, and permitted processing within each stage of the pipeline is equivalent to two two-input XOR gates or a single three-input XOR gate. On this basis, the circuit complexity of the transformed system in Fig. 3 can be compared with that of the standard parallel approach to CRC computation in Fig. 2(b).

Let L_B be the number of stages required for the computation of $\mathbf{B}_{Mt}\mathbf{u}_M$, and let L_C be the number of stages required for the computation of $\mathbf{C}_{Mt}\mathbf{x}_t$. In addition, let D_B be the maximum number of 1s in any row of $\mathbf{B}_{Mt}$, and let D_C be the maximum number of 1s in any row of $\mathbf{C}_{Mt}$. It is easy to see that L_B need never be greater than $\lceil((D_B - 1)/2)\rceil$, and that L_C need never be greater than $\lceil((D_C - 1)/2)\rceil$, where $\lceil x \rceil$ is the smallest integer greater than or equal to *x*. These upper bounds correspond to an implementation providing maximum throughput using two-input XOR gates with no grouping of terms. For the example in Section V, we find that $D_B = 22$ and $D_C = 21$, so that the upper bound for L_B is 11 while that for L_C is 10. Continuing with the assumption that only two-input XOR gates are used with no grouping of terms, the number of gates required for the computation of $\mathbf{B}_{Mt}\mathbf{u}_M$ is equal to the sum over all the rows of $\mathbf{B}_{Mt}$ of one less than the number of 1s in each row; for the example in Section V, this number is 466. The number of gates required for the computation of $\mathbf{C}_{Mt}\mathbf{x}_t$ is found in the same way; for the example in Section V, this number is 456. Additionally, each product requires the appropriate number of pipeline registers. In comparison, the number of gates required for the original block state equation (10) computing the 32-bit CRC based on $G_{32}(z)$ and taking 32 bits at a time (i.e., corresponding to the example in Section V), assuming that only two-input XOR gates are used with no grouping of terms, is 420. For this latter case, [7] and several other references have indicated that by appropriately grouping terms, the depth of the XOR tree and the total number of XOR gates can be reduced. Of course, the same techniques can be employed here for the pipelined matrix multiplications, with a reduction in the number of XOR gates and also in the number of pipeline stages. It may also be possible, by constructing the similarity transformation using a different choice for $\mathbf{b}_1$, to find cases with smaller values for D_B and D_C.

Clearly, there is a trade-off between overall circuit complexity and the ability to maximize throughput; after all, the throughput of the transformed system is at least twice that of the original system. It should also be noted that the optimum grouping of terms employed in [7] and elsewhere is mandatory to achieve speed-up factors that even approach $M/2$ for the standard *M*-bits-at-a-time systems. In contrast, the speed-up factor of *M* is an inherent property of the transformed system, with the only motivation to group terms in the pipelined blocks being the possible reduction in the number of gates required. In fact, the regularity of the structure that implements (20) with no grouping of terms is itself a useful property from an implementation perspective.

Finally, there is a simplification that can be applied to the transformed system when it is used for CRC checking. The CRC check is carried out by comparing the computed CRC of a received sequence with a polynomial of degree $K-1$ specified *a priori*. From the output equation in (15), this is equivalent to comparing the output vector $\mathbf{y}$ after the last bit of the received sequence has been processed with a vector $\mathbf{r}_{\mathrm{ref}}$ that is known *a priori*; but this is exactly equivalent to comparing the state vector $\mathbf{x}_t$ with a vector $\mathbf{C}_{Mt}^{-1}\mathbf{r}_{\mathrm{ref}}$ that can also be determined *a priori*. Thus for CRC checking the matrix multiplication $\mathbf{C}_{Mt}\mathbf{x}_t$ can be eliminated.

VII. Conclusion

In this paper, a state-space transformation was developed for the *M*-bits-at-a-time CRC system that reduces the complexity of its feedback loop to exactly that of a one-bit-at-a-time system. This simplification comes at the cost of increased circuit complexity outside the feedback loop; however, these blocks can be pipelined. Thus the maximum throughput of the transformed *M*-bits-at-a-time system is *M* times the maximum throughout of the one-bit-at-a-time system. The transformation introduced in this paper is general in that it is valid in any field. It can thus be applied to encoders for cyclic codes over arbitrary finite fields that process *M* elements of an input sequence in parallel.

References

[1] W. W. Peterson and D. T. Brown, “Cyclic codes for error detection”, *Proc. IRE*, vol. 49, pp. 228-235, Jan. 1961.

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

[3] M. Y. Hsiao and K. Y. Sih, “Serial-to-parallel transformation of linear-feedback shift-register circuits”, *IEEE Trans. Electronic Computers,* vol. EC-13, pp. 738-740, Dec. 1964.

[4] A. M. Patel, “A multi-channel CRC register”, in *AFIPS Conference Proceedings*, vol. 38, pp. 11-14, Spring 1971.

[5] A. Perez, “Byte-wise CRC calculations”, *IEEE Micro,* vol. 3, pp. 40-50, June 1983.

[6] G. Albertengo and R. Sisto, “Parallel CRC generation”, *IEEE Micro,* vol. 10, pp. 63-71, Oct. 1990.

[7] T-B. Pei and C. Zukowski, “High-speed parallel CRC circuits in VLSI”, *IEEE Trans. Commun.*, vol. 40, pp. 653-657, April 1992.

[8] S. L. Ng and B. Dewar, “Parallel realization of the ATM cell header CRC”, *Computer Commun.*, vol. 19, pp. 257-263, March 1996.

[9] 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.

[10] ISO 3309, *Information processing systems — Data communication — High-level data link control procedures — Frame structure,* 1984.

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

[12] J. J. D’Azzo and C. H. Houpis, *Linear Control System Analysis and Design.* New York: McGraw-Hill, 1981.

[13] J. H. Derby, “Parallel encoders for cyclic codes using state-space transformations”, to be submitted to *IEEE Trans. Commun.*

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

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

Fig. 5