Low-Complexity Resource-Shareable Parallel Generalized Integrated Interleaved Encoder

Yok Jye Tang and Xinmiao Zhang, *Senior Member, IEEE*

***Abstract*—Generalized integrated interleaved (GII) codes nest a set of linear block codewords to generate codewords belonging to stronger codes. They are among the best error-correcting codes for next-generation hyper-speed digital communications and storage. Serial encoders for GII codes based on BCH codes have been previous investigated. They consist of BCH encoders whose inputs and outputs are multiplied by vectors decided by the nesting scheme. However, parallel GII encoders for high-speed systems can not be designed by directly extending serial encoders due to the unique feature that BCH codes of different error-correcting capabilities are involved. Moreover, GII decoder complexity and latency can be greatly reduced by sharing the encoder to compute short remainders for syndrome computation. Although previous resource-shareable BCH encoders can be utilized to implement resource-shareable GII encoders, they are all serial. This paper first proposes a low-complexity scheme to handle the different error-correcting capabilities of the involved codes and align the input and parity symbols for parallel processing. Then two efficient parallel resource-shareable BCH encoder architectures to be used as GII encoder components are developed. The first design is achieved by deriving parallel register state update formulas for concatenated linear-feedback shift registers (LFSRs). Through reformulating the remainder polynomial divisions, the second design allows the inputs to be added to different LFSR taps, and accordingly reduces the complexity by a significant portion. For an example 160-parallel GII-BCH encoder considered for Flash memory applications, the second proposed design requires 14% smaller area compared to the first one. Besides both of them lead to around 50% latency reduction in the nested syndrome computation with small area overheads compared to the best possible alternative design.**

***Index Terms*—BCH codes, error-correcting codes, generalized integrated interleaved codes, parallel resource-shareable encoder.**

I. Introduction

For next-generation digital storage and communications, such as Flash memories and optical communications, error-correcting codes achieving hyper throughput with high coding gain are needed. Generalized integrated interleaved (GII) codes [1]–[4] are excellent candidates for such systems. A GII $[m, v]$ code nests $m > 1$ Reed-Solomon (RS) or BCH sub-codewords to form codewords of v $(v < m)$ stronger RS or BCH codes. Decoding is carried out individually on each sub-codeword most of the time to achieve hyper throughput. Also more errors can be corrected by utilizing the nested codewords.

Much research has been carried out on GII decoder design recently [5]–[8]. However, the design of GII encoder has only been investigated in [9], [10] and both of them are serial architectures. For a GII $[m, v]$ code, the encoder consists of m individual RS/BCH encoders of different error-correcting capabilities. The inputs and outputs of the RS/BCH encoders are multiplied by vectors according to the nesting scheme. The design of GII-BCH encoders faces more challenges compared to that of GII-RS encoders since the nesting is defined by polynomials instead of finite field elements. In [10], an alternative nesting scheme is developed to reduce the degree of the polynomials involved in the vectors for encoding.

Highly-parallel architectures are needed to realize the speed potential of GII codes. This paper focuses on parallel GII-BCH encoder design since it is more involved. Our proposed architectures can be easily extended to GII-RS codes. Parallel BCH encoders have been well-studied [11]–[18]. However, the design of parallel GII-BCH encoders faces many new challenges that do not exist for individual BCH encoders. Zeros need to be padded to the input polynomials of BCH encoders to make the total number of input coefficients a multiple of the parallelism, L. Due to the differences in the dimensions of the v nested codewords, different numbers of zeros need to be padded to the m BCH encoder inputs in the GII encoder and the outputs are generated at different clock cycles. On the other hand, the input and output polynomial coefficients need to be aligned according to their degrees when they are added up in the vector multiplications. Besides, the polynomials in the vectors to be multiplied to the BCH encoder inputs and outputs have different degrees according to the nesting scheme. This makes the coefficient alignment an even more challenging issue to solve.

In many digital communication and storage systems, the error-correcting encoder and decoder are not activated at the same time. In this case, the encoder can be shared to reduce the complexity of the decoder. It was proposed in [19] to utilize the multi-mode BCH encoder from [20] developed using z-transform to compute short remainder polynomials, from which the syndrome computation for the first step of BCH decoding can be greatly simplified. Such a resource-shareable BCH encoder can be adopted to realize resource-shareable GII encoders. In particular, the GII decoding consists of two stages: sub-codeword and nested decoding. Syndrome computation is the first step in both stages. The multi-round nested decoding accounts for the majority portion of the overall worst-case decoding latency, which needs to be shortened for time-constraint memory systems. A resource-shareable GII encoder would allow short reminder polynomials over different factors of the BCH generator polynomials to be computed simultaneously during the sub-codeword decoding.

The authors are with The Ohio State University, Columbus, OH 43210, U.S.

This material is based upon work supported by the National Science Foundation under Award No. 2011785.

The remainders can be stored and directly used in the nested decoding. As a result, such an encoder not only helps to reduce the decoder area but also substantially shortens the nested decoding latency.

The resource-shareable BCH encoder in [19] is serial and it has large iteration bound, which decides the minimum achievable critical path and hence the clock period. Although it was shown in [21] that the encoding scheme can be modified to reduce the iteration bound, the resulted parities are not protected. Developing an L-parallel BCH encoder through unfolding [22] further increases the iteration bound by L times. On the other hand, the register state look-ahead approach [12] that achieves parallel designs with much shorter critical path for a traditional BCH encoder implemented by a single linear feedback shift register (LFSR) can not be directly extended to the resource-shareable encoder in [19] since it consists of multiple concatenated LFSRs.

This paper first develops methodologies to solve the alignment issues of the coefficients for parallel GII-BCH encoding. Then two low-complexity parallel resource-shareable BCH encoder architectures to be utilized as GII-BCH encoder components are proposed. Both of our architectures are based on register state look-ahead computation and their critical paths are much shorter than that of the architecture derived by unfolding. In our first design, unlike that in previous work for a single LFSR, a formula is developed to describe the state update of the overall resource-shareable encoder with multiple concatenated LFSRs by analyzing the effects of the input and current register state on the next register state. Besides, the complexity reduction achievable by applying transformations on the concatenated LFSRs are analyzed and the structures of the involved matrices are given. Instead of depending on the z-transform, our second proposed parallel resource-shareable BCH encoder is achieved by reformulating the generator polynomial divisions to a set of iterative divisions by its factors. As a result, the critical path is further shortened and the complexity can be further reduced by modifying the input taps of each concatenated LFSR. The critical paths of both proposed designs are a small fraction of that of an unfolded version of the serial architecture in [19]. For an example 160-parallel GII-BCH ([8,3]) encoder over $GF(2^{12})$ with 90% code rate considered for Flash memory applications, the second proposed design requires 14% smaller area compared to the first one under the same timing constraint and both of them lead to around 50% latency reduction in the nested syndrome computation compared to a GII encoder using the best previous parallel BCH encoder [18].

The structure of this paper is as follows. In Section II, GII-BCH encoding and serial resource-shareable BCH encoders are introduced. Section III presents the coefficient alignment scheme for enabling parallel GII encoding. The two proposed parallel resource-shareable BCH encoders are detailed in Section IV and V. Complexity analyses and comparisons are given in Section VI and conclusions follow in Section VII.

II. Background

A $([m, v], n)$ GII-BCH code is defined by using $v+1$ BCH codes $\mathcal{C}_v(n, k_v) \subseteq \cdots \subseteq \mathcal{C}_1(n, k_1) \subset \mathcal{C}_0(n, k_0)$ of length n and dimension $k_v \leq \cdots \leq k_1 < k_0$ over $GF(2^q)$. A GII codeword consists of m sub-codewords, $c_0(x), c_1(x), \cdots, c_{m-1}(x)$, each of which is a codeword of $\mathcal{C}_0$. Besides, linear combinations of the sub-codewords as defined below are codewords of the stronger codes $\mathcal{C}_1, \cdots, \mathcal{C}_v$ [1], [2]

$$\begin{aligned}\mathcal{C} \triangleq &\{c(x) = [c_0(x), \cdots, c_{m-1}(x)] : c_i(x) \in \mathcal{C}_0, \\ &\tilde{c}_l(x) = \sum_{i=0}^{m-1} h_{l,i}(x) c_i(x) \in \mathcal{C}_{v-l}, 0 \leq l < v\}.\end{aligned} \tag{1}$$

In the above equation, $h_{l,i}(x)$ is the standard basis representation of α^{il}, where α is a primitive element of $GF(2^q)$.

$$H(x) = \begin{bmatrix} h_{0,0}(x) & h_{0,1}(x) & \cdots & h_{0,m-1}(x) \\ h_{1,0}(x) & h_{1,1}(x) & \cdots & h_{1,m-1}(x) \\ \vdots & \vdots & \cdots & \vdots \\ h_{v-1,0}(x) & h_{v-1,1}(x) & \cdots & h_{v-1,m-1}(x) \end{bmatrix}$$

is referred to as the nesting matrix.

Let $g_0(x), \cdots, g_v(x)$ be the generator polynomials of the $\mathcal{C}_0(n, k_0), \cdots, \mathcal{C}_v(n, k_v)$ BCH codes, respectively. Their degrees are denoted by $w_0, \cdots, w_v$. Systematic GII encoding can be carried out according to Algorithm 1 [2]. In this algorithm,'' denotes transpose. Enc$\{a(x), g(x)\}$ represents traditional systematic BCH encoding that computes the parity polynomial as the remainder of $a(x)x^{\deg(g(x))}$ divided by $g(x)$, which is denoted by $\text{Rem}(a(x)x^{\deg(g(x))})_{g(x)}$. Here $\deg(\cdot)$ means the degree of the polynomial. Each sub-codeword $c_i(x)$ of GII-BCH code is in the format of $[d_i(x), p_i(x)]$, where $d_i(x)$ and $p_i(x)$ are the data polynomial and parity polynomial, respectively. The encoding of $c_v(x), c_{v+1}(x), \cdots, c_{m-1}(x)$ is carried out as traditional BCH encoding and it is referred to as the first-level encoding in this paper. The encoding of $c_{v-1}(x), \cdots, c_1(x), c_0(x)$ is carried

Algorithm 1 Systematic GII-BCH encoding algorithm

Input :$d_0(x), \cdots, d_{m-1}(x)$
Output :$[d_0(x), p_0(x)], \cdots, [d_{m-1}(x), p_{m-1}(x)]$
1) First-level BCH encoding
For $i = m-1, m-2, \cdots, v$

$$\text{Enc}\{d_i(x), g_0(x)\} \rightarrow c_i(x) = [d_i(x), p_i(x)] \tag{2}$$

2) Higher-level BCH encoding
For $i = v-1, v-2, \cdots, 0$

- Compute $f_i(x)$

$$f_i(x) = \pi^{(i)}(x) \cdot [c_{i+1}(x), \cdots, c_{m-1}(x)]'. \tag{3}$$

- Generate $p_i^*(x)$

$$\text{Enc}\{d_i(x) + \mathcal{U}_{w_{v-i}}(f_i(x)), g_{v-i}(x)\} \rightarrow [d_i(x), p_i^*(x)]. \tag{4}$$

- Compute $p_i(x)$

$$p_i(x) = p_i^*(x) + \mathcal{L}_{w_{v-i}}(f_i(x)). \tag{5}$$

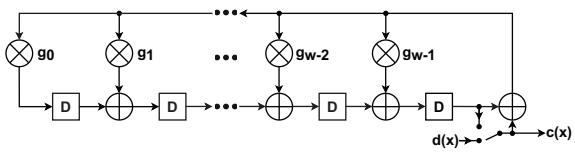


Fig. 1. Serial LFSR for (n, k) BCH encoding.

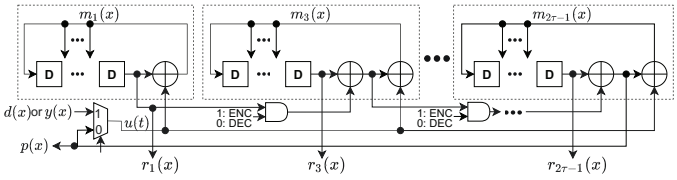


Fig. 2. Serial resource-shareable (n, k) BCH encoder architecture.

out according to (3)-(5) in Algorithm 1 and is called the higher-level encoding. In (3),

$$\pi^{(i)}(x) = (\Gamma^{(i)}(x))_i^{-1} \cdot \Theta^{(i)}(x) \mod g_i(x),$$

where $\Gamma^{(i)}(x)$ consists of row 0 through $i-1$ and column 0 through $i-1$ of $H(x)$ $(0 \le i < v)$ and $\Theta^{(i)}(x)$ is composed of row 0 through $i-1$ and column i through $m-1$ of $H(x)$. $(\Gamma^{(i)}(x))_i^{-1}$ denotes the i-th row of $(\Gamma^{(i)}(x))^{-1}$. $\mathcal{L}_a(\cdot)$ is the function that keeps the lowest a terms of the polynomial and $\mathcal{U}_b(\cdot)$ deletes the b lowest terms and divides x^b from the polynomial. Let $f(x) = f_{n-1}x^{n-1} + f_{n-2}x^{n-2} + \cdots + f_0$. Then $\mathcal{L}_a(f(x)) = f_{a-1}x^{a-1} + f_{a-2}x^{a-2} + \cdots + f_0$ and $\mathcal{U}_b(f(x)) = f_{n-1}x^{n-b-1} + f_{n-2}x^{n-b-2} + \cdots + f_b$. From Algorithm 1, the dimension of $c_i(x)$ is k_0 for $v \le i < m$ and k_{v-i} for $0 \le i < v$.

Let the error-correction capabilities of $\mathcal{C}_v, \cdots, \mathcal{C}_1, \mathcal{C}_0$ be $\tau_v \ge \cdots \ge \tau_1 > \tau_0$. GII decoding consists of two stages [2]. The first stage is traditional BCH decoding of individual sub-codewords $c_0(x), c_1(x), \cdots, c_{m-1}(x)$. The second-stage nested decoding is activated when some sub-codewords have more than τ_0 errors. In the η-th $(1 \le \eta \le v)$ nested decoding round, assume that δ_η sub-codewords remain to be corrected. First, higher-order syndromes of the nested codewords are calculated as $\tilde{S}_j^{(l)} = \tilde{y}_l(\alpha^{j+1})$ $(0 \le l < \delta_\eta, 2\tau_{\eta-1} \le j \le 2\tau_\eta - 1)$, where $\tilde{y}_l(x) = \sum_{i=0}^{m-1} h_{l,i}(x) y_i(x)$ and $y_i(x)$ is the i-th received sub-codeword. Then these nested syndromes are converted to higher-order syndromes $S_j^{(l)} = y_l(\alpha^{j+1})$ for those δ_η corrupted sub-codewords by reversing the linear combinations. From these higher-order syndromes, up to τ_η errors can be corrected in each of the δ_η sub-codewords. This process is repeated for up to $\delta_\eta \le v$ rounds. Since all the n coefficients of $\tilde{y}_l(x)$ are needed for the polynomial evaluation, the nested syndrome computation accounts for a significant part of the nested decoding latency [6].

Serial architectures for GII encoders have been proposed in [9], [10]. They consist of m serial BCH encoders, among which $m - v$ encoders are used to implement the first-level encoding in (2) and the rest v encoders are responsible for the computations in (4) for higher-level encoding. The inputs and outputs of the BCH encoders are multiplied with $\pi^{(i)}(x)$ according to (3). The degree of the polynomials in $\pi^{(i)}(x)$ is reduced by using a row-echelon nesting matrix in [10]. To take advantage of the high-speed potential of GII codes, parallel GII encoders are needed. However, parallel GII encoder designs have not been investigated so far although many work has been done on developing parallel BCH encoders.

Let $m_j(x)$ be the minimal polynomial of $GF(2^q)$ [23] that contains α^j as a root. The generator polynomial of a τ-error-correcting binary (n, k) BCH code over $GF(2^q)$ is $g(x) = LCM(m_1(x), m_3(x), \cdots, m_{2\tau-1}(x))$, where LCM means the least common multiple. Let $w = \deg(g(x)) = n-k$. Rewrite the generator polynomial as $g(x) = x^w + g_{w-1}x^{w-1} + \cdots + g_1 x^1 + g_0$, where $g_{w-1}, \cdots, g_1, g_0 \in GF(2)$. A serial LFSR architecture for implementing systematic BCH encoding is shown in Fig. 1. During the first k clock cycles, the coefficients of the message polynomial $d(x)$ is sent to the right-most tap of the LFSR starting from the most significant coefficient. After k clock cycles, the coefficients of $\text{Rem}(d(x)x^w)_{g(x)}$ are located in the registers. Then these coefficients are shifted out through the bottom switch in Fig. 1 to form the parity part of the codeword.

Parallel BCH encoders can be derived by applying look-ahead computation to the register state of the LFSR [11]–[18]. Denote the state of the registers in Fig. 1 in clock cycle t by $\mathbf{r}(t) = [r_{w-1}(t), r_{w-2}(t), \cdots, r_0(t)]'$. Let $u(t)$ be the input of the encoder at clock cycle t. The register state in the next clock cycle can be derived as

$$\mathbf{r}(t+1) = \mathbf{A}\mathbf{r}(t) + \mathbf{b}u(t), \tag{6}$$

where $\mathbf{b} = [g_{w-1}, g_{w-2}, \cdots, g_1, g_0]'$ and

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

is referred to as a companion matrix. Substitute (6) back to itself by p times, it can be derived that

$$\mathbf{r}(t+L) = \mathbf{A}^L\mathbf{r}(t) + \mathbf{B}_L\mathbf{u}_L(t), \tag{7}$$

where $\mathbf{B}_L = [\mathbf{A}^{L-1}\mathbf{b}, \cdots, \mathbf{A}\mathbf{b}, \mathbf{b}]$ and $\mathbf{u}_L(t) = [u(t), \cdots, u(t+L-2), u(t+L-1)]'$. From (7), an L-parallel encoder can be derived. The register state can be also transformed to $\mathbf{r}(t) = \mathbf{T} \times \mathbf{r}_T(t)$. Accordingly, $\mathbf{A}^L$ and $\mathbf{B}_L$ in (7) are replaced by $\mathbf{A}_{LT} = \mathbf{T}^{-1}\mathbf{A}^L\mathbf{T}$ and $\mathbf{B}_{LT} = \mathbf{T}^{-1}\mathbf{B}_L$, respectively [12]. Much work has been done to find a transformation matrix $\mathbf{T}$ to reduce the gate count and/or critical path of the matrix multiplications [12]–[17]. It was found in [18] that adding the input to a different tap of the serial LFSR leads to significant gate count reduction in the parallel LFSR without affecting the critical path. The input tap modification is also described by state transformation in [17].

In many digital storage and communication systems, the encoder and decoder are not activated at the same time. Hence, the encoder can be shared to simplify the computations in the decoder. By applying z-transform to the serial

LFSR in Fig. 1, the long LFSR for BCH encoding is decomposed into concatenated of short LFSRs for individual minimal polynomials [20]. Accordingly, the architecture in Fig. 2 [19] can be utilized to implement both encoding and remainder computation. When the second inputs of the AND gates are set to '1', this architecture is basically the multi-mode encoder from [20]. When those inputs are set to '0', the remainders are computed. In this case, since the input to each LFSR is added to the right-most tap, $r_j(x) = \text{Rem}(\mathcal{U}_{\deg(m_j(x))}(y(x))x^{\deg(m_j(x))})_{m_j(x)}$, where $y(x)$ is the received codeword, are calculated by the LFSRs. Since $\deg(\mathcal{L}_{\deg(m_j(x))}(y(x))) < \deg(m_j(x))$, $\text{Rem}(y(x))_{m_j(x)} = r_j(x) + \mathcal{L}_{\deg(m_j(x))}(y(x))$. Denote $\text{Rem}(y(x))_{m_j(x)}$ by $\bar{r}_j(x)$. If α^j is a root of $m_j(x)$, then $y(\alpha^j) = \bar{r}_j(\alpha^j)$. As a result, the syndromes for BCH decoding can be computed by evaluating $\bar{r}_j(x)$, which only has up to q instead of n coefficients as in $y(x)$.

Resource-shareable GII-BCH encoder can be achieved by using multi-mode BCH encoders with concatenated LFSRs to simplify the syndrome computation in GII decoding. The worst-case latency of GII decoding is largely decided by the multi-round nested decoding process and the nested syndrome computation accounts for a significant portion of the latency [6]. Fortunately, the generator polynomial $g_i(x)$ of $\mathcal{C}_i$ is a factor of $g_j(x)$ for $\mathcal{C}_j$ $(i < j)$. Hence, $g_j(x)$ can be decomposed into $g_i(x)$ and other factors. The remainders over those decomposed factors enable great reduction on the nested syndrome computation latency.

The design of parallel resource-shareable GII-BCH encoders faces two major challenges. First, serial GII-BCH encoders can not be directly extended to implement parallel GII-BCH encoders. The reason is that zeros need to be padded to the data coefficients for BCH encoding in the case that the number of coefficients is not an integer multiple of the parallelism, L. However, the numbers of data coefficients, $k_0, k_1, \cdots, k_v$, in the sub-codewords are different. Different numbers of zeros need to be padded to align the coefficients with the same degrees when polynomials need to be added up as required in (3), (4), and (5). Besides, the polynomials in the $\pi^{(i)}(x)$ vectors have different degrees. This makes the coefficient alignment even more complicated. The second challenge lies in the parallel design of multi-mode resource-shareable BCH encoders. Assuming $g(x)$ is decomposed into l factors, the iteration bound, T_∞, of the serial design in Fig. 2 consists of $l+1$ XOR gates, $l-1$ AND gates and 1 multiplexer. The minimum achievable clock period is lower bounded by T_∞. If unfolding [22] is applied to derive an L-parallel architecture, the minimum achievable clock period is $\lceil LT_\infty \rceil$. Although the register state update formula in (6) can be used to describe each individual LFSR in the multi-mode encoder of Fig. 2, the overall encoder can not be described by a similar formula. This is because that the registers for each LFSR are updated based on the contribution of all the registers located in each of the previous LFSRs and the cumulative sum of all the right-most register outputs is fed back to the inputs of every LFSR through the multiplexer. Accordingly it is difficult to apply the look-ahead computation to derive parallel architectures.

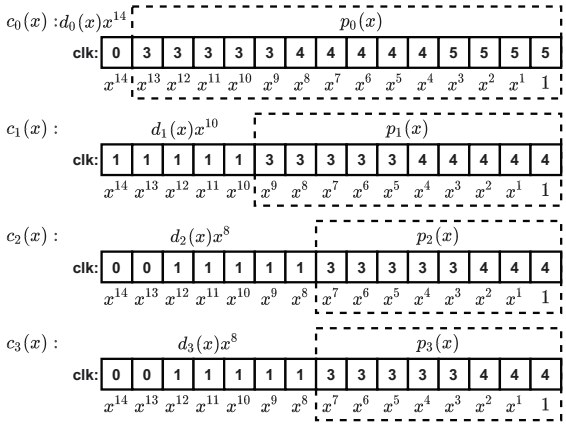


Fig. 3. Clock cycle numbers in which the coefficients of sub-codeword i are processed or generated by BCH encoder i $(0 \le i < 4)$ in parallel $([4,2],15)$ GII-BCH encoding with $L = 5$.

In the following, a polynomial coefficient alignment scheme for parallel GII-BCH encoding is first developed. Then two architectures for parallel multi-mode resource-shareable BCH encoders are proposed. In the first proposed design, novel state look-ahead formulas for the multi-mode BCH encoder are proposed to achieve a low-complexity parallel resource-shareable GII-BCH encoder with short critical path. By reformulating the remainder computation in multi-mode BCH encoding, our second GII-BCH encoder allows the input to be added to different taps of the concatenated LFSRs and accordingly achieves lower complexity and even shorter critical path for larger parallelisms.

III. Polynomial coefficient alignment for parallel GII-BCH encoding

For L-parallel encoding, the coefficients with the same degree of x should be added up in the $d_i(x) + \mathcal{U}_{w_{v-i}}(f_i(x))$ and $p_i^*(x) + \mathcal{L}_{w_{v-i}}(f_i(x))$ computations of (4) and (5), respectively. However, the data polynomial $d_i(x)$ has k_0 coefficients for $v \le i < m$ and k_{v-i} coefficients for $0 \le i < v$. Hence different numbers of zeros need to be padded before the most significant data coefficients to make the total number of coefficients sent to each BCH encoder a multiple of L. It was found in [18] that the parallel BCH encoder can be substantially simplified if the input is added to a different tap of the corresponding serial LFSR. This requires zeros to be padded after the least significant data coefficient and further complicates the coefficient alignment. Besides, the higher-level encoding needs to coordinate with the availability of the relevant coefficients from the first-level encoding taking into account the degrees of the polynomials in the $\pi^{(i)}(x)$ vectors.

The issues on the alignment of the coefficients for parallel GII encoding can be further explained using a toy code. Let us consider the encoder of a $([4,2],15)$ GII-BCH code over $GF(2^4)$ with $[\tau_0, \tau_1, \tau_2] = [2,3,4]$ and $L = 5$. The nesting matrix $H(x)$ in row-echelon form [10] below is adopted

$$H(x) = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & x & x^2 \end{bmatrix}.$$

In this case, $\pi^{(0)}(x) = [1,1,1]$ and $\pi^{(1)}(x) = [x, x^2]$. For this code, w_0, w_1, w_2 are $8, 10, 14$, respectively. Hence, $c_2(x)$ and

$c_3(x)$ have 8 parities each. They are generated by using BCH encoder 2 and 3, each of which is a $(15, 7)$ encoder. $c_1(x)$ and $c_0(x)$ have 10 and 14 parities, respectively. They are computed utilizing BCH encoder 1 and 0, which are a $(15, 5)$ encoder and a $(15, 1)$ encoder, respectively.

Each data or parity symbol in a codeword of this GII code is denoted by a square in Fig. 3. The digits in the squares are the clock cycle numbers in which the coefficients of $c_i(x)$ $(0 \leq i < 4)$ are processed or generated by BCH encoder i. For example, each of $c_2(x)$ and $c_3(x)$ has $n - w_0 = 7$ data symbols. Hence $L\lceil 7/L \rceil - 7 = 3$ zeros are padded before the most significant data coefficient. These inputs are processed in clock cycle 0 and 1. It was shown in [18] that the complexity of the matrix multiplications in parallel single-LFSR BCH encoders is minimized when the input is added to the L-th tap from the most significant tap (MST) in the corresponding serial LFSR, although L zeros need to be padded after the least significant data coefficient. When this input-tap modification is adopted, the padded zeros cost another clock cycle and hence the first $L = 5$ parity symbols in the encoding of $c_3(x)$ and $c_2(x)$ are generated in clock cycle 3 as shown in Fig. 3. For conciseness, the zero paddings are not shown in Fig. 3.

To generate $c_1(x)$, $[d_2(x), d_3(x)]'$ need to be multiplied with $\pi^{(1)}(x) = [x, x^2]$ according to (3). Then the higher coefficients of the product, $\mathcal{U}_{w_1}(f_1(x))$, are added to those of $d_1(x)$ according to (4) to produce the input to the $(15, 5)$ BCH encoder 1. Due to the multiplications of x and x^2, $\mathcal{U}_{w_1}(f_1(x))$ has 2 more coefficients than $d_1(x)$. Since $\lceil (2 + \deg(d_1(x)) + 1)/L \rceil - \lceil (\deg(d_1(x)) + 1)/L \rceil = 1$, the coefficients of $d_1(x)$ need to wait until clock cycle $0 + 1 = 1$ to be processed. Besides, $L\lceil (1+\deg(d_1(x))+1)/L \rceil - (1+\deg(d_1(x))+1) = 4$ and $L\lceil (2 + \deg(d_1(x)) + 1)/L \rceil - (2 + \deg(d_1(x)) + 1) = 3$ zeros need to be padded before the most significant coefficients of $d_2(x)$ and $d_3(x)$, respectively, due to the multiplications with x and x^2 in order to align the coefficients with the same degree of x in $\mathcal{U}_{w_1}(f_1(x))$ for the computations in (4). When the input is added to the MST of the LFSRs in BCH encoder 1, the first L coefficients of $p_1^*(x)$ are computed in clock cycle 2. However, they need to be delayed until the corresponding coefficients of $\mathcal{L}_{w_1}(f_1(x))$ are available as in (5). Hence, the first L coefficients of $p_1(x)$ are generated in clock cycle 3. Similarly, the clock cycle numbers in which the coefficients of $d_0(x)$ are processed and $p_0(x)$ are generated can be derived as shown in Fig. 3.

In the GII encoder, BCH encoder v through $m - 1$ are implemented by single LFSRs and the inputs of these encoders can be always added to the L-th tap instead of the MST to reduce the corresponding parallel encoder complexity. To enable resource-shareable GII encoding, BCH encoder 0 through $v - 1$ consist of concatenated LFSRs. Whether the inputs can be added to a none-MST in these encoders depends on the design as will be detailed in Section IV and V.

Because of the variations on the sub-codeword dimensions and polynomial degrees in the $\pi^{(i)}(x)$ vectors, the coefficients of $c_i(x)(1 \leq i < 4)$ may need to be delayed for different numbers of clock cycles before they are processed by BCH encoders for generating different sub-codewords as can be observed from Fig. 3. For example, the coefficient of x^{13} in

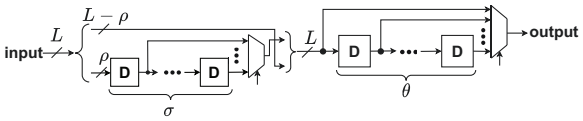


Fig. 4. Proposed L-parallel alignment block (AB)-(ρ, σ, θ)

$c_2(x)$ is processed in clock cycle 0 and those of x^{12} through x^9 are processed in clock cycle 1 in BCH encoder 2 to generate the parity part of $c_2(x)$. However, $c_2(x)$ is multiplied with x before it is used to form the input to BCH encoder 1 to compute $p_1(x)$. Therefore, the coefficients of x^{13} through x^9 from $c_2(x)$ should be added to the coefficients of x^{14} through x^{10} from $c_1(x)$ in clock cycle 1 to form the inputs of BCH encoder 1. This requires the coefficient of x^{13} from $c_2(x)$ to be delayed and combined with those of x^{12} through x^9. Additionally, it is possible that the coefficients to be processed in the same clock cycle for higher-level encoding are available in non-adjacent clock cycles. For example, in Fig. 3, the coefficient of x^8 of $c_2(x)$, which is available in clock cycle 1, and those of x^7 through x^4, which are available in clock cycle 3, form the $L = 5$ symbols that are added with the coefficients of x^9 through x^5 from $p_1^*(x)$ in clock cycle 3 to compute $p_1(x)$ according to (5).

All the alignment issues faced by parallel GII-BCH encoding can be addressed by the proposed alignment block (AB) shown in Fig. 4. It has three parameters. ρ is the number of split symbols on the bottom branch. σ and θ are the numbers of ρ-bit registers and L-bit registers on the left and right sides, respectively. The registers on the left part allow the coefficients available from different clock cycles to be put together into the same packet of L coefficients. The registers on the right part further hold the scrambled packet as needed before it is processed by the following computations.

Take the coefficients of $c_2(x)$ utilized for $c_0(x)$ encoding in Fig. 3 as an example. To compute $\mathcal{L}_{w_2}(f_0(x))$, since $\pi^{(0)} = [1, 1, 1]$, the coefficients of x^{13} through x^9 and x^8 through x^4 in $c_2(x)$ need to be added to the coefficients of the same degrees in $c_0(x)$ in clock cycles 3 and 4, respectively. For the coefficients of x^{13} through x^9, x^{13} is available in clock cycle 0 and the other four are available in clock cycle 1. Hence, the 5 input symbols are split to 1 and 4 symbols, and the symbols on the bottom go through 1-0=1 register. Since this packet is added to the symbols of $c_0(x)$ in clock cycle 3, the scrambled packet goes through 3-1=2 more registers. Hence, an alignment with $(\rho, \sigma, \theta) = (1, 1, 2)$ is needed. For the coefficients of x^8 through x^4, parts of the coefficients are data and the other parts are parities. When the input tap to the LFSR is shifted, the parities may not be generated in the clock cycle right after the data coefficients are sent in to decoder 2, as shown in the example of Fig. 3. These cases can be also handled by the proposed AB. For aligning this packet, since the coefficient of x^8 is available in clock cycle 1 and the others in this packet are generated in clock cycle 3, the bottom input symbols of the corresponding AB should go through 3-1=2 registers. These coefficients are added to those of $c_0(x)$ in clock cycle 4. Therefore, the symbols need to go through 4-3=1 more register. Accordingly, an alignment

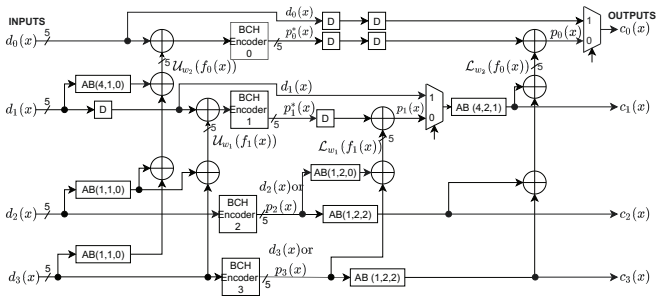


Fig. 5. Parallel GII-BCH $([4,2],15)$ encoder with $L=5$ corresponding to the clock cycle number chart in Fig. 3

with $(\rho,\sigma,\theta)=(1,2,1)$ is required for this packet. The alignments needed for different packets from the same sub-codeword have the same ρ value and registers can be shared. Therefore, take the maximum of the σ and θ values needed for different packets, a single AB can be utilized. Hence, an AB-$(\rho,\sigma,\theta)=(1,2,2)$ block is needed to align the coefficients of x^{13} through 1 of $c_2(x)$ for computing $\mathcal{L}_{w_2}(f_0(x))$, and the control signals of the multiplexers in the AB change according to the number of delays needed.

Once the clock cycle number chart is decided, a parallel GII encoder can be designed utilizing the proposed AB shown in Fig. 4. Take the clock cycle number chart in Fig. 3 as an example, the corresponding parallel GII-BCH encoder architecture is developed as illustrated in Fig. 5. The ABs on the left of the BCH encoders are used to handle the coefficient alignment for $\mathcal{U}_{w_{v-i}}(f_i(x))$ computation and those to the right are responsible for aligning the coefficients for $\mathcal{L}_{w_{v-i}}(f_i(x))$ calculation. As explained in the previous paragraph, an AB-$(1,2,2)$ is utilized to align the $c_2(x)$ coefficients for $\mathcal{L}_{w_2}(f_0(x))$ calculation. Take the coefficient alignment of $c_1(x)$ for $\mathcal{L}_{w_2}(f_0(x))$ calculation as another example. Aligning the coefficients of x^{13} through x^9 with those of $c_0(x)$ requires an AB-$(4,2,0)$ and the alignment for those of x^8 through 1 needs an AB-$(4,1,0)$. By taking the maximum of the σ and θ values of these ABs, the coefficient alignment of $c_1(x)$ for $\mathcal{L}_{w_2}(f_0(x))$ calculation can be handled by an AB-$(4,2,0)$. By following the same procedure, the parameters for the rest of the ABs located at the both sides of Fig. 5 can be determined.

To facilitate the storage or transmission of the codeword, the last data coefficient packet should be output in the clock cycle right before the first parity coefficient packet is generated. Besides, the coefficient of the same degree from each sub-codeword should appear at the output in the same clock cycle. Such synchronization can be handled by adjusting the parameters of the ABs and/or adding delay elements. Take the encoder in Fig. 5 as an example. The data coefficients for $c_0(x)$ are sent to BCH encoder 0 in clock cycle 0. They are delayed by two clock cycles so that they appear as the systematic part of $c_0(x)$ at the output in clock cycle 2 right before the first packet of parity coefficients appear in clock cycle 3. To synchronize the coefficients of $c_1(x)$ with those of $c_0(x)$ of the same degree at the output of the GII encoder, the data coefficients of $c_1(x)$ need to be delayed for one more clock cycle since they are sent to the BCH encoder in clock cycle 1 according to Fig. 3. This additional delay can be realized by increasing the θ parameter of the AB at the output of $c_1(x)$ by one. As a result, an AB-$(4,2,1)$ is utilized as shown in Fig. 5. The data coefficients of $c_2(x)$ and $c_3(x)$ also need to be delayed by two clock cycles to be synchronized at the outputs. These delays can be implemented by the AB-$(1,2,2)$ blocks since they already have $\theta=2$ registers on the right side of Fig. 4. Hence, no further adjustment is needed.

To reduce the complexity, the ABs with the same input signal can share registers. BCH encoders v through $m-1$ in a GII-BCH encoder pass their inputs to be the systematic parts of their outputs. Hence, the ABs connected to the input and output of each of these encoders actually have the same input signals and they can share registers. For example, the AB-$(1,1,0)$, AB-$(1,2,0)$, and AB-$(1,2,2)$ connected to the input and output of BCH encoder 2 in Fig. 5 can share registers. $\max\{\sigma\}=\max\{1,2,2\}=2$ registers are needed to delay the split input coefficients and $\max\{\theta\}=\max\{0,0,2\}=2$ are required to further delay the packets.

IV. Parallel resource-shareable GII-BCH encoder based on register state look-ahead

A parallel resource-shareable GII encoder can be achieved by employing parallel resource-shareable BCH encoders. However, the design of such parallel BCH encoders have not been previously investigated. Unfolding the serial resource-shareable BCH encoder in [19] to derive a parallel architecture would lead to overwhelmingly long critical path when the parallelism is large. Besides, although look-ahead computation can be applied to (6) to derive parallel architectures for a single LFSR, it is not applicable to the concatenated LFSRs in Fig. 2. This is because, unlike that for individual LFSRs, the update of the registers in an LFSR in Fig. 2 is dependent on the other LFSRs. To derive a parallel architecture for resource-shareable BCH encoders, a single formula describing the update of all registers in every LFSR of Fig. 2 is needed.

Assume that the generator polynomial of the BCH code is decomposed into l shorter polynomials as $g(x)=\beta_0(x)\beta_1(x)\cdots\beta_{l-1}(x)$ and $\deg(\beta_j(x))=\gamma_j$ $(0\le j<l)$. The architecture in Fig. 2 can be utilized to implement the serial encoding of this code if the j-th LFSR from the left is configured according to $\beta_j(x)$. Let $\beta_j(x)=x^{\gamma_j}+\beta^{(j)}_{\gamma_j-1}x^{\gamma_j-1}+\cdots+\beta^{(j)}_0$, where $\beta^{(j)}_{\gamma_j-1},\cdots,\beta^{(j)}_0$ are binary bits. Let $\mathbf{b}^{(j)}=[\beta^{(j)}_{\gamma_j-1},\beta^{(j)}_{\gamma_j-2},\cdots,\beta^{(j)}_0]'$ and $\mathbf{A}^{(j)}$ be a $\gamma_j\times\gamma_j$ companion matrix whose left-most column is $\mathbf{b}^{(j)}$. $\mathbf{C}^{(j)}_i$ is a $\gamma_j\times\gamma_i$ matrix whose left-most column is also $\mathbf{b}^{(j)}$ for $j=1,2,\cdots,l-1$ and $0\le i<j$, but the rest columns are zero. Also denote the register state at clock cycle t by $\mathbf{r}(t)=[\mathbf{r}^{(l-1)}(t),\mathbf{r}^{(l-2)}(t),\cdots,\mathbf{r}^{(0)}(t)]$, where $\mathbf{r}^{(j)}(t)$ includes the state of the γ_j registers in the j-th LFSR.

From Fig. 2, the register state at clock cycle $t+1$ is dependent on the register state and the multiplexer output at clock cycle t. Denote the multiplexer output by $u(t)$. The register update can be described in general as

$$\mathbf{r}(t+1)=\widehat{\mathbf{A}}\mathbf{r}(t)+\widehat{\mathbf{b}}u(t). \tag{8}$$

The current register state affects the next register state through the output of the first XOR gate located at the output of the

right-most register in each LFSR. The horizontal input of this XOR gate decides the contribution of the registers in the same LFSR, which can be described by $\mathbf{A}^{(j)}$. The vertical input of this XOR gate determines the effect of the registers from the other LFSRs, which can be represented by $\mathbf{C}_i^{(j)}$. Accordingly, $\widehat{\mathbf{A}}$ is in the format of

$$\widehat{\mathbf{A}} = \begin{bmatrix} \mathbf{A}^{(l-1)} & \mathbf{C}_{l-2}^{(l-1)} & \cdots & \mathbf{C}_1^{(l-1)} & \mathbf{C}_0^{(l-1)} \\ \mathbf{0} & \mathbf{A}^{(l-2)} & \cdots & \mathbf{C}_1^{(l-2)} & \mathbf{C}_0^{(l-2)} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{A}^{(1)} & \mathbf{C}_0^{(1)} \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{A}^{(0)} \end{bmatrix} \quad (9)$$

Since $u(t)$ contributes to each LFSR separately through the right-most XOR gate, $\widehat{\mathbf{b}} = [(\mathbf{b}^{(l-1)})', (\mathbf{b}^{(l-2)})', \cdots, (\mathbf{b}^{(0)})']'$. For an (n, k) serial BCH encoder, $u(t)$ equals to the data input in the first k clock cycles and is connected to the feedback from the registers in the next $n-k$ clock cycles when the parities are shifted out. Since the feedback is the XOR result of the right-most XOR gate in every LFSR, $u(t)$ can be described as

$$u(t) = \begin{cases} d(t) \text{ in the first } k \text{ clock cycles} \\ \mathbf{I}_\beta \cdot \mathbf{r}(t) \text{ in the rest } n-k \text{ clock cycles} \end{cases}. \quad (10)$$

Here $d(t)$ is the t-th data symbol, and $\mathbf{I}_\beta = [1, \underbrace{0, \cdots, 0}_{\gamma_{l-1}-1}|1, \underbrace{0, \cdots, 0}_{\gamma_{l-2}-1}|\cdots|1, \underbrace{0, \cdots, 0}_{\gamma_0-1}]$.

By substituting (8) back to itself L times, a formula for L-parallel resource-shareable BCH encoding is derived as

$$\mathbf{r}(t+L) = \widehat{\mathbf{A}}^L \mathbf{r}(t) + \widehat{\mathbf{B}}_L \mathbf{u}_L(t), \quad (11)$$

where $\widehat{\mathbf{B}}_L = [\widehat{\mathbf{A}}^{L-1}\widehat{\mathbf{b}}, \cdots, \widehat{\mathbf{A}}\widehat{\mathbf{b}}, \widehat{\mathbf{b}}]$. In the first $\lceil k/L \rceil$ clock cycles, the data coefficients are processed and $\mathbf{u}_L(t) = [u(t), u(t+1), \cdots, u(t+L-1)]' = [d(t), d(t+1), \cdots, d(t+L-1)]'$. In later clock cycles, the entries of $\mathbf{u}_L(t)$ are decided by the register state. To achieve parallel processing, $\mathbf{u}_L(t)$ should be written in the format such that it only depends on $\mathbf{r}(t)$. From (10), $u(t+1) = \mathbf{I}_\beta \mathbf{r}(t+1)$. Replacing $\mathbf{r}(t+1)$ by the formula in (8), it can be derived that $u(t+1) = \mathbf{I}_\beta(\widehat{\mathbf{A}} + \widehat{\mathbf{b}}\mathbf{I}_\beta)\mathbf{r}(t)$. The formulas for $u(t+2), u(t+3), \cdots$ in terms of $\mathbf{r}(t)$ can be derived in a similar way. As a result,

$$\mathbf{u}_L(t) = \begin{cases} \mathbf{d}_L(t) & \text{in the first } \lceil k/L \rceil \text{ clock cycles} \\ \mathbf{V} \cdot \mathbf{r}(t) & \text{in the rest } \lceil (n-k)/L \rceil \text{ clock cycles} \end{cases}, \quad (12)$$

where $\mathbf{V} = [\mathbf{v}_0', \mathbf{v}_1', \cdots, \mathbf{v}_{L-1}']'$ and $\mathbf{v}_\eta = \mathbf{I}_\beta \cdot (\widehat{\mathbf{A}} + \widehat{\mathbf{b}} \cdot \mathbf{I}_\beta)^\eta$ $(0 \le \eta < L)$.

Inspired by the schemes in [15]–[17], the register state can be transformed to $\mathbf{r}(t) = \mathbf{T} \cdot \mathbf{r}_T(t)$ to reduce the complexity of the pre-processing matrix, $\widehat{\mathbf{B}}_L$, in parallel resource-shareable encoders. Here $\mathbf{T}$ is an invertible matrix. For a traditional BCH encoder, using transformation matrix $\mathbf{T} = \mathbf{A}^L$ [17] converts (7) to $\mathbf{r}_T(t+L) = \mathbf{A}^L \mathbf{r}_T(t) + (\mathbf{T}^{-1}\mathbf{B}_L)\mathbf{u}_L(t)$. For long BCH codes, L is usually less than the length of the generator polynomial. In this case, $\mathbf{A}^L$ has L dense columns since $\mathbf{A}$ is a companion matrix. More importantly, using $\mathbf{T} = \mathbf{A}^L$ makes $\mathbf{T}^{-1}\mathbf{B}_L$ consist of only columns with single nonzero entries.

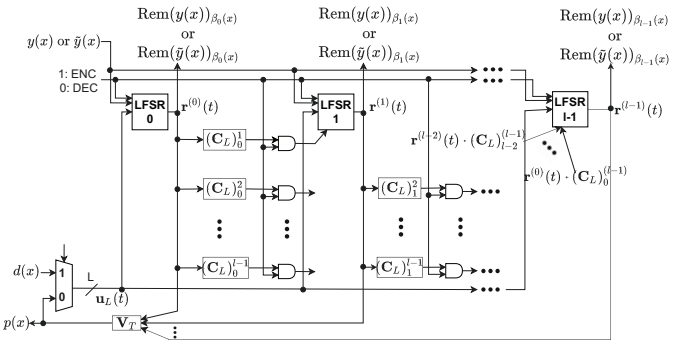


Fig. 6. Proposed parallel resource-shareable BCH encoder based on register state look-ahead

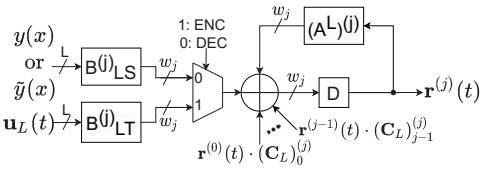


Fig. 7. Architecture for the j-th parallel LFSR in resource-shareable encoder

As a result, using $\mathbf{T} = \mathbf{A}^L$ leads to one of the simplest designs for parallel single LFSR. In the case of the resource-shareable BCH encoder that consists of multiple concatenated LFSRs, $\mathbf{A}$ becomes $\widehat{\mathbf{A}}$. Although $\widehat{\mathbf{A}}$ itself is not a companion matrix, it consists of blocks of companion matrices and matrices with single nonzero columns. As a result, using $\mathbf{T} = \widehat{\mathbf{A}}^L$ as the transformation matrix for parallel resource-shareable encoder would intuitively also lead to great complexity reduction.

When $\mathbf{T} = \widehat{\mathbf{A}}^L$, (11) becomes

$$\mathbf{r}_T(t+L) = \widehat{\mathbf{A}}^L \mathbf{r}_T(t) + \widehat{\mathbf{B}}_{LT} \mathbf{u}_L(t), \quad (13)$$

where $\widehat{\mathbf{B}}_{LT} = \widehat{\mathbf{A}}^{-L}\widehat{\mathbf{B}}_L$. Besides, the bottom formula for $\mathbf{u}_L(t)$ in (12) becomes

$$\mathbf{u}_L(t) = \mathbf{V}r(t) = \mathbf{V}\mathbf{T} \cdot \mathbf{r}_T(t). \quad (14)$$

Accordingly, the $\mathbf{T}$ matrix multiplication for reversing the register transformation can be combined with the $\mathbf{V}$ matrix multiplication as a single constant matrix multiplication by $\mathbf{V}_T = \mathbf{V}\mathbf{T}$. Rewrite $\widehat{\mathbf{B}}_{LT}$ as $[(\mathbf{B}_{LT}^{(l-1)})', (\mathbf{B}_{LT}^{(l-2)})', \cdots, (\mathbf{B}_{LT}^{(0)})']'$, where the dimension of $(\mathbf{B}_{LT}^{(j)})$ is $\gamma_j \times L$. From simulations, it was found that $(\mathbf{B}_{LT}^{(j)})$ for $1 \le j < l$ are all zero when $\gamma_0 \ge L$. Besides, column i of $\mathbf{B}_{LT}^{(0)}$ $(0 \le i < L)$ has a single nonzero entry at row i. Intuitively, this is analogues to the simplification on the pre-processing matrix achieved by using $\mathbf{T} = \mathbf{A}^L$ for single LFSR.

To construct a parallel resource-shareable BCH encoder, each parallel LFSR needs to have its own register update equation. Divide $\widehat{\mathbf{A}}^L$ into submatrices $(\mathbf{A}^L)^{(j)}$ and $(\mathbf{C}_L)_i^{(j)}$ in the same manner as that in (9). Multiplying out the

components of the matrices in (13), it can be derived that

$$\begin{aligned}
\mathbf{r}_T^{(l-1)}(t+L) =&(\mathbf{A}^L)^{(l-1)}\mathbf{r}_T^{(l-1)}(t)+\mathbf{B}_{LT}^{(l-1)}\mathbf{u}_L(t)+(\mathbf{C}_L)_{l-2}^{(l-1)}\mathbf{r}_T^{(l-2)}(t)+\\
&\cdots+(\mathbf{C}_L)_1^{(l-1)}\mathbf{r}_T^{(1)}(t)+(\mathbf{C}_L)_0^{(l-1)}\mathbf{r}_T^{(0)}(t)\\
&\vdots\\
\mathbf{r}_T^{(1)}(t+L) =&(\mathbf{A}^L)^{(1)}\mathbf{r}_T^{(1)}(t)+\mathbf{B}_{LT}^{(1)}\mathbf{u}_L(t)+(\mathbf{C}_L)_0^{(1)}\mathbf{r}_T^{(0)}(t)\\
\mathbf{r}_T^{(0)}(t+L) =&(\mathbf{A}^L)^{(0)}\mathbf{r}_T^{(0)}(t)+\mathbf{B}_{LT}^{(0)}\mathbf{u}_L(t)
\end{aligned}, \tag{15}$$

From (15), an L-parallel resource-shareable BCH encoder architecture can be developed as shown in Fig. 6. The details of each LFSR block are shown in Fig. 7. For encoding, the output of the LFSRs are multiplied with the $(\mathbf{C}_L)_i^{(j)}$ matrices and the results are added to later LFSRs according to (15). For the remainder computations needed in the decoding, the second inputs of the AND gates in Fig. 6 are '0', and each LFSR operates separately. As it was mentioned previously, $\mathbf{B}_{LT}^{(j)}$ for $1 \le j < l$ are zero. Hence, the $\mathbf{B}_{LT}^{(j)}$ multiplication blocks in LFSR 1 through $l-1$ can be deleted. The received polynomial is not multiplied with any power of x in the remainder computation for decoding. This is equivalent to adding the input to the leftmost tap in the serial LFSR architecture in Fig. 1. Accordingly, the pre-processing matrix in the j-th LFSR of Fig. 6 becomes $\mathbf{B}_{LS}^{(j)}$, which has one single nonzero entry in each column when $L \le \gamma_j$ and consists of a $\gamma_j \times \gamma_j$ identity part when $L > \gamma_j$ [18].

Two types of paths need to be considered to decide the critical path of the proposed encoder in Fig. 6. The first type starts from the outputs of the LFSRs, goes through the $(\mathbf{C}_L)_i^{(j)}$ matrix multiplications and AND gates, and ends at the registers in the LFSRs after the XOR tree. The second type starts from the output of the last LFSR, goes through the $\mathbf{V}_T$ matrix multiplication and the multiplexer to generate $\mathbf{u}_L(t)$, and then passes the computation units in the LFSRs. The relative lengths of these two types of paths are decided by the data paths of $(\mathbf{C}_L)_i^{(j)}$ vs $\mathbf{V}_T$ matrix multiplications. From simulations, $(\mathbf{C}_L)_i^{(j)}$ has $\min(L, \gamma_i)$ nonzero columns. Hence, the maximum number of nonzero columns among all the $(\mathbf{C}_L)_i^{(j)}$ matrices is $\max(\min(L, \gamma_i))$. Decompose $\mathbf{V}_T$ into $\mathbf{V}_T = [\mathbf{V}_T^{(l-1)}, \mathbf{V}_T^{(l-2)}, \cdots, \mathbf{V}_T^{(0)}]$, where $\mathbf{V}_T^{(j)}$ is a $L \times \gamma_j$ binary matrix. Simulations also showed that $\mathbf{V}_T^{(j)}$ $(0 \le j < l)$ consists of $\min(2L, \gamma_j)$ nonzero columns. Since $\sum_{j=0}^{l-1} \min(2L, \gamma_j) > \max(\min(L, \gamma_i))$, the second type of paths is the critical path of the overall encoder.

The above design is developed by applying state look-ahead on the serial multi-mode encoder in Fig. 2. The input of each LFSR in this figure can only be added to the MST. If the input is added to an alternative tap, the signal sent to the next LFSR would be different and the corresponding output $p(x)$ would not be the correct parities. As a result, the input-tap modification technique in [18] can not be employed to reduce the number of '1's in $(\mathbf{C}_L)_i^{(j)}$. For larger γ_j, $(\mathbf{C}_L)_i^{(j)}$ matrix multiplications account for a significant portion of the overall encoder area.

V. Parallel resource-shareable GII-BCH encoder based on algorithmic reformulation

Although our first parallel resource-shareable BCH encoder proposed in the previous section can achieve much shorter critical path than the unfolded version of [19], it has two limitations. First, as mentioned earlier, the input of each decomposed LFSR in Fig. 2 must be added to the MST and the complexities of $(\mathbf{C}_L)_i^{(j)}$ matrix multiplications can not be simplified. Through algorithmic reformulations, our second design proposed in this section allows input-tap modifications and makes some of the $\mathbf{C}_i^{(j)}$ $(0 \le i < j, 1 \le j < l)$ in (9) have a single nonzero entry. This substantially reduces the number of '1's in the $\widehat{\mathbf{A}}^L$ matrix for parallel processing. Secondly, as discussed previously, the longest path in Fig. 6 is the feedback path that includes the $\mathbf{V}_T$ matrix multiplication. By eliminating the $\mathbf{V}_T$ matrix multiplication, our second design also has shorter critical path. Although polynomial multiplication post-processing is needed, the overall area of our second encoder is substantially smaller than that of the first proposed design in the case of larger L for high-speed applications.

The BCH encoding with generator polynomial $g(x) = \beta_0(x)\beta_1(x)\cdots\beta_{l-1}(x)$ does not have to be decomposed by utilizing the z-transform as in [20]. Assume that

$$\begin{aligned}
d(x)x^{\gamma_0} =& q_0(x)\beta_0(x) + p_0(x)\\
q_0(x)x^{\gamma_1} =& q_1(x)\beta_1(x) + p_1(x)\\
&\vdots\\
q_{l-2}(x)x^{\gamma_{l-1}} =& q_{l-1}(x)\beta_{l-1}(x) + p_{l-1}(x),
\end{aligned} \tag{16}$$

where $q_i(x)$ and $p_i(x)$ are the quotient and reminder, respectively, from the above division by $\beta_i(x)$. Substitute the second equation of (16) into the first one scaled by x^{γ_1}.

$$d(x)x^{\gamma_0+\gamma_1} = q_1(x)\beta_0(x)\beta_1(x) + p_0(x)x^{\gamma_1} + p_1(x)\beta_0(x).$$

By repeating a similar scaling and substitution process for the rest of the equations in (16), it can be derived that

$$d(x)x^w = q_{l-1}(x)\beta_0(x)\beta_1(x)\cdots\beta_{l-1}(x) + p(x),$$

where

$$\begin{aligned}
p(x) =& \Bigg(\cdots\Big(\big(p_0(x)x^{\gamma_1} + p_1(x)\beta_0(x)\big)x^{\gamma_2} + \cdots\Big)\cdots\Bigg)x^{\gamma_{l-1}}\\
&+ p_{l-1}(x)\beta_0(x)\beta_1(x)\cdots\beta_{l-2}(x).
\end{aligned} \tag{17}$$

and $w = \gamma_0 + \gamma_1 + \cdots + \gamma_{l-1}$. It can be easily proved that the degree of the $p(x)$ computed in (17) is less than w. Hence, $p(x)$ equals the remainder polynomial, $\mathrm{Rem}(d(x)x^w)_{g(x)}$, needs to be computed by the BCH encoding.

Our proposed reformulated resource-shareable BCH encoding consists of two stages. The first stage implements the polynomial divisions according to (16). The second stage utilizes the remainders generated from the first stage to compute the parities according to (17). A serial architecture for the first stage is shown in Fig. 8. It shares similar concatenated LFSRs as the one in Fig. 2. However, the input $u(t)$ is only sent to the first LFSR and there is no feedback from the last LFSR to

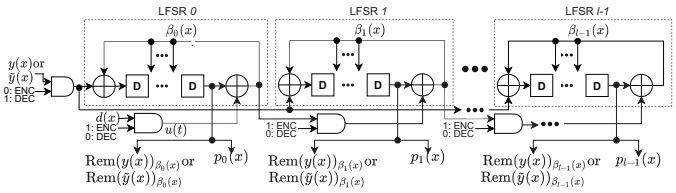


Fig. 8. Stage 1 of the proposed serial resource-shareable BCH encoder based on algorithmic reformulation.

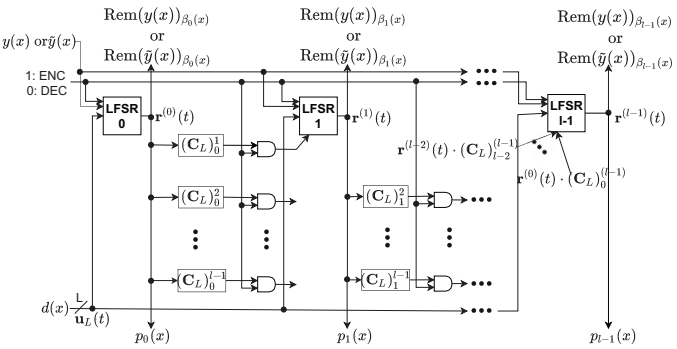


Fig. 9. Stage 1 of the proposed L-parallel resource-shareable BCH encoder based on algorithmic reformulation.

the previous LFSRs. The polynomial multiplication needed for the second stage can be implemented by architectures similar to that of finite impulse response (FIR) filters [22].

In Fig. 8, although $u(t)$ is not sent to every LFSR, it is included in the output of first LFSR. The inputs of the right-most XOR gate in every LFSR consist of $u(t)$ and the outputs of the right-most registers of all the previous LFSRs. Since these inputs are the same as those of the right-most XOR gate in each LFSR shown in Fig. 2, the state update of all registers in Fig. 8 can be described by a single formula as in (8) with the same $\widehat{\mathbf{A}}$ and $\widehat{\mathbf{b}}$. Accordingly, the formula for L-parallel processing of our second encoder is the same as (11), except that $\mathbf{u}_L(t)$ is simplified to $\mathbf{d}_L(t)$. Similarly, $\widehat{\mathbf{A}}^L$ and $\widehat{\mathbf{B}}_L$ can be decomposed to blocks of submatrices, and a set of equations in the same format as (15) can be used to describe parallel processing. As a result, a parallel architecture for the first stage of the encoder can be developed as shown in Fig. 9.

The equations in (16) can be reformulated to further reduce the parallel encoder complexity. For a LFSR whose length is w, $d(x)x^w$ is divided by the configuration polynomial of the LFSR when the input $d(x)$ is added to the MST. Adding the input to the δ-th tap from the MST implements the division by $d(x)x^{w-\delta}$. Nevertheless, the same result of dividing $d(x)x^w$ can be derived by padding δ zeros after the least significant coefficient of $d(x)$, which effectively carries out the division on $(d(x)x^\delta)x^{w-\delta} = d(x)x^w$. It was found in [18] that adding the input to a different tap of the LFSR changes the $\mathbf{b}$ vector in (6) and leads to a much simpler $\mathbf{B}_L$ matrix for parallel processing. To take advantage of the input-tap modification, our second resource-shareable BCH encoder multiplies the divisor in the j-th ($0 \le j < l$) equation of (16) by $x^{-\delta_j}$, which translates to shifting the input of the j-th LFSR by δ_j taps. The same parity polynomial $p(x)$ is computed provided that $\delta = \sum_{j=0}^{l-1} \delta_j$ zeros are padded to $d(x)$ and the formula in (17) is modified to

$$p(x) = \left(\cdots \left(\left(\hat{p}_0(x) x^{\gamma_1 - \delta_1} + \hat{p}_1(x)\beta_0(x) \right) x^{\gamma_2 - \delta_2} + \cdots \right) \cdots \right) x^{\gamma_{l-1} - \delta_{l-1}} + \hat{p}_{l-1}(x)\beta_0(x)\beta_1(x)\cdots\beta_{l-2}(x), \tag{18}$$

where $\hat{p}_j(x)$ is the remainder generated by LFSR j whose input is shifted by δ_j taps. Note that these $\hat{p}_j(x)$ are different from the $p_j(x)$ in (16). All the δ zeros are padded to $d(x)$, the input of LFSR 0. No zero is padded to the inputs of the other LFSRs.

Unlike the first proposed resource-shareable BCH encoder, the input to LFSR j can be shifted by $\delta_j > 0$ taps in our second design. The $\mathbf{C}_i^{(j)}$ submatrices in the $\widehat{\mathbf{A}}$ matrix of (9) describe how the current state of the registers affect the next state. If the addition of the input to LFSR j for $1 \le j < l$ in Fig. 9 is shifted by $0 < \delta_j \le L$ taps, each $\mathbf{C}_i^{(j)}$ ($0 \le i < j$) becomes a matrix that has a single nonzero entry in the δ_j-th row of the first column. Moreover, all $\mathbf{C}_i^{(j')}$ with $j' > j$ and $0 \le i < j$ become zero. These matrices are much simpler than the $\mathbf{C}_i^{(j)}$ with a dense column in the first proposed design. Such sparse $\mathbf{C}_i^{(j)}$ submatrices lead to simplified $(\mathbf{C}_L)_i^{(j)}$ submatrices for parallel processing. It can be derived that the lowest complexity of $(\mathbf{C}_L)_i^{(j)}$ is achieved by shifting the input to the j-th LFSR by L taps if $\gamma_j \ge L$ and by γ_j taps when $\gamma_j < L$. The latter case is equivalent to adding the input to the least significant tap of the LFSR. Following similar analysis, it can be found that the complexity of the $\widehat{\mathbf{B}}_L$ matrix multiplication in our second design is similar to that in the first proposed encoder if the input to LFSR 0 is shifted by L or more taps. Besides, although this second design requires polynomial multiplications in the second stage, it does not need the $\mathbf{V}_T$ matrix multiplication. Overall, reformulating the encoding process and modifying the input taps of the LFSRs lead to significant complexity reduction in the L-parallel design, especially when L is large. Also due to the elimination of the $\mathbf{V}_T$ matrix, the critical paths of the overall parallel architecture now lie in the $(\mathbf{A}^L)^{(j)}$ matrix multiplications in the LFSRs of Fig. 7. The other components of the encoder can be pipelined. Since $(\mathbf{A}^L)^{(j)}$ has smaller dimension than $\mathbf{V}_T$, the critical path of our second proposed L-parallel resource-shareable BCH encoder is also shorter than that of our first design.

To avoid additional coefficient alignment issues, the total number of zeros, δ, to be padded after the least significant coefficient of the data input, $d(x)$, should be an integer multiple of L. If the last packet of data coefficients are sent to the parallel BCH encoder in clock cycle a and the first packet of parity coefficients are generated in clock cycle b, it means that $\delta = (b - a - 1)L$ zeros are padded. As mentioned in the previous paragraph, for $j > 0$, δ_j is set to L if $\gamma_j \ge L$ and to γ_j if $\gamma_j < L$. On the other hand, setting δ_0 to L or larger greatly simplifies $\widehat{\mathbf{B}}_L$, whose multiplication contributes more to the encoder area compared to that of $(\mathbf{C}_L)_i^{(j)}$. If $\delta_0 = \delta - \sum_{j=1}^{l-1} \delta_j$ is smaller than L, then $b - a - 1$ should be increased to the smallest value such that $\delta_0 \ge L$. The

TABLE I
COMPLEXITIES AND CRITICAL PATHS OF 160-PARALLEL RESOURCE-SHAREABLE BCH ENCODERS

	poly	[18]	[19] unfolded*	Prop. enc. 1*	Prop. enc. 2*
# of XORs	$g_3(x)$	60069 (0.50)	58150 (0.48)	120902 (1.00)	98431 (0.81)
	$g_2(x)$	39941 (0.54)	41786 (0.57)	73371 (1.00)	70962 (0.97)
	$g_1(x)$	33313 (0.63)	35152 (0.66)	53246 (1.00)	46598 (0.88)
critical path (# of gates)	$g_3(x)$	8	800	11	9
	$g_2(x)$	8	800	11	9
	$g_1(x)$	8	800	10	9
Rem. length for nest. syn. comp.		690	354	354	354

*: $g_i(x)$ of the encoder is decomposed into $g_i(x) = g'_i(x)g_0(x)$.

parameters of the ABs need to be adjusted accordingly for different values of $b - a - 1$.

VI. COMPLEXITY ANALYSES AND COMPARISONS

This section analyzes the complexities of the two proposed encoders using a $([8, 3], 4095)$ GII-BCH code over $GF(2^{12})$ with $[t_0, t_1, t_2, t_3] = [28, 32, 39, 58]$ as an example. This code is considered because its overall codeword length is around 4kB and the code rate is 90%, which are required for Flash memory applications. For this code, the degrees of the generator polynomials, $g_0(x), g_1(x), g_2(x), g_3(x)$, of the involved BCH codes are $[w_0, w_1, w_2, w_3] = [336, 384, 462, 690]$. The parallelism of the encoder is set to $L = 160$ in order to achieve terabit/s throughput. Our encoders are also compared with the best possible alternative designs in this section.

The encoder for this GII-BCH code with $m = 8$ and $v = 3$ has 8 BCH encoders. Encoder 3 through 7 implement BCH encoding using $g_0(x)$. Encoder 0 through 2 are for BCH codes with generator polynomials $g_3(x)$, $g_2(x)$, and $g_1(x)$, respectively. Note that $g_i(x)$ is a factor of $g_j(x)$ if $i < j$. The proposed resource-shareable BCH encoder designs can be applied to any decomposition of the generator polynomial. However, splitting a generator polynomial into more parts leads to higher complexity. Decomposing $g_0(x)$ into shorter factors would reduce the latency of the syndrome computation in the sub-codeword decoding. On the other hand, the worst-case latency of the GII-BCH decoding is dominated by the nested decoding process, which is carried out for up to $v = 3$ rounds. To reduce the latency of the nested syndrome computation as well as the area of the overall GII-BCH encoder, BCH encoder 0 through 2 are implemented by decomposing the corresponding generator polynomials into two parts as $g_3(x) = g_0(x)g'_3(x)$, $g_2(x) = g_0(x)g'_2(x)$, and $g_1(x) = g_0(x)g'_1(x)$, respectively, and encoder 3 through 7 are implemented by non-decomposed single parallel LFSRs according to $g_0(x)$. In this case, the remainders from dividing the received sub-codewords, $y_i(x)$ $(0 \le i < m)$, by $g_0(x)$ are used to compute the syndromes in the sub-codeword decoding. The remainders of the nested received codewords, $\tilde{y}_i(x)$ $(0 \le i < v)$, divided by $g'_i(x)$ are utilized to calculate the syndromes in each of the nested decoding rounds.

The complexity of our two proposed resource-shareable BCH encoders are analyzed and the results are shown in Table I. Decomposing the generator polynomial $g_i(x)$ $(1 \le i \le v)$ into two parts, the first proposed parallel BCH encoder shown in Fig. 6 only consists of two LFSR blocks, one $(\mathbf{C}_L)_0^{(1)}$ matrix multiplication of dimension $\deg(g'_i(x)) \times w_0$, and one $\mathbf{V}_T$ matrix multiplication of dimension $L \times w_i$. Each LFSR block is implemented by the parallel architecture in Fig. 7. To implement a matrix multiplication, the number of XOR gates needed is estimated as the sum of the numbers of '1's in each row minus one. A 2-to-1 multiplexer has the same area requirement as an XOR gate. The areas of a 2-input AND gate and a register are estimated as 1/2 and 3 times, respectively, the area of an XOR gate. The overall complexity in Table I is derived using these assumptions. The encoders with longer generator polynomials have more gates in the critical path since their $\mathbf{V}_T$ matrices are larger.

The architecture of the first stage of the second proposed reformulated parallel resource-shareable BCH encoder shown in Fig. 9 also has two LFSRs and one $(\mathbf{C}_L)_0^{(1)}$ matrix multiplication. However, it does not have the $\mathbf{V}_T$ multiplication. Let $\delta_0^{(i)}$ and $\delta_1^{(i)}$ be numbers of input taps shifted in LFSR 0 and 1, respectively, in BCH encoder i $(0 \le i < v)$. For BCH encoder 0, $\deg(g'_3(x)) = 690 - 336 = 354 > L$. Hence, $\delta_1^{(0)}$ is set to L. Besides, $\deg(g_0(x)) = 336$, which is also larger than L. Hence, $\delta_0^{(0)}$ is also set to L. Since $\delta_0^{(0)} + \delta_1^{(0)} = 2L$, $2L$ zeros are padded after the least significant coefficient of the data input polynomial. For BCH encoder 1 and 2, the same number of zeros are padded to simplify the alignment. $\deg(g'_2(x)) = 462-336 = 126$ and $\deg(g'_1(x)) = 384-336 = 48$. Both of them are smaller than L. Hence, $\delta_1^{(1)} = 126$ and $\delta_1^{(2)} = 48$. Since $\delta_0^{(i)} + \delta_1^{(i)}$ should be $2L$. $\delta_0^{(1)}$ and $\delta_0^{(2)}$ are set to $2L - 126 = 194$ and $2L - 48 = 272$, respectively. Stage 2 of the second proposed encoder with generator polynomial $g_i(x)$ computes $p(x) = \hat{p}_1(x)g_0(x) + \hat{p}_0(x)x^{\deg(g'_i(x))-\delta_1^{(v-i)}}$. $\hat{p}_0(x)$ and $\hat{p}_1(x)$ are the remainders of divisions by $g_0(x)$ and $g'_i(x)$, respectively, and hence $\deg(\hat{p}_0(x)) \le \deg(g_0(x)) - 1$ and $\deg(\hat{p}_1(x)) \le \deg(g'_i(x)) - 1$. If $\deg(g'_i(x)) \le L$, the L-parallel version of the FIR filter architecture for implementing the multiplication of $\hat{p}_1(x)g_0(x)$ reduces to a $w_i \times \deg(g'_i(x))$ constant matrix multiplication. Since $\deg(g'_1(x)) = 48$ is small, the number of non-zero entries in the corresponding constant matrix multiplication of stage 2 is much smaller than that of the $\mathbf{V}_T$ matrix. Therefore, the area of BCH encoder 2 with generator polynomial $g_1(x)$ using the second design is much smaller than that using the first proposed design. On the other hand, the complexity of the stage-2 polynomial multiplication increases at a faster pace with $\deg(g'_i(x))$ compared to that of $\mathbf{V}_T$ matrix multiplication. Hence, the area saving of our second design over the first design is less significant for BCH encoder 1 with generator polynomial $g_2(x)$. As $\deg(g'_i(x))$ further increases, the multiplication with $(\mathbf{C}_L)_0^{(1)}$ contributes to a larger percentage of the overall encoder complexity. Since our second design substantially simplifies $(\mathbf{C}_L)_0^{(1)}$ for larger $\deg(g'_i(x))$, it can achieve substantial area reduction compared to the first proposed design for BCH encoder 0 with generator polynomial $g_3(x)$. As analyzed previously, the second proposed encoder architecture also has shorter critical path due to the elimination of the $\mathbf{V}_T$ matrix multiplication.

Parallel resource-shareable BCH encoder with generator

TABLE II
TOTAL AREAS AND CRITICAL PATHS OF 160-PARALLEL RESOURCE-SHAREABLE GII-BCH ([8,3], 4095) ENCODERS

	[18]	[19] unfolded	Prop. enc. 1	Prop. enc. 2
Alignment	23277	24477	24477	27759
BCH encoders 0-2	133323	135088	247519	215991
BCH encoders 3-7	144840	144840	144840	144840
Pipelining reg.	5760	0	5760	5760
total area (# of XORs)	307200 (0.73)	304405 (0.72)	422596 (1.00)	394350 (0.93)
critical path (# of gates)	8	800	11	9
encoding latency (# of clks)	30	29	30	31
Rem. length for nest. syn. comp.	690	354	354	354

polynomial decomposition does not exist previously. The best alternative design for reducing the decoding complexity is to use the remainder polynomial from the division by the original generator polynomial, which is implementable by a traditional BCH encoder, to compute the syndromes. Among existing parallel BCH encoders, the one from [18] has the lowest complexity and short critical path. Hence, this design is compared to the proposed designs in Table I. In this table, the areas of different designs are normalized to that of the first proposed encoder. Although the BCH encoder architecture in [18] has smaller area and shorter critical path than the proposed designs, if it is used to construct a GII-BCH encoder, the maximum length of the remainder polynomials needed for the syndrome computation in both the sub-codeword and nested decoding of GII codes is $\max(w_0, w_1, w_2, w_3) = 690$. On the other hand, the proposed designs compute the nested syndromes by using the remainders of dividing each nested codeword $\tilde{y}_i(x)$ by $g'_i(x)$ and the longest remainder length is $\max(w_1 - w_0, w_2 - w_0, w_3 - w_0) = 354$. All remainder polynomials are computed simultaneously during the first-stage sub-codeword decoding of GII decoding. They can be stored and used directly in the second-stage nested decoding. Hence, the latency of the nested syndrome computation is determined by the maximum length of the involved remainder polynomials. As a result, the proposed designs reduce the latency of the nested decoding by around a half in terms of the number of clock cycles compared to the GII-BCH encoder using traditional BCH encoders from [18]. For completeness, the multi-mode encoder in [19] is unfolded with a factor of L and the complexity is estimated in Table I. A L-unfolded design has L times logic gates and the same number of register as the serial design. Besides, the iteration bound, which decides the shortest achievable critical path, is increased to L times of the iteration bound in the serial design [22].

Utilizing the proposed parallel resource-shareable BCH encoders, GII-BCH encoders can be developed and their complexities are listed in Table II. The parameters of the ABs can be determined following the explanations in Section III and the ABs with the same input signals can share registers as explained previously. Since the parameters of the ABs need to be increased accordingly to allow zero padding in the second proposed GII-BCH encoder design, its alignment scheme needs more gates to implement compared to that of the first proposed design. In the first proposed design and the one derived from unfolding, the inputs of the LFSRs are added to the MSTs. Hence, these designs require additional registers to delay the coefficients of $p_i^*(x)$ until those of $\mathcal{L}_{w_{v-i}}(f_i(x))$ are generated. Therefore, the complexities of their alignment schemes are larger than that of the design using the BCH encoder from [18]. The complexities of BCH encoder 0 through 2 from Table I are added up and listed in Table II. Encoder 3 through 7 do not employ any decomposition and are implemented by the architecture from [18].

TABLE III
SYNTHESIS RESULTS OF 160-PARALLEL RESOURCE-SHAREABLE GII-BCH ([8,3], 4095) ENCODERS USING TSMC $65nm$ PROCESS UNDER $T = 1ns$ TIMING CONSTRAINT

	[18]	Prop. enc. 1	Prop. enc. 2
area (μm^2)	596047	830033	715031
area ratio	0.72	1.00	0.86
enc. throughput (terabit/s)	1.09	1.09	1.06
nest. syn. comp. clk cycle ratio	1.95	1.00	1.00

The encoding latency of each design is also listed in Table II. Processing $n = 4095$ inputs with a parallelism of $L = 160$ needs $\lceil 4095/160 \rceil = 26$ clock cycles. However, the zero paddings needed to align the coefficients introduce another 3 clock cycles. Additionally, the longest data path that starts from the output of the BCH encoders goes through 3 ABs, 8 XOR gates, and 3 multiplexers before it reaches the output of the GII-BCH encoder. Two of the ABs have $\theta = 0$ and hence only have one multiplexer each in the data path. Therefore, this path has 15 gates. It is longer than those in the parallel BCH encoders and needs to be pipelined into two stages. Accordingly, the latency of the first proposed GII-BCH encoder is $26 + 3 + 1 = 30$ clock cycles. The second proposed design requires one more clock cycle due to the additional zero padding to the input polynomial for enabling input-tap modifications. Adding up the complexities of the alignment scheme, BCH encoders, and the pipelining registers, the total areas of the GII-BCH encoders are derived as shown in Table II. Comparing to the first proposed architecture, the area reduction achieved by our second design is less significant compared to the ratio shown in Table I since 5 out of the 8 BCH encoders with generator polynomial $g_0(x)$ are the same in both designs. Although the proposed encoders have larger areas compared to those using traditional BCH encoders, the latency required for the nested syndrome computation is reduced to around a half as analyzed previously.

To further evaluate the complexities of the proposed parallel resource-shareable GII-BCH encoders, they are synthesized using TSMC $65nm$ process under $T = 1ns$ timing constraint with the standard voltage threshold (SVT). The total area is listed in Table III. For comparison, the parallel GII-BCH encoder architecture using the traditional BCH encoder from [18] is also synthesized under the same timing constraint. Since the critical path of the unfolded version of [19] is overwhelmingly long, it is not further compared. The second proposed design achieves 14% area reduction compared to the first design. This area saving is more significant than that in Table II since the second design has shorter critical path. The

TABLE IV
MINIMUM ACHIEVABLE CLOCK PERIODS AND AREA REQUIREMENTS OF 160-PARALLEL RESOURCE-SHAREABLE GII-BCH ([8,3], 4095) ENCODERS USING TSMC $65nm$ PROCESS

	[18]	Prop. enc. 1	Prop. enc. 2
timing constraint (ns)	0.8	1.0	0.85
area (μm^2)	642267	830033	775545
area ratio	0.77	1.00	0.93
enc. throughput (terabit/s)	1.37	1.09	1.24
nest. syn. comp. clk cycle ratio	1.95	1.00	1.00

area ratios between the GII encoder using the BCH encoder from [18] and the first proposed encoder are similar in Table II and III despite the shorter critical path of the former encoder. This is because that the matrices involved in the proposed designs are smaller and substructure sharing can be performed more efficiently by the synthesis tool.

To evaluate the minimum achievable clock periods of the proposed designs, syntheses are carried out with different timing constraints. The tightest timing constraints that can be met without negative slacks are listed in Table IV together with the corresponding area. The second proposed encoder can achieve much shorter clock period than the first proposed design due to the shorter critical path. The area ratio between these two designs matches that listed in Table II from our analysis. Since the parallel GII-BCH encoder architecture using the traditional BCH encoder from [18] has much shorter critical path, it can meet an even tighter timing constraint. However, its area compared to that of the first proposed encoder has a higher ratio than that from Table II as the matrices involved in the proposed designs are much smaller and their implementation can be further optimized by the synthesis tool.

The generator polynomials can be decomposed into more factors using the proposed designs to further reduce the lengths of the remainders needed for syndrome computation. However, the $\widehat{\mathbf{A}}$ matrix will be divided into more blocks of submatrices, and thus both the total area and the critical path of the proposed GII-BCH resource-shareable encoders will increase very fast with L. For smaller L, the complexity of the polynomial multiplication in stage 2 of the second proposed design would be much higher than the complexity of the eliminated $\mathbf{V}_T$ multiplication. As a result, the first proposed encoder is a better choice for smaller L. For example, when $L = 30$, the numbers of XOR gates needed for the GII-BCH encoders constructed using the first and second proposed designs are estimated to be 91838 and 106237, respectively.

The proposed designs can be extended to GII-RS encoders. Since the $\pi^{(i)}(x)$ vectors are replaced by vectors consisting of constant finite field elements, the alignment issues become simpler for GII-RS encoding. Although the generator polynomials of RS codes are non-binary, similar techniques can still be utilized to reduce the complexities of the matrices involved in the parallel LFSRs for RS encoding. Our design can be also applied to the case where $\mathcal{C}_v, \cdots, C_1, \mathcal{C}_0$ have the same dimension but different lengths. In this case, the alignment scheme needs to be adjusted. However, the two proposed parallel resource-shareable BCH encoder architectures can be directly employed.

VII. CONCLUSIONS

For the first time, this paper considers the design of parallel resource-shareable GII-BCH encoders. Low-complexity schemes are developed to solve the coefficient alignment issues caused by the differences on the dimensions of the BCH codes involved in the GII code construction and the variations on the degrees of the polynomials multiplied to the subcodewords in the encoding. Additionally, two efficient parallel architectures are proposed to enable the sharing of GII-BCH encoders to reduce the complexity and latency of decoders. The proposed encoders can reduce the latency of the nested syndrome computation in the nested decoding by around a half with small area overhead compared to the best possible alternative design. Besides, the second encoder developed through algorithmic reformulations achieves substantial area reduction compared to the first one in the case of larger parallelisms. Future work will focus on further reducing the complexity of parallel resource-shareable GII encoders.

REFERENCES

[1] X. Tang and R. Koetter, "A novel method for combining algebraic decoding and iterative processing," *Proc. of IEEE Int. Symp. Info. Theory*, Seattle, WA, USA, 2006, pp. 474-478.

[2] Y. Wu, "Generalized integrated interleaved codes," *IEEE Trans. on Info. Theory*, vol. 63, no. 2, pp. 1102-1119, Feb. 2017.

[3] M. Blaum and S. R. Hetzler, "Extended product and integrated interleaved codes," *IEEE Trans. Inf. Theory*, vol. 64, no. 3, pp. 1497-1513, Mar. 2018.

[4] M. Blaum, "Extended integrated interleaved codes over any field with applications to locally recoverable codes," *IEEE Trans. Inf. Theory*, vol. 66, no. 2, pp. 936-956, Feb. 2020.

[5] W. Li, J. Lin and Z. Wang, "A 124-Gb/s decoder for generalized integrated interleaved codes," *IEEE Trans. on Circuits and Syst.-I*, vol. 66, no. 8, pp. 3174-3187, Aug. 2019.

[6] X. Zhang and Z. Xie, "Efficient architectures for generalized integrated interleaved decoder," *IEEE Trans. on Circuits and Syst.-I*, vol. 66, no. 10, pp. 4018-4031, Oct. 2019.

[7] Z. Xie and X. Zhang, "Reduced-complexity key equation solvers for generalized integrated interleaved BCH decoders," *IEEE Trans. on Circuits and Syst.-I*, vol. 67, no. 12, pp. 5520-5529, Dec. 2020.

[8] Z. Xie and X. Zhang, "Fast nested key equation solvers for generalized integrated interleaved decoder," *IEEE Trans. on Circuits and Syst-I*, vol. 68, no. 1, pp. 483-495, Jan. 2021.

[9] X. Zhang, "Systematic encoder of generalized three-layer integrated interleaved codes," *Proc. of IEEE Int. Conf. on Comm.*, Shanghai, China, 2019.

[10] W. Li, J. Tian, J. Lin and Z. Wang, "Modified GII-BCH codes for low-complexity and low-latency encoders," *IEEE Comm. Letters*, vol. 23, no. 5, pp. 785-788, May 2019.

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

[12] J. H. Derby, "High-speed CRC computation using state-space transformations," *Proc. of IEEE Global Telecom. Conf.*, San Antonio, TX, USA, 2001, pp. 166-170.

[13] C. Kennedy and A. Reyhani-Masoleh, "High-speed CRC computations using improved state-space transformations," *Proc. of IEEE Intl. Conf. Electro/Info. Tech.*, Windsor, ON, Canada, 2009, pp. 9-14.

[14] M. Ayinala and K. K. Parhi, "High-speed parallel architectures for linear feedback shift registers," *IEEE Trans. on Signal Process.*, vol. 59, no. 9, pp. 4459-4469, Sept. 2011.

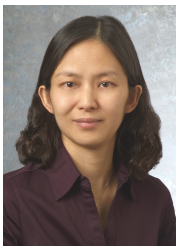
[15] G. Hu, J. Sha, and Z. Wang, "High-speed parallel LFSR architectures based on improved state-space transformations," *IEEE Trans. on VLSI Syst.*, vol. 25, no. 3, pp. 1159-1163, Mar. 2017.

[16] X. Zhang, "A low-power parallel architecture for linear feedback shift registers," *IEEE Trans. on Circuits and Syst.-II*, vol. 66, no. 3, pp. 412-416, Mar. 2019.

[17] X. Zhang, and Y. J. Tang, "Low-complexity parallel cyclic redundancy check," *Proc. of IEEE Int. Symp. on Circuits and Syst.*, Daegu, Korea, 2021.

[18] X. Zhang, “High-speed and low-complexity parallel long BCH encoder,” *Proc. of IEEE Int. Symp. on Circuits and Syst.*, Seville, Spain, 2020.
[19] H. Tang, G. Jung and J. Park, “A hybrid multimode BCH encoder architecture for area efficient re-encoding approach,” *Proc. of IEEE Int. Symp. on Circuits and Syst.*, Lisbon, Portugal, 2015, pp. 1997-2000.
[20] H. Yoo, J. Jung, J. Jo and I. Park, “Area-efficient multimode encoding architecture for long BCH codes,” *IEEE Trans. on Circuits and Syst.-II*, vol. 60, no. 12, pp. 872-876, Dec. 2013.
[21] A. K. Subbiah and T. Ogunfunmi, “Area-effcient re-encoding scheme for NAND flash memory with multimode BCH error correction,” *Proc. of IEEE Int. Symp. on Circuits and Syst.*, Florence, Italy, 2018.
[22] K. K Parhi, *VLSI Digital Signal Processing Systems: Design and Implementation*, John Wiley & Sons, 1999.
[23] S. B. Wicker, *Error Control Systems for Digital Communication and Storage*. Upper Saddle River, NJ, USA: Prentice-Hall, 1995.

Yok Jye Tang received his B.S. degree in electrical and computer engineering from The Ohio State University, Columbus OH, USA, in 2019. He is currently pursuing his Ph.D degree in electrical engineering at The Ohio State University. His current research interests focus on the area of high-performance very-large-scale-integration (VLSI) architectures for erasure codes.



Xinmiao Zhang received her Ph.D. degree in Electrical Engineering from the University of Minnesota. She joined The Ohio State University as an Associate Professor in 2017. Prior to that, she was a Timothy E. and Allison L. Schroeder Assistant Professor 2005-2010 and Associate Professor 2010-2013 at Case Western Reserve University. Between her academic positions, she was a Senior Technologist at Western Digital/SanDisk Corporation. Dr. Zhang’s research spans the areas of VLSI architecture design, digital storage and communications, security, and signal processing.

Dr. Zhang received an NSF CAREER Award in January 2009. She is also the recipient of the Best Paper Award at 2004 ACM Great Lakes Symposium on VLSI and 2016 International SanDisk Technology Conference. She authored the book “VLSI Architectures for Modern Error-Correcting Codes” (CRC Press, 2015), and co-edited “Wireless Security and Cryptography: Specifications and Implementations" (CRC Press, 2007). She was elected to serve on the Board of Governors of the IEEE Circuits and Systems Society for the 2019-2021 term. She is the Chair (2021-2022) and a Vice-Chair (2017-2020) of the Data Storage Technical Committee, and a member of the CASCOM and VSA Technical Committees of IEEE. She served on the technical program and organization committees of many conferences, including ISCAS, SiPS, ICC, GLOBECOM, GlobalSIP, and GLSVLSI. She has been an associate editor for the IEEE Transactions on Circuits and Systems-I 2010-2019 and IEEE Open Journal of Circuits and Systems since 2019.