High-speed Architectures for Parallel Long BCH Encoders *

Xinmiao Zhang and Keshab K. Parhi
Department of Electrical & Computer Engineering
University of Minnesota, Minneapolis, MN 55414 U.S.A
{jennizh, parhi}@ece.umn.edu

ABSTRACT

Long BCH codes are used as the outer error-correcting code in the second generation of Digital Video Broadcasting Standard from the European Telecommunications Standard Institute. These codes can achieve around 0.6dB additional coding gain over Reed-Solomon codes with similar codeword length and code rate in long-haul optical communication systems. BCH encoders are conventionally implemented by a linear feedback shift register architecture. High-speed applications of BCH codes require parallel implementations of encoders. In addition, long BCH encoders suffer from the effect of large fanout. In this paper, novel architectures are proposed to reduce the achievable minimum clock period of long BCH encoders after the fanout bottleneck has been eliminated. For an (8191, 7684) BCH code, compared to the original 32-parallel BCH encoder architecture without fanout bottleneck, the proposed architectures can achieve a speedup of over 100%.

Categories & Subject Descriptors

B.2.4:[High-Speed Arithmetic]:Algorithms
E.4:[Coding and Information Theory]: Error Control Codes

General Terms

Algorithms, Design, Performance

Key Words

BCH, encoder, linear feedback shift register, fanout, unfolding, parallel processing, retiming, critical loop, iteration bound, generator polynomial

1. INTRODUCTION

BCH codes are among the most extensively used error correcting codes in modern communication systems. Long BCH codes of length 32400-bit or longer are used as the outer forward error-correcting code in the second generation Digital Video Broadcasting (DVB-S2) Standard from the European Telecommunications Standard Institute (ETSI). In addition, compared to Reed-Solomon codes, BCH codes can achieve around additional 0.6dB coding gain over AWGN channel with similar rate and codeword length. High-rate Reed-Solomon codes of length 255 or longer have broad applications, such as in long-haul optical communication systems, magnetic recording systems and digital communications. Therefore, long BCH codes are also of great interest.

The encoders of BCH codes are conventionally implemented by a linear feedback shift register (LFSR) architecture. While such an architecture can be clocked at very high frequency, it suffers from the serial-in and serial-out limitation. In high-speed applications, such as optical communication systems, where throughput in the range of 10-40Gbps is usually desired, the clock frequency of such LFSR-based encoders can not keep up with the data transmission rate, and thus parallel processing must be employed. Meanwhile, long BCH encoders face another problem. Due to the large number of non-zero coefficients in the generator polynomials, some of the XOR gates in the LFSR architecture have large fanout, which can slow down the encoders significantly.

Various parallel LFSR architectures have been proposed in the past [1, 2, 3]. However, none of these have addressed the effect of large fanout in the case of long BCH codes. The fanout bottleneck in parallel long BCH encoders can be eliminated by the approach in [4]. Nevertheless, the speedup of parallel processing is offset by the increased achievable minimum clock period of this approach. In this paper, three novel architectures are proposed to reduce the achievable minimum clock period in unfolded long BCH encoders after the fanout bottleneck has been eliminated. As an example, the proposed architectures can achieve more than 100% speedup over the architecture in [4] for a 32-parallel (8191, 7684) BCH code. Without loss of generality, only binary BCH codes are considered.

The structure of this paper is as follows. Section 2 contains a brief description of the LFSR-based BCH encoder architecture and a prior approach to eliminate the large fanout effect. In Section 3, novel parallel encoder architectures, which can reduce the minimum achievable clock period of long BCH encoders after the large fanout bottleneck has been eliminated, are explained in detail. Section 4 addresses the complexity analysis of the proposed architectures and provides some comparison results. Section 5 provides conclusions.

*This research has been supported by the Army Research Office under grant number DA/DAAD19-01-1-0705.

GLSVLSI'04, April 26–28, 2004, Boston, Massachusetts, USA.

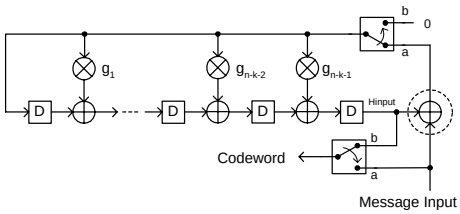

Figure 1: Serial BCH encoder architecture

2. BCH ENCODER ARCHITECTURE

An (n, k) binary BCH code encodes a k-bit message block into an n-bit codeword. Considering a k-bit message (m_{k-1}, m_{k-2}, ..., m_0) ($m_i \in GF(2)$, $0 \leq i \leq k-1$) as the coefficients of a degree $k-1$ message polynomial $m(x)$, and the corresponding n-bit codeword (c_{n-1}, c_{n-2}, ..., c_0) ($c_i \in GF(2)$, $0 \leq i \leq n-1$) as the coefficients of a degree $n-1$ codeword polynomial $c(x)$, the encoding of BCH codes can be performed as $c(x) = m(x)g(x)$, where the degree $n-k$ polynomial $g(x)$ is the generator polynomial of the (n, k) BCH code. However, systematic encoding is usually desired, since in this case the message bits can be read from the codeword directly. The systematic encoding can be implemented by:

$$c(x) = m(x) \cdot x^{n-k} + Rem(m(x) \cdot x^{n-k})_{g(x)}, \qquad (1)$$

where $Rem(a(x))_{b(x)}$ denotes the remainder polynomial of dividing $a(x)$ by $b(x)$.

Assuming $g(x)$ can be expressed as $g(x) = g_{n-k}x^{n-k} + g_{n-k-1}x^{n-k-1} + \ldots + g_0$ ($g_i \in GF(2), 0 \leq i \leq n-k$ and $g_0 = g_{n-k} =' 1'$), a systematic binary (n, k) BCH encoder can be implemented by the architecture illustrated in Fig. 1. For binary BCH codes, each multiplier in Fig. 1 can be replaced by connection or no connection when g_i ($0 \leq i \leq n-k$) is '1' or '0', respectively. Using the encoder architecture in Fig. 1, the whole encoding process takes n clock cycles. During the first k clock cycles, the two switches are connected to the 'a' port, and the k message bits are input to the LFSR serially with most significant bit (MSB) first. Meanwhile, the message bits are also sent to the codeword output to form the systematic part of the codeword. After the k^{th} clock cycle, the $n-k$ delay elements contain $Rem(m(x) \cdot x^{n-k})_{g(x)}$. At this time, the switches are moved to the 'b' port, and the remainder bits are shifted out of the delay elements to the codeword output bit by bit to form the remaining bits of the systematic codeword. As can be observed from Fig. 1, the critical path of this architecture consists of two XOR gates and the output of the right-most XOR gate is the input to all the other XOR gates. In the case of long BCH codes, this architecture may suffer from the large fanout effect on the right-most XOR gate, which can slow down the encoder significantly. In addition, parallel processing needs to be employed when the encoder can not run as fast as the application requirements. Fanout bottleneck also exists in parallel long BCH encoders.

Retiming can always be applied to eliminate the effect of large fanout in serial long BCH encoders. To make notations simple, we refer to the input to the right-most XOR gate in Fig. 1, which is the delayed output of the second XOR gate from the right, as 'Hinput'. Since there is at least one delay element at the Hinput of the right-most XOR gate, and delay elements can be added to the message input, retiming can always be performed along the dotted cutset as shown in Fig. 1 by removing one delay element from each input of the right-most XOR gate and adding one to the output.

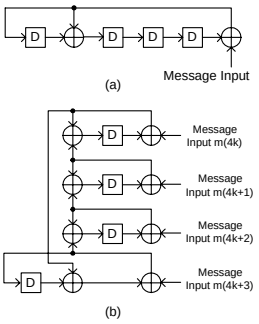

Figure 2: (a) An LFSR example, (b) 4-unfolded version of the LFSR in (a)

Parallel BCH encoder architectures can be derived by unfolding [5]. However, if unfolding is applied directly to Fig. 1, retiming may not be applied in an obvious way to eliminate the large fanout effect in the parallel architecture. The original architecture can be expressed by data flow graphs (DFG), where each of the gates are expressed as nodes, and are connected by paths with delay elements. The J-unfolded architecture makes J copies of each node in the original architecture. However, the total number of delay elements does not change. Assuming node U is connected to node V by a path with w delay elements in the original architecture, in the J-unfolded architecture, the node U_i is connected to $V_{(i+w)\%J}$ by a path with $\lfloor (i+w)/J \rfloor$ delay elements, where $U_i, V_j (0 \leq i, j < J)$ are copies of the node U and V, respectively, and $(i+w)\%J$ is the remainder of dividing $i+w$ by J [5]. It can be derived that if the number of delay elements, w, in a path is less than the unfolding factor J, there will be w corresponding paths with one delay element and the rest $J-w$ paths without any delay elements in the J-unfolded architecture. Assuming the generator polynomial $g(x)$ of a BCH code can be rewritten as:

$$g(x) = x^{t_0} + x^{t_1} + \cdots + x^{t_{s-2}} + 1, \qquad (2)$$

where t_0, t_1, $\cdots$, t_{s-2} are positive integers with $t_0 > t_1 > \cdots t_{s-2}$ and s is the total number of non-zero coefficients in $g(x)$, then there are $t_0 - t_1$ consecutive delay elements at the Hinput of the right-most XOR gate in Fig. 1. In the case of $t_0 - t_1 < J$, there will be at least one input path to the copies of the right-most XOR gate without any delay elements in the J-unfolded architecture. As a result, retiming can not be applied directly to eliminate the large fanout effect. For example, Fig. 2(a) shows the LFSR for a serial (15, 11) BCH encoder with generator polynomial $g(x) = x^4 + x + 1$. In this example, there are three delay elements at the Hinput of the right-most XOR gate. Applying 4-unfolding, the architecture in Fig. 2(b) can be derived. As can be observed, there are 4 copies of the right-most XOR gate, and 3 of them have one delay element at the Hinput while the other 4-3=1 does not have any delay elements at the Hinput. Accordingly, retiming can not be applied around the last XOR gate to eliminate the fanout bottleneck.

In the case of $t_0 - t_1 < J$, the scheme proposed in [4] can be employed to modify the generator polynomial to enable retiming of the right-most XOR gates in the J-unfolded LFSR architecture. $m(x)x^{n-k}$ can be written as:

$$m(x)x^{n-k} = q(x)g(x) + r(x), \qquad (3)$$

where $q(x)$ and $r(x)$ are the quotient polynomial and the re-

mainder polynomial, respectively. Multiplying a polynomial $p(x)$ to both sides of (3), we can get:

$$m(x)p(x)x^{n-k} = q(x)(g(x)p(x)) + r(x)p(x). \quad (4)$$

Since $r(x)$ is the remainder polynomial of the division by $g(x)$, the degree of $r(x)$ is less than that of $g(x)$. It follows that $deg(r(x)p(x)) < deg(g(x)p(x))$, where $deg(f(x))$ denotes the degree of $f(x)$. Hence, $r(x)p(x)$ can be considered as the remainder of dividing $m(x)p(x)x^{n-k}$ by $g(x)p(x)$. According to (4), the proposed architecture in [4] computes $r(x) = Rem(m(x)x^{n-k})_{g(x)}$ by a three-step process:

1) multiply $m(x)$ by $p(x)$;

2) compute the remainder of dividing $m(x)p(x)x^{n-k}$ by $g(x)p(x)$;

3) compute the quotient of dividing the remainder from step 2 by $p(x)$.

The quotient computed from the third step is $r(x)$. The first step can be implemented by an architecture similar to that of an FIR filter, while the other two steps can be implemented by LFSR architectures.

The modified generator polynomial, $g'(x) = p(x)g(x)$, can be written as:

$$g'(x) = x^{t'_0} + x^{t'_1} + \cdots + x^{t'_{s'-2}} + 1, \quad (5)$$

where t'_0, t'_1, $\cdots$, $t'_{s'-2}$ are positive integers with $t'_0 > t'_1 > \cdots > t'_{s'-2}$ and s' is the total number of non-zero coefficients in $g'(x)$. In the case of $t_0 - t_1 < J$ in (2), a $p(x)$ can be found to ensure $t'_0 - t'_1 \geq J$. Accordingly, retiming can be applied to eliminate the effect of large fanout in the J-unfolded LFSR implementing the division by $g'(x)$. Such a $p(x)$ can be computed by clustered look-ahead computation [5]. A second LFSR architecture is also needed to compute the division by $p(x)$ in the third step. It can be observed from the clustered look-ahead computation that the degree of the $p(x)$ for J-unfolding is at most $J-1$. Since the unfolding factor J is much smaller than $n-k$ for long BCH codes, the large fanout effect does not exist in the LFSR implementing the division by $p(x)$. Therefore, the fanout bottleneck can be eliminated by computing the reminder polynomial $r(x)$ through the three-step process introduced in [4]. Further speedup of parallel long BCH encoders can be achieved by the novel architectures proposed in the next section.

3. HIGH-SPEED PARALLEL LONG BCH ENCODERS WITH ELIMINATED FANOUT BOTTLENECK

For an architecture without feedback loops, pipelining can be employed to achieve higher speed. However, the minimum achievable clock period for an architecture consisting of feedback loops is determined by $\lceil T_\infty \rceil$, where T_∞ is the iteration bound of the architecture. Iteration bound is defined as the maximum of all the loop bounds, and the loop with the maximum loop bound is called the critical loop. The loop bound is defined as t/w, where t is the computation time of the loop, and w is the number of delay elements in the loop. In addition, the iteration bound of a J-unfolded architecture is J times the iteration bound of the serial architecture.

In the three-step process of BCH encoding in [4], the first step can be implemented by an architecture similar to that of an FIR filter. Since no feedback loop exists in this serial architecture and the corresponding unfolded architectures,

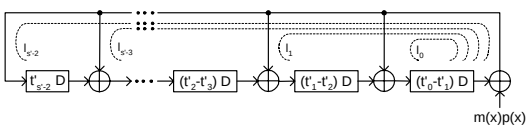

Figure 3: The LFSR architecture for the second step

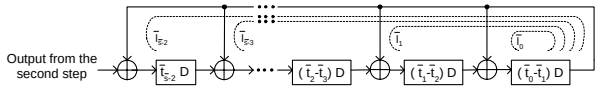

Figure 4: The LFSR architecture for the third step

pipelining can be employed to increase speed. The second and the third steps are implemented by LFSR architectures. Therefore, the minimum achievable clock period of the BCH encoder is determined by the iteration bounds of these two LFSR architectures. Assuming the iteration bounds of the LFSRs for the second and the third steps are T^A_∞ and T^B_∞, the iteration bounds of the J-unfolded architectures of the two LFSRs are JT^A_∞ and JT^B_∞, respectively. It follows that the minimum achievable clock period of the J-unfolded BCH encoder is $\max(\lceil JT^A_\infty \rceil, \lceil JT^B_\infty \rceil)$.

Assuming $g'(x)$ is expressed as in (5), the corresponding LFSR architecture for the second step is illustrated in Fig. 3. As can be observed, the loop bound for loop l_i $(0 \leq i < s'-1)$ in Fig. 3 is $(2+i)T_{XOR}/(t'_0 - t'_{i+1})$, where T_{XOR} is the delay of an XOR gate. For long BCH code, the number of non-zero coefficients in the generator polynomial is around half of the total number. Using the $p(x)$ computed from clustered look-ahead computation, it can be observed that in the modified generator polynomial $g'(x)$, the non-zero coefficients accounts for around $t'_1/2$. Meanwhile, the non-zero coefficients of $g'(x)$ are typically well-distributed, *i.e.* neither long consecutive zeros nor ones exist except the $t'_0 - t'_1 - 1$ zeros after the highest power term. Therefore, the number of XOR gates in loop l_i can be approximated by $2 + (t'_1 - t'_{i+1})/2$. It can be derived that the iteration bound of the LFSR for the second step is at most around $T_{XOR}/2$ for unfolding factor $J \geq 4$. In addition, the larger the unfolding factor J, the smaller the iteration bound of the LFSR implementing the division by $g'(x)$.

Assuming $p(x)$ can be written as $p(x) = x^{\bar{t}_0} + x^{\bar{t}_1} + \cdots + x^{\bar{t}_{\bar{s}-2}} + 1$, where $\bar{t}_0, \bar{t}_1, \cdots, \bar{t}_{\bar{s}-2}$ are positive integers with $\bar{t}_0 > \bar{t}_1 > \cdots > \bar{t}_{\bar{s}-2}$ and $\bar{s}$ is the total number of non-zero coefficients in $p(x)$, the third step of BCH encoding can be implemented by the LFSR architecture shown in Fig. 4. Since the output from the second step does not need to be multiplied by any power of x, it is added to the input of the left-most XOR gate directly. It can be computed that the loop bound of loop $\bar{l}_i$ $(0 \leq i < \bar{s}-1)$ is $(1+i)T_{XOR}/(\bar{t}_0 - \bar{t}_{i+1})$. Due to the short length, $p(x)$ does not have the property that around half of the coefficients are zero. Since $deg(p(x)) \leq J-1$, $\bar{t}_0 - \bar{t}_{i+1}$ is always less than J. However, $t'_0 - t'_{i+1}$ is always larger than J and increases with i. Hence, the iteration bound of the LFSR for the third step is typically larger than that of the LFSR for the second step. Therefore, reducing the iteration bound of the LFSR for the third step is critical to speed up the entire BCH encoder. Particularly, in the case of $t_0 - t_1 = 1$, it can be observed from clustered-look ahead computation that $\bar{t}_0 - \bar{t}_1 = 1$. In this case, the iteration bound of the LFSR for the third step is T_{XOR}. This case is not rare. In fact, around half of the generator polynomials for long BCH codes belong to this case. In addition, since $\bar{t}_0 - \bar{t}_1$ does not change with the unfolding

factor J, the iteration bound of the LFSR for the third step is always T_{XOR} in this case. However, the iteration bound of the LFSR for the second step decreases as J increases. Therefore, reducing the iteration bound of the LFSR for the third step becomes more and more important as J increases.

It is extremely difficult to find a general approach to reduce the iteration bound of the LFSR for the third step of BCH encoding. However, since $p(x)$ is not long, this LFSR can be replaced by architectures free of feedback loops, such that pipelining can be applied. In this case, the minimum achievable clock period is no longer affected by the third step. Three different approaches can be employed to eliminate the feedback loops in the third step. They are explained in detail in the remainder of this section.

3.1 Approach A

Denote the output of the second step by $a(x)$, $r(x) = a(x)/p(x)$ needs to be computed in the third step. Instead of using a second LFSR architecture, the coefficients of $r(x)$ can be computed by a matrix multiplication:

$$\underline{r} = Q\underline{a},$$

where $\underline{r}$ and $\underline{a}$ are the column vectors formed by the coefficients of $r(x)$ and $a(x)$, respectively. Assuming the degree of $p(x)$ computed from clustered look-ahead is z ($z \leq J-1$), $deg(a(x)) \leq deg(g(x)p(x)) - 1 \leq n-k+z-1$. Hence, $a(x)$ has $n-k+z$ coefficients. Similarly, since $deg(r(x)) \leq deg(g(x)) - 1 = n-k-1$, $r(x)$ has $n-k$ coefficients. Therefore, the dimension of Q is $(n-k)\times(n-k+z)$. In addition, since $deg(p(x)) = z$, the terms with the lowest z powers of x in $a(x)$ do not contribute to the quotient of $a(x)/p(x)$. Consequently, the computation of $\underline{r}$ can be simplified as:

$$\underline{r} = Q'_{(n-k)\times(n-k)}\underline{a}', \tag{6}$$

where $\underline{a}' = [a_{n-k+z-1}, a_{n-k+z-2}, \cdots, a_z]^T$ is a vector formed by the coefficients of the terms with the highest $n-k$ powers of x in $a(x)$. Assuming $\underline{r} = [r_{n-k-1}, r_{n-k-2}, \cdots, r_0]^T$ and $p(x) = p_z x^z + p_{z-1}x^{z-1} + \cdots p_0$, such a Q' matrix with entries $q'_{i,j} (0 \leq i,j < n-k)$ can be constructed by the pseudo code listed in Algorithm 1. The polynomial $\tilde{r}(x) = \tilde{r}_{z-1}x^{z-1} + \tilde{r}_{z-2}x^{z-2} + \cdots + \tilde{r}_0$ and $\tilde{q}(x) = \tilde{q}_{n-k-1}x^{n-k-1} + \tilde{q}_{n-k-2}x^{n-k-2} + \cdots + \tilde{q}_0$ are used to keep track of the remainder and quotient polynomial of dividing $x^{n-k+z-1-j}$ by $p(x)$ in each iteration, respectively.

Algorithm 1
initialization:
  set $\tilde{r}_i = p_i$, for $0 \leq i \leq z-1$
  set $\tilde{q}_0 = 1, \tilde{q}_i = 0$ for $1 \leq i \leq n-k-1$
begin construction:
  for j in $n-k-1$ downto 0 do
    $q'_{i,j} = \tilde{q}_{n-k-1-i}$ for $0 \leq i \leq n-k-1$
    $\tilde{q}(x) \Leftarrow x \cdot \tilde{q}(x) + \tilde{r}_{z-1}$
    $\tilde{r}(x) \Leftarrow Rem(x \cdot \tilde{r}(x))_{p(x)}$
  end

From Algorithm 1, It can be observed that the constructed matrix Q' is a lower triangular matrix. The matrix multiplication can be pipelined and operated in a parallel manner. The complexity of implementing this matrix multiplication is in the order of $(n-k)^2$. Due to the large $n-k$ for long BCH codes, this approach is hardware demanding.

3.2 Approach B

Although the scheme proposed in [3] to speed up parallel processing may not be applicable for general BCH codes, their method to compute the remainder polynomial $r(x)$ gives rise to another approach to eliminate the second LFSR in BCH encoders. $m(x)x^{n-k}$ can be rewritten as

$$m(x)x^{n-k} = q'(x)(g(x)p(x)) + r'(x), \tag{7}$$

where $q'(x)$ and $r'(x)$ are the quotient and the remainder of dividing $m(x)x^{n-k}$ by $g(x)p(x)$, respectively. In addition, $r'(x)$ can be rewritten in terms of the quotient and remainder of the division by $g(x)$:

$$r'(x) = q''(x)g(x) + r(x). \tag{8}$$

The $r(x)$ in (8) is the same as that in (3), which is the desired remainder polynomial for the BCH encoding. According to (7) and (8), $r(x)$ can be computed alternatively by:

i) Compute the remainder of dividing $m(x)x^{n-k}$ by $g(x)p(x)$

ii) Divide the remainder from the previous step by $g(x)$.

The remainder from the last step is the desired $r(x)$. Similarly, the division by $g(x)p(x)$ can be implemented by a LFSR architecture, and $p(x)$ can be computed by the clustered look-ahead computation, such that large fanout bottleneck can be eliminated from this LFSR by retiming. Since the degree of $r'(x)$ is at most $n-k+z-1$, the division of $r'(x)$ by $g(x)$ can be implemented by a matrix multiplication without suffering from overwhelming complexity:

$$\underline{r} = H_{(n-k)\times(n-k+z)}\underline{r}', \tag{9}$$

where $\underline{r}$ and $\underline{r}'$ are column vectors formed by the coefficients of $r(x)$ and $r'(x)$, respectively. Such an H matrix with entries $h_{i,j}$ $(0 \leq i < n-k, 0 \leq j < n-k+z)$ can be constructed by Algorithm 2. The polynomial $\hat{r}(x) = \hat{r}_{n-k-1}x^{n-k-1} + \hat{r}_{n-k-2}x^{n-k-2} + \cdots + \hat{r}_0$ is used to keep track of the remainder polynomial in each iteration.

Algorithm 2
initialization:
  set $\hat{r}_0 = 1$, $\hat{r}_i = 0$ for $1 \leq i \leq n-k-1$
begin construction:
  for j in $n-k+z-1$ downto 0 do
    $h_{i,j} = \hat{r}_{n-k-1-i}$ for $0 \leq i \leq n-k-1$
    $\hat{r}(x) \Leftarrow Rem(x \cdot \hat{r}(x))_{g(x)}$
  end

The construction of the H matrix is similar to that of the check matrix for BCH codes. Meanwhile, it can be observed that the right-most $n-k$ columns of the H matrix form an identity matrix:

$$H = [H'_{(n-k)\times z}\ I_{(n-k)\times(n-k)}].$$

Therefore, the complexity of multiplying the H matrix is in the order of $(n-k)\times J$.

3.3 Approach C

As in Approach A, $r(x)$ can be computed by dividing $a(x)$ by $p(x)$. Instead of dividing $p(x)$ directly, $r(x)$ can be computed as follows:

$$r(x) = a(x)/p(x) = a(x)b(x)/(x^v - 1), \tag{10}$$

where $p(x)b(x) = x^v - 1$, and v is a positive integer. Therefore, if we can find a $b(x)$, such that $p(x)b(x)$ is in the form of $x^v - 1$, $r(x)$ can be computed by multiplying $a(x)$ with $b(x)$ and then dividing the product by $x^v - 1$. The multiplication of $b(x)$ can be computed by an architecture similar to that of an FIR filter. Meanwhile, the advantage of

this approach comes from that, compared to a general division, the division by a polynomial in the form of $x^v - 1$ can be computed in a much simpler way. Denote the product of $a(x)$ and $b(x)$ by $f(x)$, $deg(f(x)) = deg(x^v - 1) + deg(r(x)) \leq v + n - k - 1$. Assuming $f(x)$ can be written as $f(x) = f_{v+n-k-1}x^{v+n-k-1} + f_{v+n-k-2}x^{v+n-k-2} + \cdots f_0$, and $\alpha = \lfloor (n-k)/v \rfloor$, $\beta = (n-k)\%v$, by examining the results of long division, the coefficients of $r(x)$ can be computed by the following algorithm:

Algorithm 3

```
for j in 1 to v
   r_{n-k-j} = f_{n-k+v-j}
end
for i in 1 to α − 1
   for j in 1 to v
      r_{n-k-iv-j} = r_{n-k-(i-1)v-j} + f_{n-k-(i-1)v-j}
   end
end
for j in 1 to β
   r_{n-k-αv-j} = r_{n-k-(α-1)v-j} + f_{n-k-(α-1)v-j}
end
```

Meanwhile, by induction, the coefficients for the remainder polynomial $rv(x)$ of dividing $f(x)$ by $(x^v - 1)$ can be derived as:

$$rv_i = \begin{cases} \sum_{j=0}^{\alpha+1} f_{i+jv} & 0 \leq i \leq \beta \\ \sum_{j=0}^{\alpha} f_{i+jv} & \beta < i < v \end{cases} \quad (11)$$

Since $x^v - 1$ can divide $f(x)$, $rv_i = 0$ for $0 \leq i < v$. In addition, from Algorithm 3, it can be derived that $r_i + f_i = rv_i$ for $0 \leq i < v$. Therefore, the computation of the last v coefficients of $r(x)$ can be simplified as $r_i = f_i$ $(0 \leq i < v)$. Using substructure sharing, it can be observed from Algorithm 3 that the computation of each r_i $(v \leq i \leq n - k - 1 - v)$ only needs one additional XOR gate. It follows that the computation of dividing $f(x)$ by $x^v - 1$ can be implemented by $\max(n - k - 2v, 0)$ XOR gates with $\alpha - 1$ XOR gates in the critical path.

If the $p(x)$ computed by clustered look-ahead is used directly in this approach, the smallest v, such that $p(x)$ divides $x^v - 1$ may be large. Although large v leads to smaller gate count in the division by $x^v - 1$, the degree of $b(x)$ becomes larger at the same time. In addition, the gate count in the J-unfolded architecture of computing $a(x)b(x)$ is J times of that in the serial architecture. However, the gate number of the division by $x^v - 1$ does not change in unfolded architectures. Therefore, larger v may lead to higher complexity as J increases. Instead of using $p(x)$, extensions of $p(x)$ can be used in each step of the three-step BCH encoding. We define the extension of $p(x)$ as $p'(x) = p(x)x^d + e(x)$, where d is a positive integer, and $deg(e(x)) < d - (J - 1 - deg(p(x)))$. Since the coefficients of the J highest power terms in $p'(x)$ are the same as that in $p(x)$, using $p'(x)$ can still guarantee that the product $g'(x) = g(x)p'(x)$ satisfy $t'_0 - t'_1 \geq J$. The purpose of using $p'(x)$ instead of $p(x)$ is that the minimum v, such that $p'(x)$ divides $x^v - 1$, can be much smaller than that for $p(x)$ at the cost of slightly increased degree of $p'(x)$. As a result, the overall complexity of BCH encoding can be reduced. For example, the minimum v such that $p(x) = x^5 + x^4 + x^2 + x + 1$ divides $x^v - 1$ is 31, while the minimum v for the extension $p'(x) = x^3p(x) + x + 1$ is 15.

It takes exponential time to find the minimum v, such that some extensions of $p(x)$ can divide $x^v - 1$, by exhaustive search. Alternatively, we can choose a list of numbers, and approximate v by one of the numbers in the list. A good choice is to use a list of $v = 2^i - 1$, where i is a positive integer. The irreducible polynomial factors of $x^{2^i-1} - 1$ can be found easily, and are listed in many documentations, such as [6]. Testing can be carried out by examining if the coefficients of $p(x)$ match the coefficients of the highest power terms in the product of any possible combinations of the irreducible polynomials. Although the estimated v found from this list may be larger than the actual minimum v, this estimation can be completed in realistic time. It turns out that for moderate unfolding factors, and thus moderate degrees of $p(x)$, the v found from the list is not large. Table 1 lists the percentage of $p(x)$ of certain degree, which has some extensions that can divide $x^{31} - 1$, $x^{63} - 1$, $x^{127} - 1$, and $x^{255} - 1$, respectively. Form Table 1, it can be observed that any $p(x)$ of degree 16 has some extensions that can divide $x^{127} - 1$, while 99.96% of all the $p(x)$ of degree 32 have some extensions that can divide $x^{255} - 1$.

Table 1: Percentage of $p(x)$, which has extensions that can divide $x^{31} - 1$, $x^{63} - 1$, $x^{127} - 1$ and $x^{255} - 1$

degree of $p(x)$	8	16	24	32
$x^{31} - 1$	42.19%	0.14%	0.00%	0.00%
$x^{63} - 1$	100%	11.04%	0.04%	0.00%
$x^{127} - 1$	100%	100%	3.01%	0.00%
$x^{255} - 1$	100%	100%	100%	99.96%

Table 2: The complexities of Approach A, B, C for J-unfolded architectures

Approach	Complexity (Number of XOR gates)
A	$[J + (n-k)]J/2 + (n-k)^2/4$
B	$[(n-k) + (n-k)]J/2$
C	$[(J-1+d) + (n-k+d) + (v-(J-1+d))]J/2$ $+ \max(n-k-2v, 0)$

4. COMPLEXITY ANALYSIS AND EXAMPLES

The complexities of the three approaches to reduce the iteration bound are dependent on $g(x)$ and the unfolding factor J. Since the pattern of $g(x)$ varies, it is hard to generalize the precise complexity in terms of the number of XOR gates needed for general BCH codes. Assuming the number of non-zero coefficients in $g(x)$ is half of the total number and the number of non-zero coefficients in $g'(x)$ is $t'_1/2$, the complexities of the three approaches for J-unfolded architectures can be estimated as listed in Table 2. For the purpose of complexity analysis, we assume the non-zero coefficients also account for half of all the coefficients in $p(x)$. Although this assumption is not precise, the degree of $p(x)$ for moderate unfolding factors is much smaller than the degree of $g(x)$ for long BCH codes. Therefore, the resulted estimation error can be ignored.

In Approach A, assuming the degree of $p(x)$ is $J - 1$, the multiplication of $p(x)$ needs $J/2 \cdot J$ XOR gates in the J-unfolded architecture. Furthermore, the LFSR implementing the division by $g'(x) = g(x)p(x)$ needs $t'_1/2 \cdot J \approx (n - k)J/2$ XOR gates. The remaining complexity in Approach A is attributed to the multiplication of the lower triangular matrix Q'. Assuming the non-zero entries account for half of all the entries in the lower triangular matrix, the complexity of multiplying matrix Q' is $(n-k)^2/4$.

In Approach B, the J-unfolded LFSR to implement the division by $g(x)p(x)$ also needs approximately $(n-k)J/2$ XOR gates. Meanwhile, as can be observed from Algorithm 2, the H' matrix is formed by shifting modulo $g(x)$. Since the non-zero coefficients in $g(x)$ account for half of all the coefficients and are well-distributed, it can be assumed that the number of non-zero entries in H' is also half. Therefore, the complexity of multiplying H is $J/2 \cdot (n-k)$ XOR gates.

$p'(x) = x^d p(x) + e(x)$ is used instead of the $p(x)$ computed from clustered look-ahead in Approach C. Since $deg(p(x)) = J-1$, the degree of $p'(x)$ is $J-1+d$. Assuming the non-zero coefficients account for half in $p'(x)$, the multiplication of $m(x)$ by $p'(x)$ needs $(J-1+d)/2 \cdot J$ XOR gates to implement in a J-unfolded architecture. The degree of $g'(x) = p'(x)g(x)$ is $n-k+J-1+d$, and the coefficients of the $J-1$ terms after the highest power term are zero. Assuming the non-zero coefficients also account for half of the remaining $n-k+d$ coefficients, the J-unfolded LFSR to implement the division by $g'(x)$ has $(n-k+d)/2 \cdot J$ XOR gates. Since $deg(b(x)) = deg(x^v-1) - deg(p'(x)) = v-(J-1+d)$, the complexity of the J-unfolded multiplication by $b(x)$ is $(v-(J-1+d))/2 \cdot J$. In addition, as discussed in Approach C, the division by x^v-1 needs $\max(n-k-2v, 0)$ XOR gates to implement.

It can be shown that given $n-k \geq 2J$, the complexity of Approach A is always larger than that of Approach B. The assumption $n-k \geq 2J$ is typically satisfied for long BCH codes with moderate unfolding factors. Meanwhile, it can be shown that the complexity of Approach B is larger than that of Approach C, given $v+d < n-k$. As can be observed from Table 1, $v = 255$ can be used for 99.96% of all the $p(x)$ for unfolding factor 32, and smaller v are sufficient for smaller unfolding factors. However, for example, for BCH codes with block length longer than $8K$, $n-k$ is larger than 500 for code rate up to 93.81%. Therefore, $v+d < n-k$ is usually satisfied for long BCH code with moderate unfolding factors.

As an example, the three proposed architectures are applied to an (8191, 7684) BCH code using generator polynomial $g(x) = x^{507} + x^{506} + x^{502} + \cdots + x^6 + x^2 + 1$. Some of the results for different unfolding factors are summarized in Table 3. Since the polynomial multiplications and the matrix multiplications can be pipelined, the associated latency can be excluded from the total number of clock cycles needed to encode each data block. Therefore, the number of clock cycles to process one data block is determined by the latency of the LFSR architecture, which can be computed as $\lceil$the length of the input to the LFSR$/J\rceil$. For example, in Approach A, the length of the input to the LFSR, which is $m(x)p(x)$, is 7684+15=7699 for $J = 16$. Hence, $\lceil 7699/16 \rceil = 482$ clock cycles are needed to encode one block of data in this case. The critical loop of the architecture in [4] lies in the LFSR implementing the division by $p(x)$, and the iteration bound of this architecture is JT_{XOR} for the J-unfolded (8191, 7684) BCH encoder. Approach A, B and C can reduce the iteration bound by replacing the LFSR implementing the division by $p(x)$ with architectures free of feedback loops. As can be observed from Table 3, the iteration bounds of these three approaches are much smaller than that of [4]. In addition, since the same $p(x)$ can be used in Approach A and B, the iteration bounds of these two approaches are the same. However, $p'(x)$ instead of $p(x)$ is used in Approach C. As a result, the iteration bound of Approach C is different. The speedup and gate count for each proposed architecture are also listed in Table 3. As can be observed, Approach C has the smallest area requirement and can achieve similar or higher speedup than Approach A and B. Therefore, Approach C is the optimum approach for unfolding factors ranging from 8 to 32.

Table 3: The speedup and gate counts of Approach A, B, C for J-unfolded architecture

	J=8	J=16	J=24	J=32
$deg(p(x))$	7	15	21	30
$deg(p'(x))$(used v) in Appr. C	17(63)	49(127)	71(255)	93(255)
# of clock cycles/data block in [4] and Appr. A	962	482	322	242
# of clock cycles/data block in Appr. B	961	481	321	241
# of clock cycles/data block in Appr. C	963	484	324	244
T_∞ of [4]($\#T_{XOR}$)	8	16	24	32
T_∞ of Appr. A, B ($\#T_{XOR}$)	5.103	8.000	12.793	15.956
T_∞ of Appr. C ($\#T_{XOR}$)	4.166	7.768	11.111	14.034
Speedup of Appr. A over [4]	33.33%	100.00%	84.62%	100%
Speedup of Appr. B over [4]	33.47%	100.42%	85.19%	100.83%
Speedup of Appr. C over [4]	59.83%	99.17%	98.77%	111.58%
# of XOR in Appr. A	57622	67338	69230	71549
# of XOR in Appr. B	4048	8172	11981	16344
# of XOR in Appr. C	2845	5469	9432	12512

5. CONCLUSION

Novel architectures have been proposed in this paper to reduce the achievable minimum clock period of parallel long BCH encoders derived through unfolding after the fanout bottleneck has been eliminated. In addition, the proposed schemes can also be used to speed up long Cyclic Redundancy Checking (CRC) and systematic long Reed-Solomon encoders. Among the three approaches, Approach C is optimum for moderate unfolding factors. However, the complexity for finding the smallest v, such that some extensions of $p(x)$ divide x^v-1, is extremely high if exhaustive searching is used. Future work will be directed to develop systematic algorithm to compute the smallest v and the corresponding extensions of $p(x)$. In addition, reducing the iteration bound of the LFSR implementing the division by $g'(x)$ is of great interest to achieve further speedup for parallel long BCH encoders.

6. ACKNOWLEDGMENTS

The authors would like to thank Niegel Boston and Rene Glaise for the helpful discussions.

7. REFERENCES

[1] T. -B. Pei and C. Zukowski, "High-Speed Parallel CRC Circuits in VLSI," *IEEE Trans. Commun.*, Vol. 40, Issue.4, pp. 653-657, Apr.1992.

[2] J. H. Derby, "High-Speed CRC Computation Using State-Space Transformation," *Global Telecommunications Conference, 2001*, GLOBECOM '01.IEEE, Vol.1, pp. 166-170.

[3] 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.

[4] K. K. Parhi, "Eliminating the Fanout Bottleneck in Parallel Long BCH Encoders," *IEEE. Trans. on Circuits and Systems-Part I: Regular Papers*, vol.51, No.3, Mar.2004.

[5] K. K. Parhi, "VLSI Digital Signal Processing Systems-Design and Implementation," John Wiley & Sons, 1999.

[6] S. B. Wicker, "Error Control Systems for Digital Communication and Storage," Prentice Hall, Upper Saddle River, New Jersey, 1995.