V. Conclusion

We consider the voltage setup problem for application specific multiple voltage DVS system design. The problem seeks to determine the number of voltage levels and the voltage at each level to minimize the average energy consumption for a given set of applications. We give optimal solutions in analytic form for the dual-voltage system and develop two heuristics (an iterative approach and an approximation method) for the general case. The hardware overhead to supply multiple voltages, once obtained, can be conveniently integrated into our techniques to solve the voltage setup problem. We apply our methods to the designs of an ad hoc application specific system and the MPEG video encoder. Simulation results show the correctness and efficiency of our approaches. We also observe that multiple voltage system, if the voltage levels are set properly, can indeed achieve energy reduction very close to the full potential by DVS.

Acknowledgment

The authors thank the editor-in-chief, the associated editor, and the reviewers for their valuable comments. A full version of this paper can be found in [14].

References

[1] T. D. Burd, T. Pering, A. Stratakos, and R. Brodersen, "A dynamic voltage scaled microprocessor system," *IEEE J. Solid-State Circuits*, vol. 35, no. 11, pp. 1571–1580, Nov. 2000.

[2] G. Qu, "What is the limit of energy saving by dynamic voltage scaling?," in *IEEE/ACM Int. Conf. Computer-Aided Design*, 2001, pp. 560–563.

[3] M. Fleischmann, "LongRun power management—dynamic power management for Crusoe processors," Whitepaper, Transmeta Corp., 2001.

[4] Advanced Micro Devices, "AMD Athlon 4 processors," data sheet reference no. 24319, 2001.

[5] Intel, "The Intel Xscale Microarchitecture," Technical Summary, 2000.

[6] T. Ishihara and H. Yasuura, "Voltage scheduling problem for dynamically variable voltage processors," in *Int. Symp. Low Power Electronics and Design*, 1998, pp. 197–202.

[7] G. Qu and M. Potkonjak, "Techniques for energy-efficient communication pipeline design," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 10, no. 5, pp. 542–549, Oct. 2002.

[8] G. Quan and X. Hu, "Energy efficient fixed-priority scheduling for real-time systems on variable voltage processors," in *Proc. Design Automation Conf.*, 2001, pp. 828–833.

[9] C. Chen and M. Sarrafzadeh, "Provably good algorithm for low power consumption with dual supply voltages," in *Proc. IEEE/ACM Int. Conf. Computer Aided Design*, 1999, pp. 76–79.

[10] S. Dhar and D. Maksimovic, "Low-power digital filtering using multiple voltage distribution and adaptive voltage scaling," in *Proc. Int. Symp. Low Power Electronics and Design*, 2000, pp. 207–209.

[11] J.-M. Chang and M. Pedram, "Energy minimization using multiple supply voltages," in *Proc. Int. Symp. Low Power Electronics and Design*, 1996, pp. 157–162.

[12] S. Raje and M. Sarrafzadeh, "Variable voltage scheduling," in *Proc. Int. Symp. Low Power Electronics and Design*, 1995, pp. 9–14.

[13] A. Kalavade and P. Moghe, "A tool for performance estimation of networked embedded end-systems," in *Proc. Design Automation Conf.*, 1998, pp. 257–262.

[14] S. Hua and G. Qu, "Voltage set-up problem for embedded systems with multiple voltages," Institute for Advanced Computer Studies (UMIACS), Univ. Maryland, Tech. Rep., Feb. 2005.

High-Speed Architectures for Parallel Long BCH Encoders

Xinmiao Zhang and Keshab K. Parhi

***Abstract*—Long Bose–Chaudhuri–Hocquenghen (BCH) codes are used as the outer error correcting codes in the second-generation Digital Video Broadcasting Standard from the European Telecommunications Standard Institute. These codes can achieve around 0.6-dB additional coding gain over Reed–Solomon codes with similar code rate and codeword length 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 implementation of the encoders. In addition, long BCH encoders suffer from the effect of large fanout. In this paper, three novel architectures are proposed to reduce the achievable minimum clock period for 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%.**

***Index Terms*—Bose–Chaudhuri–Hocquenghen (BCH), critical loop, encoder, fanout, generator polynomial, iteration bound, linear feedback shift register (LFSR), parallel processing, retiming, unfolding.**

I. Introduction

Bose–Chaudhuri–Hocquenghen (BCH) codes are among the most extensively used error correcting codes in modern communication systems. Long BCH codes of block length 32 400-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 0.6-dB coding gain over AWGN channel with similar rate and codeword length [1]. High-rate Reed–Solomon codes of length 255 or longer have broad applications in optical communication systems. Therefore, long BCH codes are of great interest.

BCH encoders are conventionally implemented by a linear feedback shift register (LFSR) architecture. While such an architecture can operate 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 10–40 Gb/s is usually desired, the clock frequency of such LFSR-based encoders can not keep up with the data transmission rate, thus parallel processing must be employed. Moreover, long BCH encoders face the problem of large fanout effect. The delay of a gate grows linearly with the fanout. Due to the large number of nonzero coefficients in the generator polynomials of long BCH codes, 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 [2]–[4]. 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 achieved through unfolding can be eliminated by the approach in [5]. Nevertheless, the speedup of parallel processing is offset by the increased achievable minimum clock period in this approach. In this paper, three novel architectures are proposed to reduce the achievable minimum clock period in unfolded long BCH encoders after the large fanout effect has been eliminated. As an

Manuscript received February 12, 2004; revised November 17, 2004. This work was supported by the Army Research Office under Grant DA/DAAD19-01-1-0705.

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

Digital Object Identifier 10.1109/TVLSI.2005.850125

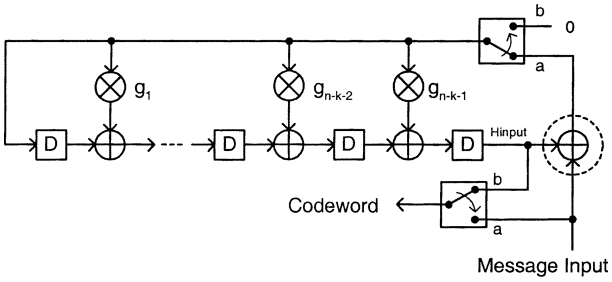

Fig. 1. Serial BCH encoder architecture.

example, the proposed architectures can achieve more than 100% speedup over the architecture in [5] for a 32-parallel (8191, 7684) BCH code. Without loss of generality, only binary BCH codes are considered. For high-speed implementations of BCH decoders, the interested reader is referred to [6]. Parts of this paper are based on [7].

The structure of this paper is as follows. Section II contains a brief description of the LFSR-based BCH encoder architecture and a prior approach to eliminate the large fanout effect. In Section III, novel parallel architectures, which can reduce the minimum achievable clock period of long BCH encoders free of large fanout bottleneck, are explained in detail. Section IV addresses the complexity analysis of the proposed architectures and provides some comparison results. Conclusions are provided in Section V.

II. 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 $(m_{k-1}, m_{k-2}, \ldots, m_0)$ $(m_i \in GF(2), 0 \le i < k)$ as the coefficients of a degree $k-1$ message polynomial $m(x) = m_{k-1}x^{k-1} + m_{k-2}x^{k-2} + \ldots + m_0$, and the corresponding n-bit codeword $(c_{n-1}, c_{n-2}, \ldots, c_0)(c_i \in GF(2), 0 \le i < n)$ as the coefficients of a degree $n-1$ codeword polynomial $c(x) = c_{n-1}x^{n-1} + c_{n-2}x^{n-2} + \ldots + c_0$, the systematic BCH encoding can be implemented according to

$$c(x) = m(x) \cdot x^{n-k} + \mathrm{Rem}\left(m(x) \cdot x^{n-k}\right)_{g(x)} \tag{1}$$

where $g(x)$ is the degree $n-k$ generator polynomial, and $\mathrm{Rem}(a(x))_{b(x)}$ denotes the remainder polynomial of dividing $a(x)$ by $b(x)$.

Assuming the generator polynomial can be expressed as $g(x) = g_{n-k}x^{n-k} + g_{n-k-1}x^{n-k-1} + \cdots + g_0$ $(g_i \in GF(2), 0 \le i \le n-k)$, a systematic binary (n, k) BCH encoder can be implemented by the LFSR architecture illustrated in Fig. 1. The function of this architecture is to implement the polynomial division by $g(x)$. For binary BCH codes, each multiplier in Fig. 1 can be replaced by connection or no connection when g_i is "1" or "0," respectively. Since $g_{n-k} = g_0 =$ '1' for any generator polynomial of an (n, k) BCH code, the multipliers for g_{n-k} and g_0 are replaced by connections. The entire 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 kth clock cycle, the $n-k$ delay elements contain $\mathrm{Rem}(m(x) \cdot x^{n-k})_{g(x)}$. 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. Although this LFSR-based architecture for BCH encoding is fairly simple and can run at high speed, when the clock frequency can not meet the requirements of high-speed applications, parallel processing needs to be explored. Furthermore, long BCH encoders face another problem. Due to the large number of nonzero coefficients in the generator polynomial, the right-most XOR gate in Fig. 1 has large fanout, which can slow down the encoder significantly. The fanout bottleneck also exists in parallel long BCH encoders.

The effect of large fanout in serial BCH encoders can be always eliminated by retiming. 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 be always 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.

The unfolding technique [8] can be employed to derive parallel BCH encoders. 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 gate is expressed as a node, and is connected to other nodes by paths with delay elements. The J-unfolded architecture makes J copies of each node in the original architecture. Assume node U is connected to node V by a path with w delay elements in the original architecture. Then in the J-unfolded architecture, U_i is connected to $V_{(i+w)\%J}$ by a path with $\lfloor (i+w)/J \rfloor$ delay elements, where $U_i, V_j (0 \le i, j < J)$ are copies of nodes U and V, respectively, and $(i+w)\%J$ is the remainder of dividing $i+w$ by J [8]. It can be derived that if the number of delay elements, w, in a path is less than the unfolding factor J, then 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. Assume the generator polynomial $g(x)$ of a BCH code can be rewritten as

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

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 nonzero 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 copy 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.

In the case of $t_0 - t_1 < J$, the scheme proposed in [5] can be employed to modify the generator polynomial such that the retiming around the right-most XOR gates in the J-unfolded LFSR architecture is enabled. $m(x)x^{n-k} = q(x)g(x) + r(x)$, where $q(x)$ and $r(x)$ are the quotient and remainder polynomial, respectively. Multiplying a polynomial $p(x)$ to both sides, we can get

$$m(x)p(x)x^{n-k} = q(x)\left(g(x)p(x)\right) + r(x)p(x). \tag{3}$$

According to (3), the proposed scheme in [5] computes $r(x) = \mathrm{Rem}(m(x)x^{n-k})_{g(x)}$ 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 $g(x)p(x)$; and 3) compute the quotient of dividing the remainder from step 2 by $p(x)$.

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

$$g'(x) = x^{t'_0} + x^{t'_1} + \cdots + x^{t'_{s'-2}} + 1 \tag{4}$$

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 nonzero coefficients in $g'(x)$. In the case of $t_0 - t_1 < J$, a $p(x)$ can be found to ensure $t'_0 - t'_1 \ge J$. Accordingly,

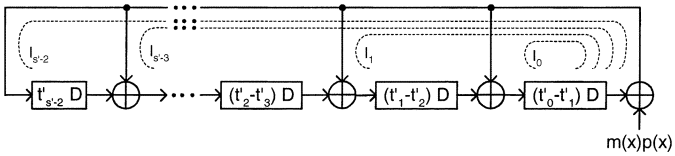

Fig. 2. LFSR architecture for the second step.

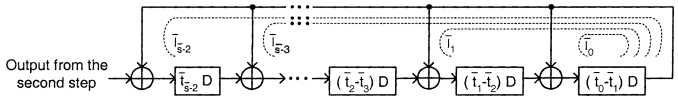

Fig. 3. LFSR architecture for the third step.

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 derived according to clustered look-ahead computation [8].

A second LFSR architecture is needed to compute the division by $p(x)$ in the third step. It can be observed from clustered look-ahead computation that the degree of $p(x)$ 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. Further speedup of parallel long BCH encoders can be achieved by the novel architectures proposed in the next section.

III. High-Speed Parallel Long BCH Encoders With Eliminated Fanout Bottleneck

Pipelining can be employed to achieve higher speed in an architecture without feedback loops. 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. T_∞ is defined as the maximum of all the loop bounds, and the loop with the largest loop bound is called the critical loop. 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 the J-unfolded architecture is J times the iteration bound of the serial architecture. In the case that the critical path is longer than $\lceil T_\infty \rceil$, retiming can be applied to achieve clock period minimization. Moreover, retiming does not change iteration bound.

In the three-step process of BCH encoding, 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 architecture, 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. Assume the iteration bounds of the LFSRs in the second and the third steps are T_∞^A and T_∞^B, respectively. Then the minimum achievable clock period of the entire J-unfolded BCH encoder is $\max(\lceil JT_\infty^A \rceil, \lceil JT_\infty^B \rceil)$.

Assuming $g'(x)$ is expressed as in (4), the LFSR architecture illustrated in Fig. 2 can be used to implement the second step. The total number of loops in this architecture is $s'-1$. Since this LFSR architecture can be retimed to eliminate the large fanout effect, and retiming does not change the iteration bound, it can be assumed that the delay for each gate is the same in the iteration bound computation. Assuming the delay for each XOR gate is $T_{\rm XOR}$, it can be computed that the loop bound for loop l_i, $(0 \le i < s'-1)$ in Fig. 2 is $(2+i)T_{\rm XOR}/(t'_0 - t'_{i+1})$. For long BCH codes, the number of nonzero coefficients is typically half of the total number of coefficients in the generator polynomial $g(x)$. In addition, it can be observed that in $g'(x) = p(x)g(x)$, approximately half of the coefficients for the t'_1+1 lowest-power terms are nonzero. Furthermore, these nonzero 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. Each nonzero coefficient corresponds to one XOR gate in Fig. 2. It follows that the number of XOR gates in loop l_i can be approximated by $2 + (t'_1 - t'_{i+1})/2$. Therefore, the loop bound for l_i is

$$\frac{(2+i)}{t'_0 - t'_{i+1}} T_{\rm XOR} \approx \frac{4 + (t'_1 - t'_{i+1})}{2\left(t'_0 - t'_{i+1}\right)} T_{\rm XOR}.$$

Since $t'_0 - t'_1 \ge J$, it can be derived that

$$\frac{4 + (t'_1 - t'_{i+1})}{2\left(t'_0 - t'_{i+1}\right)} T_{\rm XOR} \le \frac{(t'_0 - t'_{i+1}) - (J-4)}{2\left(t'_0 - t'_{i+1}\right)} T_{\rm XOR} = \left(\frac{1}{2} - \frac{J-4}{2\left(t'_0 - t'_{i+1}\right)}\right) T_{\rm XOR}.$$

For practical unfolding factor $J \ge 4$, it follows that

$$\left(\frac{1}{2} - \frac{J-4}{2\left(t'_0 - t'_{i+1}\right)}\right) T_{\rm XOR} \le \left(\frac{1}{2} - \frac{J-4}{2t'_0}\right) T_{\rm XOR}. \quad (5)$$

Since the iteration bound is the maximum of all the loop bounds, it can be observed from (5) that the iteration bound is at most around $T_{XOR}/2$. In addition, the larger the unfolding factor J, the smaller the iteration bound of the LFSR implementing the division by $g'(x)$.

Assume $p(x)$ 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 nonzero coefficients in $p(x)$. Then the third step of BCH encoding can be implemented by the LFSR architecture shown in Fig. 3. It can be computed that the loop bound of loop $\bar{l}_i (0 \le i < \bar{s}-1)$ is $(1+i)T_{\rm XOR}/(\bar{t}_0 - \bar{t}_{i+1})$. Due to the short length, $p(x)$ does not have the property that around half of the coefficients are nonzero. Since $\deg(p(x)) \le J-1$, $\bar{t}_0 - \bar{t}_{i+1}$ is always less than J. However, $t'_0 - t'_{i+1}$ is always larger than J and increases with i. Hence, the loop bounds for the LFSR in the third step is typically larger than that for the LFSR in the second step. The iteration bound for the encoder is the maximum of the iteration bounds of the two LFSRs. Therefore, reducing the iteration bound of the LFSR for the third step is critical to speed up the entire encoder. Particularly, in the case of $t_0 - t_1 = 1$, it can be observed from clustered look ahead computation that $\bar{t}_0 - \bar{t}_1 = 1$. Hence, the loop bound for $\bar{l}_0$ in Fig. 3 is $T_{\rm XOR}$. 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 $T_{\rm XOR}$ 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. 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 next.

A. Approach A

Denote the output of the second step, $\mathrm{Rem}(m(x)p(x))_{g'(x)}$, 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 $p(x)$ is $z(z \leq J-1)$, $\deg(a(x)) \leq \deg(g(x)p(x)) - 1 = n-k+z-1$. 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 $a(x)$. The Q' matrix can be constructed by finding the quotients of dividing $x^z, x^{z+1}, \cdots, x^{z+n-k-1}$ by $p(x)$. For the detailed algorithm, the interested reader is referred to [7, Algorithm 1]. It can be observed that Q' is a lower triangular matrix of dimension $(n-k) \times (n-k)$. 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.

B. Approach B

The remainder polynomial $r(x)$ can be computed alternatively as proposed in [4]. $m(x)x^{n-k}$ 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 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 $g(x)$

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

It can be derived that the $r(x)$ in (8) is the desired remainder polynomial for BCH encoding. The scheme proposed in [4] tries to speed up J-parallel encoders by finding a $p(x)$, such that in the product $p(x)g(x)$, there are at least J consecutive zero coefficients between each nonzero coefficient. However, such a $p(x)$ is extremely hard to find and may not exist for every BCH code, especially for long BCH codes. Instead, we propose to use the $p(x)$ computed from clustered look-ahead, which is guaranteed to exist for every BCH code. According to (7) and (8), $r(x)$ can be computed alternatively by: 1) compute the remainder of dividing $m(x)x^{n-k}$ by $g(x)p(x)$ and 2) compute the remainder of dividing the remainder from the previous step by $g(x)$. Similarly, the division by $g(x)p(x)$ can be implemented by a LFSR architecture as that in Fig. 2, and $p(x)$ can be computed from clustered look-ahead, such that large fanout bottleneck can be eliminated from this LFSR by retiming. If the division by $g(x)$ in the last step of this approach is implemented by a LFSR, the J-unfolded architecture will face the problem of large fanout effect when $t_0 - t_1 < J$. Nevertheless, since the degree of $r'(x)$ is less than $n-k+J-1$, the division of $r'(x)$ by $g(x)$ can be implemented by a matrix multiplication without suffering from overwhelming complexity. Assuming $\deg(p(x)) = z$, $\deg(r'(x)) \leq \deg(g(x)p(x)) - 1 = n-k+z-1$. In addition, $\deg(r(x)) \leq n-k-1$. Therefore, the coefficients of $r(x)$ can be computed by a matrix multiplication of dimension $(n-k)\times(n-k+z)$

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

where $\underline{r}$ and $\underline{r}'$ are vectors formed by the coefficients of $r(x)$ and $r'(x)$, respectively. The H matrix can be constructed by finding the remainders of dividing $1, x, x^2, \cdots x^{n-k+z-1}$ by $p(x)$. For the detailed algorithm, the interested reader is referred to [7, Algorithm 2]. It can be observed that the right-most $n-k$ columns of the H matrix form an identity matrix. Hence, the complexity of multiplying the H matrix is in the order of $(n-k) \times J$. For long BCH codes with moderate unfolding factors, the complexity of Approach B is lower than that of Approach A. Nevertheless, the complexity can be reduced further by the following approach.

C. Approach C

As in Approach A, $r(x) = a(x)/p(x)$. Alternatively, $r(x)$ can be computed as follows:

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

where $p(x)b(x) = x^v - 1 (v \in \mathbb{Z}^+)$. 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 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 $f(x) = f_0 + f_1 + \cdots + f_{v+n-k-1}x^{v+n-k-1}$, 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 A

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

On the other hand, the coefficients for the remainder polynomial, $rv(x)$, of dividing $f(x)$ by (x^v-1) can be derived as

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

Since x^v-1 can divide $f(x)$, $rv_i = 0$ for $0 \leq i < v$. In addition, from Algorithm A, 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, 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 minimum 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 count of the division by x^v-1 does not change in the unfolded architecture. 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 $p'(x) = p(x)x^d + e(x)$,

TABLE I
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	10	16	18	20	24	32
$x^{31}-1$	42.19%	12.89%	0.14%	0.00%	0.00%	0.00%	0.00%
$x^{63}-1$	100%	100%	11.04%	2.87%	0.68%	0.04%	0.00%
$x^{127}-1$	100%	100%	100%	49.07%	31.71%	3.01%	0.00%
$x^{255}-1$	100%	100%	100%	100%	100%	100%	99.96%

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 $g'(x) = g(x)p'(x)$ satisfies $t'_0 - t'_1 \geq J$. 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.

The minimum v, such that some extensions of $p(x)$ divide $x^v - 1$, can be found by increasing v by one each time and test if $x^v - 1$ can be divided by any extensions of $p(x)$. However, this exhaustive search takes exponential time. 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 the list of $v = 2^i - 1 (i \in \mathbb{Z}^+)$. The irreducible polynomial factors of $x^{2^i-1} - 1$ can be found easily, and are listed in many documentations, such as [9]. 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 I lists the percentage of $p(x)$, which has some extensions that can divide $x^{31}-1$, $x^{63}-1$, $x^{127}-1$, and $x^{255}-1$, respectively. Form Table I, it can be observed that any $p(x)$ of degree 16 has some extensions that divide $x^{127}-1$, while 99.96% of all the $p(x)$ of degree 32 have some extensions that divide $x^{255}-1$.

IV. COMPLEXITY ANALYSES AND EXAMPLES

The complexities of the three approaches to reduce the iteration bound are dependent on the generator polynomial $g(x)$ and the unfolding factor J. Since the pattern of $g(x)$ various, 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 nonzero coefficients in $g(x)$ is half of the total number and the number of nonzero coefficients in $g'(x)$ is $t'_1/2$, the complexities of the three approaches for J-unfolded architectures can be estimated. In addition, for the purpose of complexity analysis, we assume the nonzero 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 that of $g(x)$ for long BCH codes. Therefore, the resulting estimation errors can be ignored.

In Approach A, assume the degree of $p(x)$ is $J-1$. Then the multiplication by $p(x)$ 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 nonzero entries account for half of all the entries in the lower triangular, the complexity of multiplying matrix Q' is $(n-k)^2/4$.

Compared to Approach A, Approach B does not need to multiply the message polynomial $m(x)$ by any polynomials before it is fed into the LFSR architecture. Similarly, the J-unfolded LFSR to implement the division by $g(x)p(x)$ in (7) needs approximately $(n-k)J/2$ XOR gates. In addition, the H matrix is formed by two parts: an $(n-k) \times (n-k)$ identity matrix and an $(n-k) \times J$ is the dimension of H' matrix formed by shifting modulo $g(x)$. Since the nonzero coefficients in $g(x)$ account for half of all the coefficients and are well-distributed, it can be assumed that the number of nonzero entries in H' is also half. Therefore, the complexity of multiplying H is $J/2 \cdot (n-k)$.

TABLE II
COMPLEXITIES OF APPROACH A, B, AND 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	$[(J-1+d)+(n-k+d)+(v-(J-1+d))]J/2$ $+\max(n-k-2v,0)$

$p'(x) = x^d p(x) + e(x)$ 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 nonzero coefficients account for half in $p'(x)$, the J-unfolded architecture to implement the multiplication of $m(x)$ by $p'(x)$ needs $(J-1+d)/2 \cdot J$ XOR gates. 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 nonzero 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.

In summary, the complexities of the three approaches for the J-unfolded architectures are listed in Table II. In the following, we will show 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. To show

$$\frac{[J+(n-k)]\,J}{2} + \frac{(n-k)^2}{4} > \frac{[(n-k)+(n-k)]\,J}{2}$$

it it sufficient to prove

$$2J^2 + (n-k)^2 > 2(n-k)J$$

which is equivalent to showing

$$2J^2 + (n-k)(n-k-2J) > 0.$$

Since we know $n-k > 0$, and $n-k-2J \geq 0$, it is clear that Approach B has lower complexity than Approach A. Similarly, 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 I, $v = 255$ 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 rates up to 93.81%. Therefore, $v+d < n-k$ is usually satisfied for long BCH codes with moderate unfolding factors.

TABLE III
SPEEDUP AND GATE COUNTS OF APPROACH A, B, AND 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 [5] 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 [5]($\#T_{XOR}$)	8	16	24	32
T_∞ of Appr. A, B ($\#T_{XOR}$)	5.103	8.000	12.793	15.956
T_∞ of Appr. C ($\#T_{XOR}$)	4.167	7.769	11.111	14.034
Speedup of Appr. A over [5]	33.33%	100.00%	84.62%	100%
Speedup of Appr. B over [5]	33.47%	100.42%	85.19%	100.83%
Speedup of Appr. C over [5]	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

As an example, all of the three proposed schemes 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 results for different unfolding factors are summarized in Table III. 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, the encoding of one data block requires $\lceil 7699/16 \rceil = 482$ clock cycles in this case. The critical loop of the architecture in [5] lies in the LFSR implementing the division by $p(x)$, and the iteration bound of this architecture is $JT_{\rm XOR}$ 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 III, the iteration bounds of these three approaches are much smaller than that of [5]. 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 III. 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.

V. CONCLUSION

Three 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 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 $p(x)$ divides $x^v - 1$, is extremely high if exhaustive searching is used. Future work will be directed to developing systematic algorithms 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.

REFERENCES

[1] Y. Chen and K. K. Parhi, "Area efficient parallel decoder architecture for long BCH codes," in *Proc. IEEE Int. Conf. Acoustics, Speech, and Signal Processing*, vol. 5, Montreal, QC, Canada, May 2004, pp. V-73–V-76.

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

[3] J. H. Derby, "High-speed CRC computation using state-space transformation," in *Proc. Global Telecommunications Conf.*, vol. 1, 2001, pp. 166–170.

[4] 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.

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

[6] L. Song, M. Yu, and M. S. Shaffer, "10- and 40-Gb/s forward error correction devices for optical communications," *IEEE J. Solid-State Circuits*, vol. 37, no. 11, pp. 1565–1573, Nov. 2002.

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

[8] K. K. Parhi, *VLSI Digital Signal Processing Systems-Design and Implementation*. New York: Wiley, 1999.

[9] S. B. Wicker, *Error Control Systems for Digital Communication and Storage*. Englewood Cliffs, NJ: Prentice-Hall, 1995.

A Case for Asymmetric-Cell Cache Memories

Andreas Moshovos, Babak Falsafi, Farid N. Najm, and Navid Azizi

***Abstract*—In this paper, we make the case for building high-performance asymmetric-cell caches (ACCs) that employ recently-proposed asymmetric SRAMs to reduce leakage proportionally to the number of resident zero bits. Because ACCs target memory value content (independent of cell activity and access patterns), they complement prior proposals for reducing cache leakage that target memory access characteristics. Through detailed simulation and leakage estimation using a commercial 0.13-μm CMOS process model, we show that: 1) on average 75% of resident data cache bits and 64% of resident instruction cache bits are zero; 2) while prior research carefully evaluated the fraction of accessed zero bytes, we show that a high fraction of accessed zero bytes is neither a necessary nor a sufficient condition for a high fraction of resident zero bits; 3) the zero-bit program behavior persists even when we restrict our attention to live data, thereby complementing prior leakage-saving techniques that target inactive cells; and 4) ACCs can reduce leakage on the average by 4.3× compared to a conventional data cache without any performance loss, and by 9× at the cost of a 5% increase in overall cache access latency.**

***Index Terms*—Cache memories, computer architecture, high performance, leakage power reduction.**

I. INTRODUCTION

In this work, we study methods that exploit the memory bit-value behavior of programs to drastically reduce leakage or static power dis-

Manuscript received June 14, 2004; revised February 7, 2005. This work was supported in part by the Semiconductor Research Corporation under Grant SRC 2001-HJ-901 and in part by an NSERC Discovery Grant. The work of N. Azizi was supported by an NSERC postgraduate scholarship (PGS A).

A. Moshovos, F. N. Najm, and N. Azizi are with the Electrical and Computer Engineering Department, University of Toronto, Toronto, ON M5S 3G4, Canada (e-mail: moshovos@eecg.toronto.edu; najm@toronto.edu; nazizi@eecg.toronto.edu).

B. Falsafi is with the Electrical and Computer Engineering Department of the Carnegie-Mellon University, Pittsburgh, PA 15213 USA (e-mail: babak@cmu.edu).

Digital Object Identifier 10.1109/TVLSI.2005.850127