Fast and Low-Complexity Soft-Decision Generalized Integrated Interleaved Decoder

Yok Jye Tang and Xinmiao Zhang

***Abstract*—Generalized integrated interleaved (GII) codes are essential to next-generation digital communication and storage systems since they can achieve very high decoding throughput with low complexity. Only hard-decision GII decoding has been considered in previous work. To further improve the error-correcting capability, soft-decision decoding algorithms utilizing the channel probability information need to be developed. The decoding of GII codes constructed based on BCH codes consists of multiple rounds of BCH decoding. Among existing soft-decision decoding algorithms of BCH codes, the Chase algorithm that carries out decoding trials on multiple test vectors can achieve a better trade-off on the coding gain and complexity. Although one-pass Chase algorithms can derive the error-locator polynomials for all the test vectors in one run, the exhaustive Chien search is carried out previously on each error-locator polynomial to decide which one is correct and it leads to long latency. For the first time, this paper proposes an efficient soft-decision GII decoding algorithm. Different methods of incorporating the Chase process into the GII scheme are analyzed and compared to identify the best GII Chase decoding algorithm. Besides, a new error-locator polynomial selection scheme is developed to avoid carrying out the Chien search on each error-locator polynomial by pre-flipping a bit in the received word. Accordingly, the error-locator polynomial can be selected by testing whether it consists of a pre-determined factor. The latency is further reduced by pre-computing short remainder polynomials in our second proposed scheme. In addition, formulas have been developed to estimate the error-correcting performance of the proposed designs. This paper also develops low-complexity hardware architectures to implement the proposed GII-BCH Chase decoders. For an example GII-BCH code with 8 sub-codewords of 4095 bits over $GF(2^{12})$, the proposed GII-BCH Chase decoder can achieve significant coding gain over hard-decision decoder with negligible silicon area overhead. Besides, our proposed designs can reduce the worst-case latency of GII-BCH Chase nested decoding rounds by 54%-80%.**

***Index Terms*—BCH codes, Chase decoding, error-correcting codes, generalized integrated interleaved codes, hardware architecture, soft-decision decoding.**

I. Introduction

For next-generation digital communications and storage, error-correcting codes that can achieve hyper throughput with high coding gain are needed. Generalized integrated interleaved (GII) codes [1], [2] are one of the best candidates that meet such requirements. A $[m, v]$ GII code consists of m sub-codewords. Each sub-codeword can be a Reed-Solomon (RS) or BCH codeword. Their nestings form v codewords of stronger RS or BCH codes. Most of the time, the decoding is carried out on each sub-word separately and it can easily achieve hundreds of megabit/s throughput [3], [4] with low complexity. Besides, the nested codewords can be utilized to correct more errors. As a result, GII codes achieve orders of magnitude lower decoding failure rates compared to individual RS or BCH codes. GII decoding consists of two stages. The first stage is the traditional RS/BCH decoding of individual sub-words. When any of the sub-words is not corrected in the first stage, the second-stage nested decoding consisting of multiple rounds of more powerful RS/BCH decoding is activated. GII decoder hardware implementation architectures are available in [3]–[6].

Previous GII schemes [1], [2] only consider hard-decision RS/BCH decoding based on the Berlekamp's algorithm [7]. Soft-decision GII decoding algorithms that utilize the channel probability information can improve the error-correcting performance. The Chase algorithm [8] flips the η least reliable bits of a RS/BCH word and carries out 2^η decoding trials. It achieves a better trade-off on error-correcting performance and decoding complexity compared to other soft-decision decoding algorithms of RS/BCH codes [9]–[14]. To avoid restarting the decoding for each test vector, one-pass Chase algorithms [15]–[17] have been developed to derive the error-locator polynomials of all the test vectors in one run. However, the exhaustive Chien search is carried out on each error-locator polynomial until finding one whose degree equals the root number. The Chien search can not be shared among the test vectors and has long latency. In the GII decoding scenario, the Chase decoding can be activated multiple times over the decoding rounds and the long latency caused by the Chien search is an even more significant issue to address. To avoid implementing the Chien search for each test vector, different schemes [18], [19] have been proposed to select the correct error-locator polynomial. However, the scheme in [18] still has high complexity and it requires a large number of clock cycles to identify the correct error-locator polynomial, especially for codes with higher error-correcting capabilities. Although the polynomial selection in [19] has low complexity and short latency, it is only applicable to interpolation-based RS/BCH decoding. Since GII decoding can not be carried out using the interpolation-based scheme, the polynomial selection in [19] can not be applied to the Chase process in the GII decoding.

For the first time, this paper considers soft-decision GII decoding. Different methods of integrating the Chase process into the GII decoding are analyzed and compared in this paper to identify the best GII Chase decoding algorithm. Besides, a new polynomial selection scheme for the Chase decoding

Yok Jye Tang and Xinmiao Zhang are with the Department of Electrical and Computer Engineering, The Ohio State University Columbus, OH 43210 U.S.A. E-mails: {tang.1121, zhang.8952}@osu.edu

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

is developed by pre-flipping a bit in the received sub-word. Accordingly, the error-locator polynomial can be selected based on whether it has a pre-determined factor. For codes over $GF(2^q)$, the error-locator polynomial of a correctable test vector is selected with a probability of $1-2^{-q}$ using such a scheme. Hence, for codes over higher-order finite fields, the Chien search only needs to be carried out once in most cases. To further speed up the polynomial selection process, the remainders of dividing each error-locator polynomial by the pre-determined factor can be computed without deriving the polynomial itself. Since the remainders are very short, multiple of them can be calculated in parallel with low complexity. In addition, formulas for estimating the error-correcting performance of the proposed designs are given. This paper also develops low-complexity hardware architectures to implement our proposed GII-BCH Chase decoders. For an example GII code that consists of 8 sub-codewords with 4095 bits over $GF(2^{12})$, our proposed GII-BCH Chase decoder has much better error-correcting performance and negligible silicon area overhead compared to a hard-decision GII-BCH decoder. Besides, our proposed polynomial selection schemes can reduce the worst-case latency by 54%-80% in each nested decoding round.

The structure of this paper is as follows. Background information is introduced in Section II. Different GII Chase decoding schemes are proposed, analyzed, and compared in Section III. Section IV presents our proposed pre-bit-flipping polynomial selection scheme for Chase decoding. The method of pre-computing the remainder polynomials for our polynomial selection scheme is proposed in Section V. Section VI presents the formulas for estimating the error-correcting performance of the proposed designs. Low-complexity hardware architectures for implementing the proposed GII-BCH Chase decoders are provided in Section VII. Complexity analyses and comparisons are given in Section VIII and conclusions follow in Section IX.

II. GII-BCH Chase decoding

A $([m,v],n)$ GII codeword is divided into m sub-codewords, $c_0(x), c_1(x), \cdots, c_{m-1}(x)$, each of which is a codeword of $\mathcal{C}_0(n,k_0)$ Reed-Solomon (RS) or BCH code with length n and dimension k_0. Besides, linear combinations of the m sub-codewords produce v codewords of stronger RS or BCH codes $\mathcal{C}_1(n,k_1), \cdots, \mathcal{C}_v(n,k_v)$ $(k_0 > k_1 \geq k_2 \geq \cdots \geq k_v)$ [1], [2] as follows

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

Let β be an n-th root of unity in $GF(2^q)$. For GII-RS codes, $h_{l,i}(x)$ is the constant finite field element β^{li}. For GII-BCH codes, $h_{l,i}(x)$ is the standard basis polynomial representation of β^{li}. All the $h_{l,i}(x)$ form the entries of the nesting matrix

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

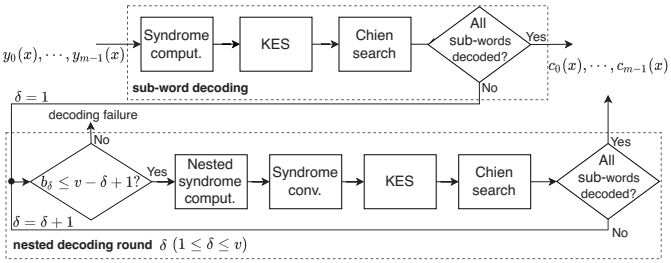


Fig. 1: Data flow of $[m,v]$ GII decoding.

For systematic GII codeword, $c_i(x)$ for $i = 0, 1, \cdots, v-1$ has k_{v-i} data symbols and each of $c_v(x), c_{v+1}(x), \cdots, c_{m-1}(x)$ has k_0 data symbols [2]. The rate of such a GII code is $R = (k_0 \cdot (m-v) + k_1 + \cdots + k_v)/(m \cdot n)$. Alternatively, GII codes can be defined by having the same number of data symbols but different lengths in the sub-codewords.

The decoding procedure of GII codes is summarized in Fig. 1. A sub-codeword received by the decoder may contain errors and is referred to as a 'sub-word'. Let $y_0(x), y_1(x), \cdots, y_{m-1}(x)$ be the m received sub-words and the error-correcting capabilities of $\mathcal{C}_v, \cdots, \mathcal{C}_1, \mathcal{C}_0$ be $\tau_v \geq \cdots \geq \tau_1 > \tau_0$, respectively. There are two stages in GII decoding [2]. First, the traditional RS/BCH decoding is carried out on individual sub-words. Consider narrow sense RS/BCH codes. Syndromes of individual sub-words are computed as $S_j^{(i)} = y_i(\beta^{j+1})(0 \leq j < 2\tau_0, 0 \leq i < m)$. If all $2\tau_0$ syndromes of $y_i(x)$ are zero, it has no error. Otherwise, the Berlekamp-Massey (BM) key-equation solver (KES) algorithm [7] is carried out to compute the error-locator polynomial $\Lambda(x)$. Then the error locations, which are the inverse roots of $\Lambda(x)$, can be found by using the exhaustive Chien search that evaluates $\Lambda(x)$ over n non-zero finite field elements of $GF(2^q)$, $\beta^0, \beta^1, \cdots, \beta^{n-1}$. If the root number of $\Lambda(x)$ is the same as its degree, the decoding is considered successful. Such decoding can correct up to τ_0 errors in each $y_i(x)$.

The second-stage nested decoding is only necessary when there are at least one sub-word with more than τ_0 errors. It is carried out for up to v rounds. Assume that $y_{i_0}(x), y_{i_1}(x), \cdots, y_{i_{b_{\delta-1}}}(x)$ remain to be corrected in the beginning of round δ $(1 \leq \delta \leq v, b_\delta \leq v - \delta + 1)$. Let $I = \{i_0, i_1, \cdots, i_{b_\delta - 1}\}$ be the set of the indices of the b_δ sub-words with extra errors and I^C be the set of the indices of the other sub-words that have been corrected in previous decoding rounds. Higher-order syndromes of the first b_δ nested words are computed as $\tilde{S}_j^{(l)} = \tilde{y}_l(\beta^{j+1})$ $(0 \leq l < b_\delta, 2\tau_{\delta-1} \leq j \leq 2\tau_\delta - 1)$, where $\tilde{y}_l(x) = \sum_{i \in I} \beta^{il} y_i(x) + \sum_{i \in I^C} \beta^{il} c_i(x)$. Then higher-order syndromes for the b_δ sub-words are derived as $[S_j^{(i_0)}, S_j^{(i_1)}, \cdots, S_j^{(i_{b_\delta - 1})}] = A^{-1}[\tilde{S}_j^{(0)}, \tilde{S}_j^{(1)}, \cdots, \tilde{S}_j^{(b_\delta - 1)}]^T$, where '$T$' denotes transpose and A consists of the entries in the first b_δ rows and columns $i_0, i_1, \cdots, i_{b_\delta - 1}$ of the nesting matrix H. After that, the KES computes the error-locator polynomial according to the $2\tau_\delta$ syndromes of each sub-word followed by the Chien search. Each nested decoding round can correct up to τ_δ errors in each of the $b_\delta \leq v - \delta + 1$ sub-words and the uncorrectable ones are passed to the next

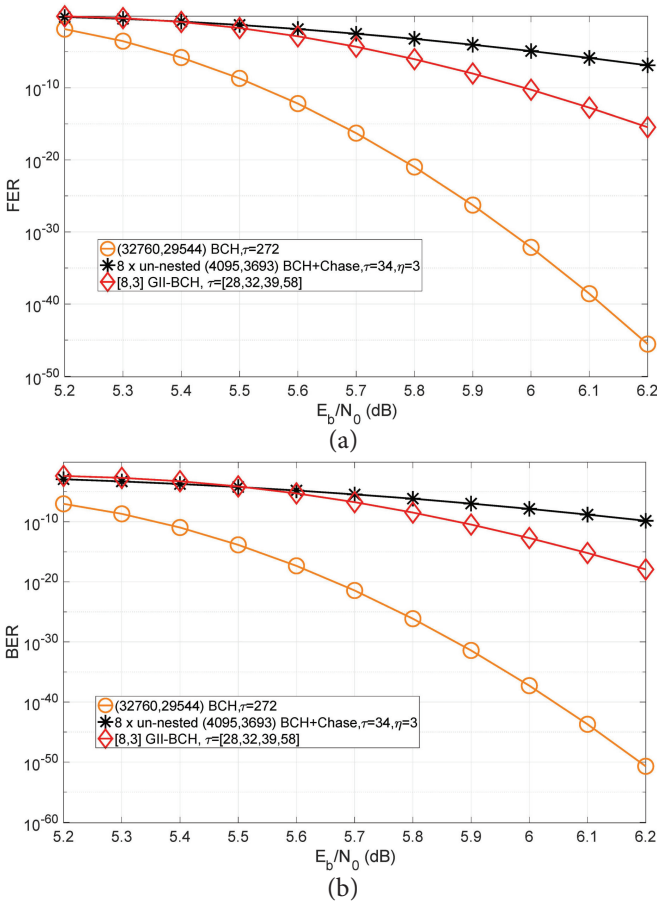


Fig. 2: Error-correcting performances of a long $(32760, 29544)$ BCH code, eight un-nested $(4095, 3693)$ BCH codes and a $([8, 3], 4095)$ GII-BCH code with $\tau = [\tau_0, \tau_1, \tau_2, \tau_3] = [28, 32, 39, 58]$ over AWGN channel: a) FER; b) BER.

nested decoding round. If $b_\delta > v - \delta + 1$, decoding failure is declared. GII decoder architectures have been developed in [3]–[6]. A systematic encoding algorithm of GII codes can be found in [2] and GII encoder architectures have been explored in [20]–[22].

Due to the sub-codeword and nested codeword structure, GII-BCH codes can achieve hyper-speed decoding with good correction capability and are among the best candidates for error correction in next-generation memories. Fig. 2 plots the decoding frame error rate (FER) and bit error rate (BER) of a $([8, 3], 4095)$ GII-BCH code with 90.2% code rate over the additive white gaussian noise (AWGN) channel. It was found in [4], [6] that using $\tau = [\tau_0, \tau_1, \tau_2, \tau_3] = [28, 32, 39, 58]$ achieves a better trade-off between error-correcting performance and decoding latency compared to other GII-BCH codes with the same m, v, n and code rate but different τ. For comparisons, the FER and BER of a single long $(32760, 29544)$ BCH code with the same code rate are also plotted in Fig. 2. This code has $\tau = 272$ and achieves lower FER and BER. However, it has overwhelming hardware complexity and very long decoding latency that is unacceptable for practical systems.

Hard-decision decoders first make the decisions on whether each received bit is '1' or '0' before the decoding is carried out. Hard-decision BCH decoding using the BM algorithm [7] can correct up to $\tau = \lfloor (\gamma - 1)/2 \rfloor$ errors, where γ is the designed minimum distance of the code. More errors are correctable by soft-decision decoding through incorporating the probability information from the channel. Among existing soft-decision BCH decoding algorithms, the Chase algorithm [8] achieves a better trade-off on error-correcting performance and hardware complexity. It identifies the η least reliable bits and carries out trial decoding over 2^η test vectors derived by flipping those unreliable bits. As a result, up to $\tau + \eta$ errors are correctable. To keep the complexity low, small η needs to be used. On the other hand, for codes with larger τ, flipping a small number of bits leads to small improvement. Fig. 2 shows the error-correcting performance of eight un-nested $(4095, 3693)$ BCH codes with $\tau = 34$ and $\eta = 3$. Such un-nested codes have the same code rate as the $[8, 3]$ GII-BCH code. Compared to the hard-decision GII-BCH decoding, the FERs and BERs of the soft-decision BCH decoding are still higher.

Algorithm 1 BCH decoding with one-pass Chase algorithm

Input: $\alpha_0, \cdots, \alpha_{\eta-1}$; received word $y(x)$
Hard-decision decoding:
1. Compute syndromes $S_j = y(\beta^{j+1})(0 \le j < 2\tau)$
2. Calculate $\Lambda(x)$ and $\mathcal{B}(x)$ using the BM's algorithm
If $\deg(\Lambda(x))$ is equal to the number of roots of $\Lambda(x)$:
3. **Output:** $\Lambda(x)$
else:
One-pass Chase decoding:
4. $\sigma_0 = [0, 0, \cdots, 0]$ (vector with no bit flipped)
5. $\Lambda^{(0)}(x) = \Lambda(x)$, $\mathcal{B}^{(0)}(x) = x^2\mathcal{B}(x)$
6. $L_\Lambda^{(0)} = L_\Lambda$, $L_B^{(0)} = L_B$
for $(i = 1; i < 2^\eta; i++)$:
7. Find $\sigma_j (j < i)$ such that σ_i equals σ_j with an extra '1' in the l-th bit
8. Compute $a_l^{(j)} = \Lambda^{(j)}(\alpha_l^{-1})$, $b_l^{(j)} = \mathcal{B}^{(j)}(\alpha_l^{-1})$
9. **Case 1:** $a_l^{(j)} = 0 \mid a_l^{(j)} \neq 0$ & $b_l^{(j)} \neq 0$ & $L_\Lambda^{(j)} \ge L_\mathcal{B}^{(j)}$
10. $\Lambda^{(i)}(x) = b_l^{(j)}\Lambda^{(j)}(x) + a_l^{(j)}\mathcal{B}^{(j)}(x)$
11. $\mathcal{B}^{(i)}(x) = (x^2 - \alpha_l^{-2})\mathcal{B}^{(j)}(x)$
12. $L_\Lambda^{(i)} = L_\Lambda^{(j)}$, $L_\mathcal{B}^{(i)} = L_\mathcal{B}^{(j)} + 2$
13. **Case 2:** $b_l^{(j)} = 0 \mid a_l^{(j)} \neq 0 \& b_l^{(j)} \neq 0 \& L_\Lambda^{(j)} < L_\mathcal{B}^{(j)} - 1$
14. $\Lambda^{(i)}(x) = (x^2 - \alpha_l^{-2})\Lambda^{(j)}(x)$
15. $\mathcal{B}^{(i)}(x) = b_l^{(j)}x^2\Lambda^{(j)}(x) + \alpha_l^{-2}a_l^{(j)}\mathcal{B}^{(j)}(x)$
16. $L_\Lambda^{(i)} = L_\Lambda^{(j)} + 2$, $L_\mathcal{B}^{(i)} = L_\mathcal{B}^{(j)}$
17. **Case 3:** $a_l^{(j)} \neq 0$ & $b_l^{(j)} \neq 0$ & $L_\mathcal{B}^{(j)} = L_\Lambda^{(j)} - 1$
18. $\Lambda^{(i)}(x) = b_l^{(j)}\Lambda^{(j)}(x) + a_l^{(j)}\mathcal{B}^{(j)}(x)$
19. $\mathcal{B}^{(i)}(x) = b_l^{(j)}x^2\Lambda^{(j)}(x) + \alpha_l^{-2}a_l^{(j)}\mathcal{B}^{(j)}(x)$
20. $L_\Lambda^{(i)} = L_\Lambda^{(j)} + 1$, $L_\mathcal{B}^{(i)} = L_\mathcal{B}^{(j)} + 1$
21. Stop if $\deg(\Lambda^{(i)}(x))$ equals the number of roots of $\Lambda^{(i)}(x)$
end for
Output: $\Lambda^{(i)}(x)$

In the original Chase algorithm [8], the hard-decision decoding is repeatedly carried out on each test vector. To reduce the decoding latency, error-locator polynomials for each test vector can be derived by using the one-pass Chase algorithms [15], [16], and the one in [16] has lower complexity. Let $\alpha_i (0 \le i < \eta)$ be the locations of the bits to flip in the Chase

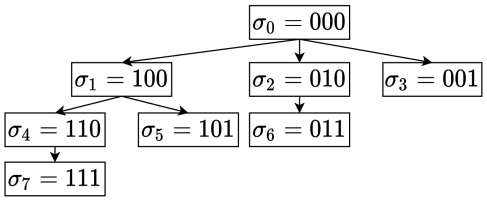


Fig. 3: Example order of bit flipping for one-pass Chase algorithm with $\eta = 3$ in [16].

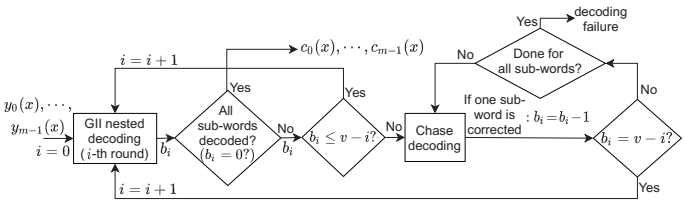


Fig. 4: $[m, v]$ GII Chase decoding scheme.

decoding and σ_j be the binary vector representing whether the η bits are flipped in the j-th test vector. For BCH codes, the one-pass Chase algorithm in [16] is listed in Algorithm 1. In this algorithm, $\mathcal{B}(x)$ is a polynomial associated with $\Lambda(x)$. The lengths of $\Lambda(x)$ and $\mathcal{B}(x)$ are L_Λ and $L_\mathcal{B}$, respectively.

In Algorithm 1, Lines 1-3 are the conventional BCH decoding on the hard-decision received word with no bit flipped. If the computed error-locator polynomial, $\Lambda(x)$, has a degree that is equal to its root number, it is considered as the correct error-locator polynomial and the locations of the errors in the received word are the inverse roots of $\Lambda(x)$. Otherwise, decoding trial is carried out on the other test vectors according to Lines 4-21. From [16], the $\Lambda^{(i)}(x)$ and $\mathcal{B}^{(i)}(x)$ of a test vector can be derived from those of another vector that has one less flipped bit. Fig. 3 shows an example order of bit flipping in the one-pass Chase scheme with $\eta = 3$. In this figure, the hard-decision received word with no bit flipped is denoted by vector $\sigma_0 = [000]$. If a bit is flipped, it is denoted by a '1' in the vector. Depending on the evaluation values $a_l^{(j)}$ and $b_l^{(j)}$ and the polynomial lengths, the error-locator and its associated polynomials are derived by using the formulas in one of the three cases. The process stops when the degree of $\Lambda^{(i)}(x)$ equals to its root number. Besides, the one-pass Chase decoding in Algorithm 1 does not allow a bit to be flipped back in later test vectors, as shown in Fig. 3. At the cost of extra logic complexity, the design in [17] divides both $\Lambda^{(i)}(x)$ and $\mathcal{B}^{(i)}(x)$ by $x - \alpha_l^{-1}$ at the end for each test vector such that a bit can be flipped back in the next test vector and only one intermediate result needs to be stored at any time.

III. Soft-decision GII decoder

Only hard-decision GII decoding has been considered previously. To further improve the error-correcting performance of GII codes, this section proposes an efficient soft-decision decoding scheme. Similar to BCH Chase decoding, bit flipping can be incorporated into GII-BCH decoding. However, the bit flipping can be integrated in various ways due to the multi-round nested decoding of different error-correcting capabilities and limited number of sub-words that can be corrected in each nested decoding round. Potential methods of incorporating bit flipping into GII decoding are analyzed and compared in this section to identify the best scheme.

If a sub-word has been corrected, further flipping its bits would not correct more errors. Hence, bit flipping should only be applied to the uncorrected sub-words. This can be done in two different ways:

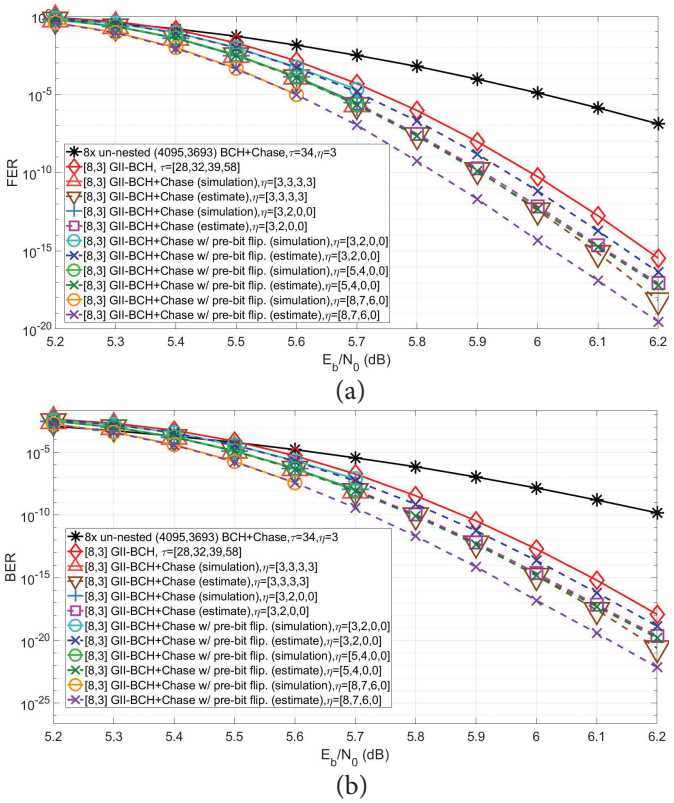


Fig. 5: Error-correcting performance of the proposed soft-decision $([8, 3], 4096)$ GII-BCH Chase decoding with different $\eta = [\eta_0, \eta_1, \eta_2, \eta_3]$: a) FER; b) BER.

1) Only flip the unreliable bits in the erroneous sub-words when GII decoding is unable to continue to the next nested decoding round.
2) Try to flip the unreliable bits in the erroneous sub-words and correct as many sub-words as possible before moving to the next GII nested decoding round.

These two methods are different in when and for which sub-words the Chase decoding is activated. Considering the sub-word decoding as nested decoding round 0, the GII decoding has $v + 1$ rounds. Let b_i $(b_i > v - i)$ be the number of sub-words that remains uncorrected at the end of the i-th $(0 \le i \le v)$ GII decoding round. The first method carries out the Chase decoding on the sub-words one at a time until b_i is reduced to $v - i$, after which the decoding moves to round $i + 1$. If b_i is still larger than $v - i$ after applying the Chase decoding on every sub-word, decoding failure is declared. This GII Chase decoding flow is illustrated in Fig. 4. Unlike the first method, the second scheme activates the Chase decoding whenever there are uncorrected sub-words.

The Chase decoding is applied to each of them before moving to the next GII decoding round.

If η_i bits are flipped in a sub-word during nested decoding round i, up to $\tau_i+\eta_i$ errors can be corrected. If $\tau_i+\eta_i > \tau_{i+1}$, the nested decoding round $i+1$ will not correct additional errors and becomes meaningless. Besides, η_i needs to be small to reduce the complexity. Hence it should be set to a value smaller than $\tau_{i+1}-\tau_i$. The second method corrects additional sub-words before nested decoding round $i+1$ compared to the first approach. However, all those sub-words have up to $\tau_i+\eta_i \leq \tau_{i+1}$ errors. If they are sent to nested decoding round $i+1$ as in the first method, they will also get corrected. On the other hand, the number of sub-words sent to the next nested decoding round by the second method is not larger than that sent by the first scheme. Also the numbers of errors in those sub-words are not increased. As a result, if the errors are correctable by the first decoding scheme, they can be also corrected by the second method. In conclusion, the first and second methods have exactly the same error-correcting performance.

The second method has longer decoding latency compared to the first one because it carries out the Chase decoding on more sub-words. In the Chase decoding with η bits to flip, although the error-locator polynomials for all the 2^η test vectors can be derived in one run by using the one-pass algorithm, the exhaustive Chien search needs to be carried out on each of the polynomials to tell whether the errors are corrected. In the worst case, the long-latency Chien search needs to be carried out 2^η times and no intermediate result can be shared among them. On the other hand, only one Chien search is needed for each sub-word at the end of each GII nested decoding round. Considering that the first method achieves the same error-correcting performance as the second approach and has shorter decoding latency, it is chosen in our design.

The error-correcting performance of our proposed soft-decision GII Chase decoding scheme is affected by the values of η_i. Since $\tau_0 < \tau_1 \leq \cdots \leq \tau_v$, η_i for later rounds have less effect on the FERs and BERs. They can be set to smaller values to reduce the decoding complexity without affecting the overall error-correcting performance much. The FERs of the proposed soft-decision decoding scheme for the $([8,3],4095)$ GII-BCH code with different $\eta = [\eta_0, \eta_1, \eta_2, \eta_3]$ values are plotted in Fig. 5(a). It can be observed that our proposed scheme leads to much lower FERs compared to hard-decision GII decoding even with small η_i. Besides, setting the η_i's for later rounds to even smaller values, such as '0', leads to little performance loss. Similar observations can be made on the BER plot in Fig. 5(b).

IV. Pre-bit-flipping polynomial selection for one-pass Chase decoding

In the one-pass Chase decoding of [16], the exhaustive Chien search is carried out on the error-locator polynomial of each test vector until the one whose degree equals the root number is found. The Chien search requires a large number of clock cycles in the hardware implementation, and has much longer latency than the one-pass Chase KES. To address this issue, different schemes have been proposed to avoid using the Chien search for error-locator polynomial selection [18], [19], and the Chien search only needs to be applied to the selected polynomial to find the error locations. However, the design in [18] has high complexity and it needs long time to tell whether the error-locator polynomial is correct, especially for codes with higher error-correcting capabilities. Although the polynomial selection in [19] has low complexity and short latency, it is for RS/BCH codes with evaluation map encoding and interpolation-based decoding. The least significant message bit is pre-set to '1' so that the bivariate polynomial output from the interpolation process has a zero evaluation value over the pre-set point. This method is not applicable to the Chase process in GII decoding since the nested decoding process can not be carried out using the interpolation-based scheme.

Inspired by the design in [19], this paper proposes a new polynomial selection scheme for one-pass Chase decoding by setting a bit in the encoding process and pre-flipping the bit during the decoding. Without loss of generality, assume that the bit at position α_0 is set in the encoding. For GII Chase decoding, the bit at position α_0 can be flipped in two different ways. The first method flips the pre-set bit at the very beginning of the GII decoding before any Chase process is activated. If a test vector is correctable, its corresponding error-locator polynomial should have α_0^{-1} as a root. By setting $\alpha_0 = 1$, *i.e.* selecting the first bit of a sub-codeword as the pre-set bit, the evaluation value over α_0^{-1} is just the sum of the error-locator polynomial coefficients. If this evaluation value is zero, the error-locator polynomial is considered correct and the Chien search is carried out to find all the roots. Although this method is simple, one additional error is introduced to every sub-word and the hard-decision GII decoding in the i-th round can only correct up to $\tau_i - 1$ errors in each sub-word. As a result, the probabilities of activating the Chase process and nested decoding become higher and the decoding latency becomes longer.

The second approach is to flip the pre-set bit only when the Chase process is activated. This reduces the probabilities of activating the Chase and nested decoding compared to the first method. However, since the bit flipping is done in the beginning of the Chase decoding, the error-locator polynomial derived by using the one-pass Chase scheme in Algorithm 1 can not be selected as in the first method by calculating the evaluation value over α_0^{-1}. This is because, as proved in [16], if a test vector has a flipped bit at position α_i, the corresponding error-locator polynomial derived by using Algorithm 1 has a factor of $(x-\alpha_i^{-1})$. Hence, if the bit at α_0 is flipped in the beginning of the Chase process, the error-locator polynomial of every test vector has the $(x-\alpha_0^{-1})$ factor, regardless of whether it is correct or not. Let $\Lambda^*(x)$ denotes the error-locator polynomial of a correctable test vector computed by using Algorithm 1. It was shown in [17] that $\Lambda'(x) = \Lambda^*(x)/\prod_i(x-\alpha_i^{-1})$, where α_i include all the flipped bit positions, is an error-locator polynomial that consists of the inverses of all the error locations in the correctable test vector as distinct roots. Therefore, if α_0 is intentionally flipped to be

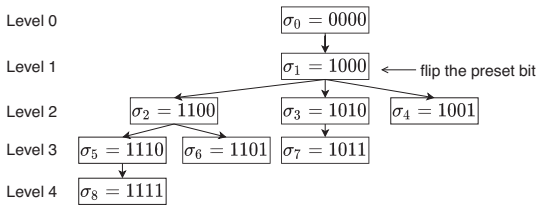


Fig. 6: Example order of bit flipping for the one-pass Chase that employs the proposed pre-bit-flipping polynomial selection scheme with $\eta = 3$.

wrong and the total number of errors in a test vector does not exceed τ for a τ-error-correcting BCH code, $\Lambda'(x)$ will have $x - \alpha_0^{-1}$ as a factor. Since $\Lambda^*(x) = \Lambda'(x) \times \prod_i (x - \alpha_i^{-1})$, $\Lambda^*(x)$ will have a factor of $(x - \alpha_0^{-1})^2$. As a result, the error-locator polynomial of a correctable test vector with the pre-set bit flipped derived by using Algorithm 1 can be selected based on whether it has $(x - \alpha_0^{-1})^2$ as a factor. Since this pre-determined factor has a degree of two, the correct error-locator polynomial has a root number that is equal to its degree minus one. The remainder computation of the division by $(x - \alpha_0^{-1})^2 = x^2 - 1$ can be done by simple additions. Let $r(x) = r_1 x + r_0$ be the remainder of dividing a polynomial $f(x) = f_0 + f_1 x + f_2 x^2 + f_3 x^3 + \cdots$ by $x^2 - 1$. It can be derived that $r_0 = f_0 + f_2 + \cdots$ and $r_1 = f_1 + f_3 + \cdots$.

Since the second approach has shorter latency, it is employed in our design. This pre-bit-flipping polynomial selection scheme is summarized as follows.

1) Pre-set the first bit of each sub-codeword in the encoding.
2) When the Chase process is activated in the GII Chase decoding, flip the pre-set bit in the sub-word and derive the corresponding error-locator polynomial by using the one-pass Chase KES Algorithm in Algorithm 1.
3) Flip the η least reliable bits in addition to the pre-set bit to generate test vectors. If the remainder of dividing an error-locator polynomial by $(x - \alpha_0^{-1})^2 = x^2 - 1$ is zero, find all the roots of this polynomial by using Chien search. If the root number is equal to the degree of the polynomial minus one, stop and declare decoding success. Otherwise, continue for the next test vector.

Let e denotes the number of errors in a sub-word before flipping any bit. In our design, the pre-set bit is flipped to be erroneous in the Chase process and the number of errors in each of the test vectors with the pre-set bit flipped is increased by one. For a τ-error-correcting code, the test vector with additional η bits flipped is correctable when $e + 1 \leq \tau + \eta$. Hence our design can correct up to $\tau + \eta - 1$ errors besides the flipped pre-set bit in the Chase process. Take $\eta = 3$ as an example. The proposed scheme tests the vectors in the order shown by the tree structure in Fig. 6.

The remainder of dividing an incorrect error-locator polynomial by $x^2 - 1$ may be zero accidentally. This happens with a probability of around 2^{-q} for codes over finite field $GF(2^q)$ since the error-locator polynomial of each test vector with the pre-set bit flipped is already divisible by $x - 1$. For codes over high-order fields, this probability is very small. Besides, if the root number turns out not equal the degree of the error-locator polynomial minus one from the Chien search, the wrong error-locator polynomial is still identified and the decoding trial on the other test vectors continues. Hence, even if the proposed scheme may select the wrong error-locator polynomial accidentally, it does not lead to any degradation on the error-correcting performance. Since the long-latency Chien search is only carried out on the selected polynomial, our scheme has much shorter latency than previous designs that carry out the Chien search on every error-locator polynomial. Miscorrections may also happen in GII Chase decoding. Low-complexity miscorrection mitigation schemes of GII decoding have been developed in [23], [24] and they can be easily extended to GII Chase decoding. However, for codes with larger τ_0, a sub-word is less likely to be miscorrected and the error-correcting performance degradation resulted from miscorrections can be ignored.

To achieve the same error-correcting performance as the original Chase decoding, larger η is needed when our proposed polynomial selection scheme is utilized. Having more bits to flip leads to longer latency in the KES step. However, each additional test vector only introduces a few more clock cycles to the KES process using a parallel hardware implementation architecture for the one-pass algorithm. On the other hand, a highly parallel hardware architecture for the Chien search has overwhelming complexity and routing congestion issues. The Chien search with reasonable hardware complexity typically takes many clock cycles to finish. Since our proposed scheme only activates the Chien search when the error-locator polynomial has the $x^2 - 1$ factor, it has much shorter overall latency even if η needs to be increased.

The pre-set bit is flipped to be erroneous in our proposed design. To achieve the same error-correcting performance as the conventional Chase decoding with η-bit flipping, our design needs to flip more than $\eta + 1$ bits besides the pre-set bit since the pre-set bit is definitely flipped to be wrong and the other $\eta + 1$ bits to flip in the test vectors may not be erroneous. From Fig. 5, our proposed scheme with $\eta = [5, 4, 0, 0]$ can achieve similar error-correcting performance as the conventional Chase process with $\eta = [3, 2, 0, 0]$. The performance of the proposed scheme further improves with larger η, as shown by the curves for $\eta = [8, 7, 6, 0]$ in Fig. 5. However, the worst-case decoding latency increases substantially with η.

V. Remainder pre-computation for Pre-bit-flipping polynomial selection

In conventional one-pass Chase decoding, the error-locator polynomial for one test vector is derived each time. Although the proposed pre-bit-flipping polynomial selection substantially reduces the latency of the overall one-pass Chase decoding by avoiding the Chien search most of the time, it needs a larger η. Deriving the error-locator polynomial for more test vectors leads to longer latency in the KES step. To further reduce the Chase decoding latency, this section proposes to pre-compute the remainders of dividing the error-locator polynomials by $x^2 - 1$ first, without computing the polynomials themselves. A low-complexity method is developed to compute multiple remainder polynomials at the

Algorithm 2 Proposed pre-bit flipping one-pass Chase algorithm with remainder pre-computation

Input: $\alpha_0, \cdots, \alpha_{\eta-1}$, and $\Lambda^{(0)}(x)$ and $B^{(0)}(x)$ from hard-decision decoding
1. $\sigma_1 = [1, 0, \cdots, 0]$ (vector with only the pre-set bit flipped)
2. Derive $\Lambda^{(1)}(x)$ and $B^{(1)}(x)$ from $\Lambda^{(0)}(x)$ and $B^{(0)}(x)$ by using the one-pass Chase KES Algorithm in Algorithm 1.
3. $r_{\Lambda}^{(1)}(x) = \Lambda^{(1)}(x) \text{mod } x^2 - 1$,
$r_{\mathcal{B}}^{(1)}(x) = \mathcal{B}^{(1)}(x) \text{mod } x^2 - 1$
for $(i = 2;\ i \leq 2^{\eta};\ i++)$:
4. Find $\sigma_j (j < i)$ such that σ_i equals σ_j with an extra '1' in the l-th bit
5. Compute $a_l^{(j)} = \Lambda^{(j)}(\alpha_l^{-1}),\ b_l^{(j)} = \mathcal{B}^{(j)}(\alpha_l^{-1})$
6. **Case 1:** $a_l^{(j)} = 0\ |(a_l^{(j)} \neq 0\ \&\ b_l^{(j)} \neq 0\ \& L_{\Lambda}^{(j)} \geq L_{\mathcal{B}}^{(j)})$
7. $r_{\Lambda}^{(i)}(x) = b_l^{(j)} r_{\Lambda}^{(j)}(x) + a_l^{(j)} r_{\mathcal{B}}^{(j)}(x)$
8. $r_{\mathcal{B}}^{(i)}(x) = (1 - \alpha_l^{-2}) r_{\mathcal{B}}^{(j)}(x)$
9. $L_{\Lambda}^{(i)} = L_{\Lambda}^{(j)},\ L_{\mathcal{B}}^{(i)} = L_{\mathcal{B}}^{(j)} + 2$
10. **Case 2:** $b_l^{(j)} = 0\ |(a_l^{(j)} \neq 0 \& b_l^{(j)} \neq 0 \& L_{\Lambda}^{(j)} < L_{\mathcal{B}}^{(j)} - 1)$
11. $r_{\Lambda}^{(i)}(x) = (1 - \alpha_l^{-2}) r_{\Lambda}^{(j)}(x)$
12. $r_{\mathcal{B}}^{(i)}(x) = b_l^{(j)} r_{\Lambda}^{(j)}(x) + \alpha_l^{-2} a_l^{(j)} r_{\mathcal{B}}^{(j)}(x)$
13. $L_{\Lambda}^{(i)} = L_{\Lambda}^{(j)} + 2,\ L_{\mathcal{B}}^{(i)} = L_{\mathcal{B}}^{(j)}$
14. **Case 3:** $a_l^{(j)} \neq 0\ \&\ b_l^{(j)} \neq 0\ \&\ L_{\mathcal{B}}^{(j)} = L_{\Lambda}^{(j)} - 1$
15. $r_{\Lambda}^{(i)}(x) = b_l^{(j)} r_{\Lambda}^{(j)}(x) + a_l^{(j)} r_{\mathcal{B}}^{(j)}(x)$
16. $r_{\mathcal{B}}^{(i)}(x) = b_l^{(j)} r_{\Lambda}^{(j)}(x) + \alpha_l^{-2} a_l^{(j)} r_{\mathcal{B}}^{(j)}(x)$
17. $L_{\Lambda}^{(i)} = L_{\Lambda}^{(j)} + 1,\ L_{\mathcal{B}}^{(i)} = L_{\mathcal{B}}^{(j)} + 1$
If $r_{\Lambda}^{(i)}(x)$ is zero:
18. Derive $\Lambda^{(i)}(x)$ by using the one-pass Chase KES Algorithm in Algorithm 1
19. Stop if the number of roots of $\Lambda^{(i)}(x)$ equals $\deg(\Lambda^{(i)}(x)) - 1$
end for
Output: $\Lambda^{(i)}(x)$

same time. Then the KES is only carried out to derive the error-locator polynomial with the zero remainder, followed by the Chien search. Since the error-locator polynomial of a correctable test vector with the pre-set bit flipped always has $x^2 - 1$ as a factor, the test vector corrected by the pre-bit flipping scheme proposed in the previous section is also correctable through pre-computing the remainders. Hence, the two proposed designs have exactly the same error-correcting performance.

As mentioned in the previous section, the remainder of dividing a polynomial by $x^2 - 1$ equals the XOR results of the even and odd coefficients of the polynomial. Therefore, the remainder of a linear combination of two polynomials equals the linear combination of the remainders of the two polynomials. After the remainder of vector σ_1, which is the test vector with only the pre-set bit flipped, is calculated by XOR operations, the remainders of the other test vectors can be derived by using similar updating as in Lines 9-20 of Algorithm 1.

The proposed pre-bit flipping one-pass Chase algorithm with remainder pre-computation is summarized in Algorithm 2. In this algorithm, $r_{\Lambda}^{(i)}(x)$ and $r_{\mathcal{B}}^{(i)}(x)$ denote the remainder polynomials of dividing $\Lambda^{(i)}(x)$ and $\mathcal{B}^{(i)}(x)$, respectively, by $x^2 - 1$. Since $(x^2 - \alpha_l^{-2})\mathcal{B}^{(j)}(x) = ((x^2 - 1) + (1 - \alpha_l^{-2}))\mathcal{B}^{(j)}(x)$, the remainder of dividing $(x^2 - \alpha_l^{-2})\mathcal{B}^{(j)}(x)$ by $x^2 - 1$ equals the remainder of dividing $(1 - \alpha_l^{-2})\mathcal{B}^{(j)}(x)$. Hence, Lines 11 and 14 in Algorithm 1 become Lines 8 and 11 in Algorithm 2 for remainder computation. Similar analysis can be extended to derive the formulas in Lines 12 and 16 of Algorithm 2. If $r_{\Lambda}^{(i)}(x)$ is zero, the corresponding error-locator polynomial $\Lambda^{(i)}(x)$ is derived by using the one-pass Chase KES Algorithm in Algorithm 1. The process of the remainder pre-computation stops when the number of roots of $\Lambda^{(i)}(x)$ is equal to its degree minus one. Since each of the remainders of dividing $x^2 - 1$ only has two coefficients, its computation has much lower complexity compared to those of long $\Lambda^{(i)}(x)$ and $\mathcal{B}^{(i)}(x)$. Accordingly, the remainders of multiple test vectors can be computed in parallel to reduce the latency of the KES step in the Chase decoding without bringing large silicon area overhead.

As shown in Algorithm 2, both the evaluation values, $a_l^{(j)}$ and $b_l^{(j)}$, and the lengths of the polynomials, $L_{\Lambda}^{(j)}$ and $L_{\mathcal{B}}^{(j)}$, are required to decide which of the three cases to use to calculate the remainder polynomials for each test vector. The lengths of $\Lambda^{(0)}(x)$ and $\mathcal{B}^{(0)}(x)$ for the hard-decision vector are available at the end of the KES. $L_{\Lambda}^{(1)}$ and $L_{\mathcal{B}}^{(1)}$ can be derived according to Algorithm 1. For the other test vectors, without deriving their error-locator polynomials, $\Lambda^{(i)}(x)$ and associated polynomials, $\mathcal{B}^{(i)}(x)$, the lengths of these polynomials can still be calculated by using the formulas in Lines 9, 13, and 17 of Algorithm 2. On the other hand, the evaluation values are more difficult to compute without deriving the polynomials themselves.

Assume that σ_j has one extra bit flipped at position α_r compared to σ_p $(j > p)$ in Algorithm 1. Then for $l \neq r$, $a_l^{(j)}$ and $b_l^{(j)}$ can be derived as linear combinations of $b_r^{(p)}$, $b_l^{(p)}$, $a_r^{(p)}$, and $a_l^{(p)}$ according to Lines 10, 11, 14, 15, 18 and 19 of Algorithm 1. These four evaluation values can be computed in turn by linear combinations of the evaluation values of the test vectors covered in previous iterations of Algorithm 1. Tracing back these computations in the tree of test vectors, such as that shown in Fig. 6, as long as the evaluation values of those parent test vectors including every flipped bit are available, the evaluation values of the later test vectors can all be computed by linear combinations. This means that actual polynomial evaluation only needs to be carried out the $\eta+1$ vectors in the first and second levels of the test vector tree, which include $\sigma_1, \sigma_2, \sigma_3$, and σ_4 in the example shown in Fig. 6.

Each of $\Lambda^{(i)}(x)$ and $\mathcal{B}^{(i)}(x)$ for $1 \leq i \leq \eta + 1$ needs to be evaluated over each α_l^{-1} for $1 \leq l \leq \eta$. Since polynomial evaluation has high complexity, the evaluation values of one pair of polynomials over one point are derived each time. However, each linear combination is implementable by a few multipliers. Hence those evaluation values of $\Lambda^{(i)}(x)$ and $\mathcal{B}^{(i)}(x)$ $(\eta + 1 < i \leq 2^{\eta})$ without data dependence can be computed in parallel with low complexity. If P pairs of evaluation values are calculated in each clock cycle, all evaluation values of $\sigma_{\eta+2}, \cdots, \sigma_{2^{\eta}}$ can be derived in $\lceil (2^{\eta} - \eta - 1)/P \rceil$ clock cycles. The corresponding remainder polynomials are computed according to Algorithm 2 right in one additional clock cycle after the evaluation values are available.

The proposed Chase decoding with remainder pre-computation for pre-bit-flipping polynomial selection is sum-

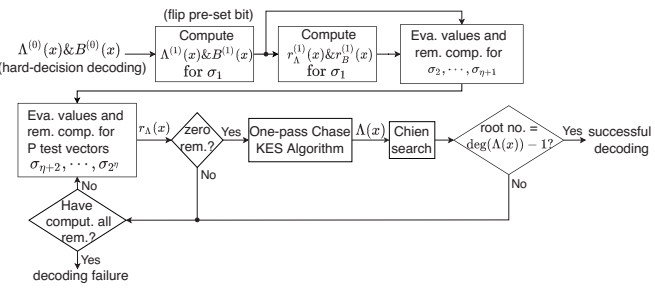


Fig. 7: Proposed Chase decoding process with remainder pre-computation for pre-bit-flipping polynomial selection.

marized in Fig. 7. Given $\Lambda^{(0)}(x)$ and $B^{(0)}(x)$ from the hard-decision decoding, $\Lambda^{(1)}(x)$ and $B^{(1)}(x)$ for vector σ_1 that flips the pre-set bit are derived by using Algorithm 1. Then the remainder polynomials $r_\Lambda^{(1)}(x)$ and $r_B^{(1)}(x)$ are computed by adding the even and odd coefficients of $\Lambda^{(1)}(x)$ and $B^{(1)}(x)$, respectively. Simultaneously, the evaluation values of $\Lambda^{(1)}(x)$ and $B^{(1)}(x)$ over α_1^{-1} through $\alpha_{\eta+1}^{-1}$ are computed one pair at a time. After they are calculated, the corresponding pair of remainder polynomials for vectors $\sigma_2,\cdots,\sigma_{\eta+1}$ are derived according to Algorithm 2. When all the evaluation values over α_2^{-1} through $\alpha_{\eta+1}^{-1}$ are available, the evaluation values for the rest remainder computations are derived through linear combinations and the corresponding remainder polynomials are derived afterward. Whenever the $r_\Lambda(x)$ of a test vector is found to be zero, the computations for the rest test vectors pause and the one-pass Chase KES Algorithm in Algorithm 1 is utilized to derive the error-locator polynomial for the selected test vector. Then the Chien search follows to find all the roots. If the root number is equal to the degree of the error-locator polynomial minus one, the decoding is considered successful. Otherwise, the remainder polynomials for the set of test vectors are calculated and this process is repeated until a correct test vector is found or decoding failure is declared.

The remainder pre-computation proposed in this section brings additional significant latency reduction to the overall Chase decoding process compared to our first proposed scheme. This is because that instead of deriving the long error-locator polynomial for one test vector at a time, our second design computes the error-locator polynomial only when its corresponding remainder polynomial of division by x^2-1 is zero. Since each remainder only has two coefficients, multiple remainders can be computed in parallel with low complexity.

VI. Error-rate formulas of soft-decision GII Chase decoding

At higher E_b/N_o, the error-correcting performance of the soft-decision GII Chase decoding and that of the proposed GII Chase decoding with pre-bit flipping can not be easily determined by simulations. This section develops formulas to estimate their FERs and BERs.

The FER and BER of hard-decision GII decoding can be estimated by the formulas proposed in [2], [25]. The formulas in [25] are more accurate compared to those in [2]. Let p_b denote the decoder input bit error rate. The probability of having w errors among the n bits of a sub-word is $\phi_w = \binom{n}{w} p_b^w (1-p_b)^{n-w}$. Then $\theta_{i,j} = \sum_{w=i}^{j} \phi_w$ is the probability of a sub-word having between i and j errors and the average number of erroneous bits can be calculated by $\varphi_{i,j} = \sum_{w=i}^{j} \phi_w \cdot w$. Consider the sub-word decoding as nested decoding round 0. The FER of hard-decision GII-BCH decoding can be estimated by adding up the failure rates of GII decoding in each nested decoding round as $P_f = \sum_{i=0}^{v} P_{f_i}$, where the probability of GII-BCH decoding failure in nested decoding round 0 is [25]

$$P_{f_0} = \sum_{b=v+1}^{m} \binom{m}{b} (\theta_{\tau_0+1,n})^b (\theta_{0,\tau_0})^{m-b}$$

and that for nested decoding round i $(1 \le i \le v)$ can be computed by

$$P_{f_i} = \binom{m}{v-i+1} (\theta_{\tau_i+1,n})^{v-i+1} (\theta_{0,\tau_{i-1}})^{m-(v-i+1)}.$$

Accordingly, the BER of GII-BCH codes is estimated in [25] as

$$P_b = \frac{\sum_{i=0}^{v} P_{f_i} \cdot \mathcal{N}_i}{m \cdot n},$$

where

$$\mathcal{N}_0 = \left(\frac{\sum_{b=v+1}^{m} \binom{m}{b} (\theta_{\tau_0+1,n})^b (\theta_{0,\tau_0})^{m-b} \cdot b}{P_{f_0}} \right) \cdot \frac{\varphi_{\tau_0+1,n}}{\theta_{\tau_0+1,n}} \tag{2}$$

is the average number of erroneous bits of an uncorrectable GII codeword in nested decoding round 0 and

$$\mathcal{N}_i = \frac{(v-i+1) \cdot \varphi_{\tau_i+1,n}}{\theta_{\tau_i+1,n}} \tag{3}$$

are those for nested decoding round $i = 1, 2, \cdots, v$.

GII decoding fails when at least $v+1$ sub-words are not correctable in nested decoding round 0. At most $v-i+1$ sub-words are sent to nested decoding round i $(i > 0)$ and decoding failure is also declared if none of them is corrected in round i. Unlike hard-decision GII decoding, soft-decision GII Chase decoding can correct up to $\tau_i + \eta_i$ errors in each sub-word by flipping the η_i least reliable bits. When the decoding fails in the i-th nested decoding round, each of the uncorrectable sub-words has either more than $\tau_i + \eta_i$ errors or between $\tau_i + 1$ and $\tau_i + \eta_i$ errors but those errors are not correctable by the Chase process due to wrong bits being chosen to flip. Hence the probability that the Chase decoding on a sub-word in nested round i fails can be calculated by

$$\Phi_{\tau_i+1,n} = \sum_{w=\tau_i+\eta_i+1}^{n} \phi_w + \sum_{w=\tau_i+1}^{\tau_i+\eta_i} \phi_w \cdot G_w^{\tau_i,\eta_i}.$$

In the above equation, $G_w^{\tau_i,\eta_i} (\tau_i + 1 \le w \le \tau_i + \eta_i)$ is the probability that the sub-word has w errors and flipping the η_i chosen bits does not generate any test vector that has up to τ_i erroneous bits. It can be determined from simulations over a limited number of samples. A tight bound involving complicated formulas for estimating such a probability can be also found in [26]. Then the probability that the soft-decision

GII Chase decoding fails in nested decoding round i can be estimated by

$$P'_{f_i} = \begin{cases} \sum_{b=v+1}^{m} \binom{m}{b} (\Phi_{\tau_0+1,n})^b (1-\Phi_{\tau_0+1,n})^{m-b}, \text{for } i=0 \\ \binom{m}{v-i+1} (\Phi_{\tau_i+1,n})^{v-i+1} \\ \qquad \cdot (1-\Phi_{\tau_{i-1}+1,n})^{m-(v-i+1)}, \text{for } i=1,\cdots,v. \end{cases} \tag{4}$$

As a result, the FER of GII-BCH Chase decoding can be computed by summing up the failure rates of each nested decoding round as $P'_f = \sum_{i=0}^{v} P'_{f_i}$.

For soft-decision GII Chase decoding, the average number of erroneous bits of an uncorrectable sub-word in nested decoding round i can be computed by

$$\Psi_{\tau_i+1,n} = \sum_{w=\tau_i+\eta_i+1}^{n} \phi_w \cdot w + \sum_{w=\tau_i+1}^{\tau_i+\eta_i} \phi_w \cdot G_w^{\tau_i,\eta_i} \cdot w.$$

By following a method similar to that for deriving (2) and (3), the average number of erroneous bits of an uncorrectable GII codeword in nested decoding round i is

$$\mathcal{N}'_i = \begin{cases} \left((\sum_{b=v+1}^{m} \binom{m}{b} (\Phi_{\tau_0+1,n})^b (1-\Phi_{\tau_0+1,n})^{m-b} \cdot b) / P'_{f_0} \right) \\ \cdot \Psi_{\tau_0+1,n} / \Phi_{\tau_0+1,n} \qquad \text{, for } i=0 \\ (v-i+1) \cdot \Psi_{\tau_i+1,n} / \Phi_{\tau_i+1,n} \qquad \text{, for } i=1,\cdots,v. \end{cases}$$

Therefore, the BER of GII-BCH Chase decoding can be estimated by

$$P'_b = \frac{\sum_{i=0}^{v} P'_{f_i} \cdot \mathcal{N}'_i}{m \cdot n}. \tag{5}$$

For AWGN channel with the binary phase shift keying (BPSK) modulation scheme, the input bit error rate can be computed by $p_b = Q(\sqrt{2RE_b/N_o})$ [29], where R is the code rate and $Q(x) = 1/2\pi \int_x^\infty \mathrm{e}^{-z^2/2} dz$. The FERs and BERs of the example GII-BCH Chase decoding estimated by using the proposed formulas in (4) and (5) are shown in in Fig. 5. They match the simulation results very well.

As mentioned in Section IV, the proposed GII Chase decoding flips the pre-set bit to an erroneous bit in the Chase process. To correct a sub-word with $\tau_i + j$ $(1 \le j \le \eta_i)$ errors, the proposed design needs to flip $j+1$ erroneous bits and only up to $\tau_i + \eta_i - 1$ errors besides the flipped pre-set bit can be corrected. When the proposed GII Chase decoding fails in nested decoding round i, each of the uncorrectable sub-words has either more than $\tau_i + \eta_i - 1$ errors or between $\tau_i + 1$ and $\tau_i + \eta_i - 1$ errors but those errors are not correctable by the Chase process. The probability that the Chase decoding fails to correct a sub-word in round i now becomes

$$\Phi'_{\tau_i+1,n} = \sum_{w=\tau_i+\eta_i}^{n} \phi_w + \sum_{w=\tau_i+1}^{\tau_i+\eta_i-1} \phi_w \cdot G_{w+1}^{\tau_i,\eta_i}. \tag{6}$$

The FER of the proposed GII-BCH Chase decoding with pre-bit flipping can be estimated by replacing the $\Phi_{\tau_i+1,n}$ in (4) by $\Phi'_{\tau_i+1,n}$. Besides, a formula similar to (5) can be used to estimate the BER. The estimated FERs and BERs of the proposed GII-BCH Chase decoding with pre-bit-flipping are shown in Fig. 5, and they also match the simulation results well.

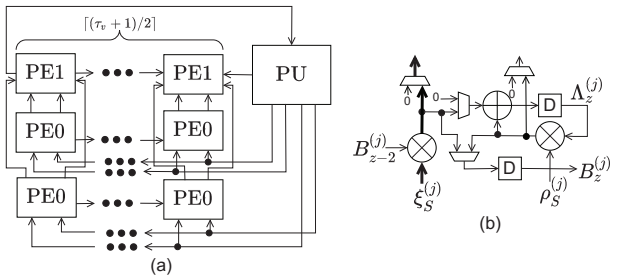


Fig. 8: GII-BCH nested KES architecture from [6]: a) overall architecture; b) details of one PE0.

VII. Fast and low-complexity GII-BCH Chase decoder architecture

In this section, a fast and low-complexity GII-BCH Chase decoder hardware implementation architecture is proposed. Our proposed decoding process consists of Chase decoding and GII nested decoding. Similar syndrome computation and Chien search architectures can be used for both of them, while their KES algorithms are different. However, the polynomials in the KES of the nested decoding process are also computed by linear combinations [3]–[6]. Therefore, the nested KES architectures can be shared to derive the error-locator polynomials for the one-pass Chase decoding. For hardware implementation, the combinational logic path that has the longest computation time is referred to as the critical path and it decides the maximum achievable clock frequency. The critical paths of the nested KES architectures in [3]–[6] consist of one multiplier. Nevertheless, directly implementing the one-pass Chase KES in Algorithm 1 leads to two multipliers in the critical path. To reduce the critical path to one multiplier, this section proposes reformulations to Algorithm 1 such that the two multiplications can be split and finished in two clock cycles. Additionally, our reformulation enables the sharing of intermediate results, which leads to lower hardware complexity. Similar reformulations are also applied to the remainder pre-computation to reduce its critical path and complexity.

Among the available KES architectures for GII nested decoding [3]–[6], the fast nested KES architecture design in [6] is the most efficient. One KES architecture is shared for all the nested decoding rounds. The nested decoding round i $(i > 1)$ has $\tau_i - \tau_{i-1}$ iterations in the KES. In the j-th iteration, the error-locator polynomial is derived as [6]

$$\Lambda^{(j+1)}(x) = \rho_S^{(j)} \Lambda^{(j)}(x) + \xi_S^{(j)} x^2 \mathcal{B}^{(j)}(x), \tag{7}$$

where $\rho_S^{(i)}$ and $\xi_S^{(i)}$ are scalars computed before the j-th iteration. $\mathcal{B}(x)$ is updated in one of two ways as

$$\mathcal{B}^{(j+1)}(x) = \rho_S^{(j)} \Lambda^{(j)}(x) \text{ or } \xi_S^{(j)} x^2 \mathcal{B}^{(j)}(x), \tag{8}$$

depending on whether the discrepancy coefficient is zero and the relative lengths of $\Lambda(x)$ and $\mathcal{B}(x)$. The overall block diagram of the nested KES architecture is shown in Fig. 8(a). ρ_S and ξ_S are derived in the pre-computation (PU) unit. For GII-BCH code with τ_v as the error-correcting capability of the last nested decoding round, $\lceil (\tau_v + 1)/2 \rceil$ groups of processing elements (PEs) are employed. Each group consists of one PE1

and two PE0s for updating one coefficient of the involved polynomials. PE1 computes the discrepancy polynomials. $\Lambda(x)$ and $\mathcal{B}(x)$ are updated in PE0s. The detail of one PE0 is illustrated in Fig. 8(b). An adder over finite field $GF(2^q)$ is implemented by bit-wise XOR. A multiplier is much more complicated and has much longer data path [27]. The fast KES architecture of GII-BCH decoder in [6] has one multiplier and four adders/multiplexers in the critical path. Parts of the critical path are located in the PU and PE1 and the rest in PE0 is highlighted by the thicker wires in Fig. 8(b).

From (7) and (8), it can be observed that the $\Lambda(x)$ and $\mathcal{B}(x)$ in the nested KES are updated as linear combinations in a similar way as the polynomials in Algorithm 1. Hence the KES architecture for the nested decoding can be shared to implement the one-pass Chase KES. However, directly implementing the formulas in Algorithm 1 would lead to two multipliers in the critical path since $\mathcal{B}^{(j)}(x)$ in Lines 15 and 19 need to be multiplied with two scalars: α_l^{-2} and $a_l^{(j)}$.

To reduce the critical path of the one-pass Chase KES, the computations of $\mathcal{B}^{(i)}(x)$ in Lines 15 and 19 of Algorithm 1 can be decomposed and completed in two clock cycles. In the first clock cycle, $\Lambda^{(j)}(x)$ and $\mathcal{B}^{(j)}(x)$ are multiplied with $b_l^{(j)}$ and $a_l^{(j)}$, respectively. These intermediate results are stored in the registers in Fig. 8(b). Then in the second clock cycle, $a_l^{(j)}\mathcal{B}^{(j)}(x)$ is multiplied with α_l^{-2} and the result is added with $b_l^{(j)}x^2\Lambda^{(j)}(x)$. Scaling a polynomial by x^2 does not require any multiplier. Instead, it is done by directly routing the coefficients to higher PE0s to taking care of the higher coefficients. As a result, each polynomial updating is completed in two clock cycles with one multiplier in the critical path.

In Case 3 of Algorithm 1, the $\Lambda^{(i)}(x)$ in Line 18 can be calculated by directly adding up the $b_l^{(j)}\Lambda^{(j)}(x)$ and $a_l^{(j)}\mathcal{B}^{(j)}(x)$ derived during the $\mathcal{B}^{(i)}(x)$ computation. For Case 2, multiplying $\Lambda^{(j)}(x)$ by α_l^{-2} in the first clock cycle of the iteration would require another multiplier in each PE0. On the other hand, waiting until the second clock cycle to compute this polynomial demands extra registers to store the coefficients of $\Lambda^{(j)}(x)$. To avoid these overheads, $\Lambda^{(j)}(x)$ is scaled by $b_l^{(j)}$ in our design. In this case, the intermediate result $b_l^{(j)}\Lambda^{(j)}(x)$ can be reused to get $b_l^{(j)}\Lambda^{(i)}(x)$ without extra registers or multipliers. The decoding output would not be affected by scaling $\Lambda^{(i)}(x)$ since the evaluation values of this polynomial and any linear combination in the following iterations are scaled by the same factor. Accordingly, the roots of the final error-locator polynomial, which are the inverse error locations, remain the same. Case 1 of Algorithm 1 can be handled in a way similar to Case 3 by scaling $\mathcal{B}^{(j)}(x)$ with $a_l^{(j)}$.

Our proposed reformulations for Algorithm 1 are summarized as follows.

- Reformulate Line 11 as
  $\mathcal{B}^{(i)}(x) = (x^2 - \alpha_l^{-2}) \cdot a_l^{(j)} \cdot \mathcal{B}^{(j)}(x)$
- Reformulate Line 14 as
  $\Lambda^{(i)}(x) = (x^2 - \alpha_l^{-2}) \cdot b_l^{(j)} \cdot \Lambda^{(j)}(x)$

To implement the reformulated one-pass Chase KES algorithm, the PE0 for updating the z-th coefficient of the error-locator and its associated polynomials is modified as in Fig. 9. In the first clock cycle of each iteration, $\Lambda^{(j)}(x)$ and $\mathcal{B}^{(j)}(x)$ are multiplied with $b_l^{(j)}$ and $a_l^{(j)}$, respectively, and the results are stored in the registers. Depending on which of the three cases of Algorithm 1 will be executed, these intermediate results are routed to different units in the second clock cycle to compute the coefficients of $\Lambda^{(i)}(x)$ and $\mathcal{B}^{(i)}(x)$. The extra units in Fig. 9 compared to the architecture in Fig. 8(b) are highlighted by gray color. An adder or 2-to-1 multiplexer over $GF(2^q)$ only takes q XOR gates to implement. On the other hand, a multiplier requires a much larger number of logic gates. For example, a $GF(2^{12})$ multiplier takes 201 gates to implement [4]. Therefore, sharing the hardware units between the nested KES and one-pass Chase KES leads to significant saving. Besides, the critical path is not increased. This is because that the additional hardware units in the modified PE0 does not alter the part of critical path, as highlighted by the thicker wires in Fig. 9. Also none of the other data paths has more than one multiplier and four adders/multiplexers.

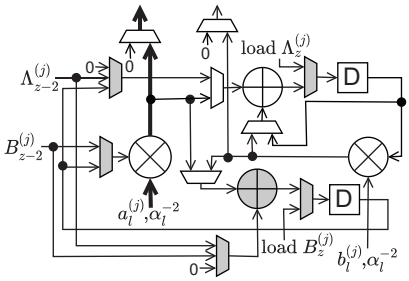


Fig. 9: Modified PE0 for joint nested KES and one-pass Chase KES.

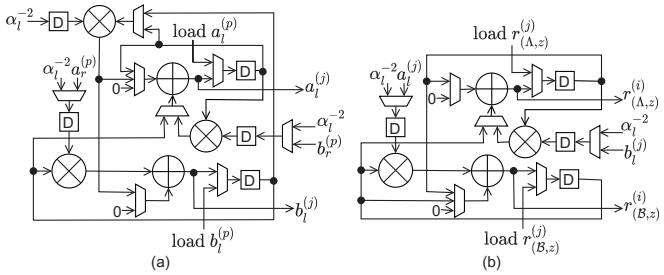


Fig. 10: Remainder pre-computation architectures for pre-bit flipping polynomial selection: a) architecture for evaluation value computation by linear combination, b) remainder computation architecture.

In the proposed Chase decoding with pre-bit-flipping polynomial selection, the number of bits to flip is $\eta_i + 1$ including the pre-set bit and the error-locator polynomial has at most $\tau_i + \eta_i + 1$ factors. From the previous sections, if the polynomial is correct, one of its factors is $x^2 - 1$. Therefore, the degree of the error-locator polynomial is at most $\max_{i=0,1,\cdots,v}\{\tau_i + \eta_i + 2\}$. This value increased by one is the number of PE0 in Fig. 8(a) that needs to be modified to accommodate the Chase KES.

As mentioned in the previous section, the evaluation values for vectors σ_1 through $\sigma_{\eta+1}$ are computed by actually evaluating the polynomials. The complexity of the polynomial evaluation can be reduced by applying the Horner's rule and

the implementation architecture is available in [3]. For the other test vectors, the evaluation values are computed by linear combinations. Assume that σ_j has one extra bit flipped at position α_r compared to σ_p $(j > p)$ in Algorithm 1. For $l \neq r$, the evaluation values, $a_l^{(j)}$ and $b_l^{(j)}$, can be derived as linear combinations of $b_r^{(p)}$, $b_l^{(p)}$, $a_r^{(p)}$, and $a_l^{(p)}$. The formulas for calculating $a_l^{(j)}$ and $b_l^{(j)}$ can be derived in similar format as those in Lines 9-20 of Algorithm 1 by tracing back to the j-th iteration, which $\Lambda^{(j)}(x)$ and $\mathcal{B}^{(j)}(x)$ are computed by the linear combinations of $\Lambda^{(p)}(x)$ and $\mathcal{B}^{(p)}(x)$, and substituting each x by α_l^{-1}. The corresponding implementation architecture is shown in Fig. 10(a). Unlike the polynomial updating, x^2 now become α_l^{-2}. The multiplication with this value requires one additional multiplier and it is located at the top-left corner in Fig. 10(a). For each test vector, one pair of the evaluation values is computed by using the architecture in Fig. 10(a). Since it has low complexity, multiple copies can be utilized to compute the evaluation values for multiple test vectors in parallel.

To keep only one multiplier in the critical path, the proposed remainder pre-computation in Algorithm 2 is reformulated in a similar way as Algorithm 1 and each iteration is carried out in two clock cycles. To enable the sharing of intermediate results, $r_\Lambda^{(j)}(x)$ and $r_B^{(j)}(x)$ are scaled by $b_l^{(j)}$ and $a_l^{(j)}$, respectively. The architecture in Fig. 10(b) computes one coefficient of $r_\Lambda^{(i)}(x)$ and $r_B^{(i)}(x)$ in two clock cycles and two copies are needed to compute both coefficients. In the first clock cycle, $r_\Lambda^{(j)}(x)$ and $r_B^{(j)}(x)$ are multiplied with $b_l^{(j)}$ and $a_l^{(j)}$, respectively. Also '0's are passed through the multiplexers to be the other inputs of the two adders, so that the two products are stored back into the registers. In the second clock cycle, depending on which of the three cases of the reformulated Algorithm 2 needs to be carried out, the intermediate results stored in the registers are routed to different units to complete the rest computations. Since the architecture in Fig. 10(b) also has low complexity, multiple copies can be utilized to compute the remainder polynomials for multiple test vectors in parallel.

VIII. GII-BCH Chase decoding latency and hardware complexity analysis

This section analyzes the latencies and hardware complexities of our proposed soft-decision GII-BCH Chase decoders using a $([8,3],4095)$ code over $GF(2^{12})$ with $[\tau_0,\tau_1,\tau_2,\tau_3] = [28,32,39,58]$ and $[\eta_0,\eta_1,\eta_2,\eta_3] = [5,4,0,0]$ as an example. This codeword length and code rate are considered for Flash memory applications. Our design is also compared to hard-decision GII-BCH decoder and alternative possible GII-BCH Chase decoding using the Chien-search-based polynomial selection.

A. Worst-case decoding latency analysis

As shown in Fig. 5, our proposed decoder achieves much better error-correcting performance compared to previous hard-decision GII-BCH decoders. The proposed decoding is carried out according to the flow chart in Fig. 4. In our design, the Chase process in nested decoding round i is activated when the number of uncorrected sub-words is larger than $v-i$ and it continues until this number is reduced to $v-i$. Therefore, in the worst case, the Chase decoding in round 0 needs to be carried out over all the m sub-words until it corrects $m-v$ sub-words. For nested decoding round i $(i > 0)$, the Chase decoding needs to be done for $v-i+1$ sub-words and corrects one sub-word at most. For each sub-word that the Chase decoding is carried out in nested round i, 2^{η_i} vectors are tested in the worst case. These numbers are listed in Table I for the $[8,3]$ example GII-BCH code. For this code, since $\eta_2 = \eta_3 = 0$, only the numbers for nested decoding round 0 and 1 are listed in separate columns in this table.

TABLE I: Worst-case latency of ([8,3], 4095) GII-BCH Chase decoding with $\tau = [28,32,39,58]$ and $\eta = [\eta_0,\eta_1,0,0]$ over $GF(2^{12})$ in different nested decoding rounds

Nested dec. round	0	1	0	1	0	1
	GII-BCH Chase [16] w/ Chien -search-based poly. select. ($[\eta_0,\eta_1]=[3,2]$)		Prop. GII-BCH Chase w/ pre-bit-flip. poly. select. ($[\eta_0,\eta_1]=[5,4]$)		Prop. GII-BCH Chase w/ pre-bit-flip. poly. select.& rem. pre-comput. ($[\eta_0,\eta_1]=[5,4]$)	
Worst-case # of sub-words carried out by Chase dec.	8	3	8	3	8	3
Worst-case # of sub-words corrected by Chase dec.	-	-	5	1	5	1
Worst-case # of vec. to test /sub-word	8	4	32	16	32	16
Worst-case nested dec. latency (# clks)	84	205	84	205	84	205
Worst-case Chase dec. latency (# clks)	3160	513	1056	203	554	124
Worst-case nested & Chase dec. latency(# clks) (normalized)	3244 (1)	718 (1)	1140 (0.35)	408 (0.57)	638 (0.20)	329 (0.46)
Worst-case nested dec. latency for round 2 & 3 (# clks)	nested decoding round 2: 274; nested decoding round 3: 184					

In the sub-word decoding, which is referred to as nested decoding round 0, $m = 8$ BCH decoders are employed to decode all the received sub-words in parallel. BCH decoder hardware implementation architectures can be found in many literature, such as [28]. The KES of traditional BCH decoding takes $\tau_0 = 28$ clock cycles. To improve the hardware utilization efficiency, the syndrome computation and Chien search steps should be completed in a similar number of clock cycles and this goal can be achieved by adjusting their parallelisms. As a result, the latency of the sub-word decoding is $3\tau_0 = 84$ clock cycles.

Architectures for the nested decoding have been developed in [3]–[6]. Since the later nested decoding rounds are activated with low probability, hardware units are shared among all the rounds. The KES in the i-th $(i > 0)$ nested decoding round takes $\tau_i - \tau_{i-1} + 1$ clock cycles for each sub-word by

using the fast nested KES architecture in [6]. Besides, the syndrome computation and Chien search architectures can use smaller parallelisms to reduce the complexity. Let $t_s^{(i)}$ and $t_c^{(i)}$ be the numbers of clock cycles needed for carrying out the syndrome computation for a sub-word and Chien search on an error locator polynomial, respectively, in nested decoding round i. The reconfigurable architecture in [3] designed to finish the Chien search on a polynomial with x coefficients in y clock cycles can also complete the search on a polynomial with up to ax ($a \in Z^+$) coefficients in ay clock cycles. To reduce the hardware complexity, assume that this architecture is configured to finish the Chien search on an error locator polynomial from nested decoding round 1 for the example GII-BCH code in $t_c^{(1)} = 56$ clock cycles. Then it can complete the Chien search for an error locator polynomial in round 2 and 3 nested decoding in $t_c^{(2)} = t_c^{(3)} = 56 \times 2 = 112$ clock cycles. The resource-shareable encoder in [22] helps to reduce the length of the polynomials involved and accordingly the complexity of the syndrome computation in the nested decoding. Even with half of the parallelism used for the Chien search, this architecture can finish the syndrome computation for a sub-word of the example code in nested decoding round 1, 2, and 3 in $t_s^{(1)} = 10$, $t_s^{(2)} = 20$, and $t_s^{(3)} = 50$ clock cycles, respectively. After the nested syndromes are calculated, 2 clock cycles are needed to convert them to higher-order syndromes for each sub-word [3].

In hard-decision GII-BCH decoding, the Chien search is carried out once after the KES for each sub-word in each nested decoding round. Since the Chien search takes much longer time than the KES, it can start to work on the error-locator polynomial of the next sub-word once it finishes the current one. Hence, the worst-case latencies of nested decoding round $i > 0$ can be calculated as $\tau_i - \tau_{i-1} + 3 + (v-i+1)(t_s^{(i)} + t_c^{(i)})$. Plugging in the values of τ_i, $t_s^{(i)}$, and $t_c^{(i)}$, it can be calculated that the numbers of clock cycles needed for nested decoding round $i = 1, 2, 3$ in hard-decision GII-BCH decoding in the worst case are $32 - 28 + 3 + (3 - 1 + 1)(10 + 56) = 205, 39 - 32 + 3 + (3 - 2 + 1)(20 + 112) = 274$, and $58 - 39 + 3 + (3 - 3 + 1)(50 + 112) = 184$, respectively.

For GII-BCH Chase decoding, the same numbers of clock cycles are needed for the nested decoding parts as in the hard-decision GII-BCH decoding and they are listed in Table I for the case that $t_s^{(1)} = 10$, $t_s^{(2)} = 20$, $t_s^{(3)} = 50$, $t_c^{(1)} = 56$ and $t_c^{(2)} = t_c^{(3)} = 112$. Next, analyses are carried out on the latencies needed by the Chase parts using different schemes.

Our first proposed polynomial selection scheme uses a pre-flipped bit to identify the correct error-locator polynomial. To achieve similar FER as the GII-BCH Chase decoding with $[\eta_0, \eta_1] = [3, 2]$ that uses Chien search for polynomial selection, $[\eta_0, \eta_1] = [5, 4]$ is adopted in our design due to the pre-flipped bit. Each pair of the evaluation values for updating the error-locator polynomial of a test vector can be computed in one clock cycle using two parallel polynomial evaluation architectures. Then the proposed joint KES architecture takes two clock cycles to derive the error locator polynomial for each test vector. The remainder computation is done in one clock cycle by simple additions and can be pipelined with the error-locator polynomial derivation for the next test vector. From simulations over 10^7 samples, the error-locator polynomial selected by our proposed scheme is always correct. Hence, for latency analysis, the Chien search is only carried out once for each sub-word that is correctable in the Chase decoding. Use $t'^{(i)}_c$ to represent the number of clock cycles for such Chien search in nested decoding round i. By reusing the reconfigurable architecture for the nested decoding, $t'^{(0)}_c = t'^{(1)}_c = 56$ clock cycles.

As listed in Table I, in the worst case, the Chase decoding is carried out on $m = 8$ and $v - i + 1 = 3 - 1 + 1 = 3$ sub-words in nested decoding round 0 and 1, respectively. Also since the proposed schemes terminate the Chase process when the nested decoding is able to continue for the next round, at most $m - v = 5$ and 1 sub-word are corrected in these two rounds. The Chien search is only carried out for those correctable sub-words in our scheme. As a result, the worst-case latencies of the Chase decoding using the first proposed design in nested decoding round 0 and 1 are $m \times (3 \times 2^{\eta_0} + 1) + (m - v) \times t'^{(0)}_c = 8 \times (3 \times 2^5 + 1) + 5 \times 56 = 1056$ and $(v - i + 1) \times (3 \times 2^{\eta_i} + 1) + t'^{(i)}_c = 3 \times (3 \times 2^4 + 1) + 56 = 203$ clock cycles, respectively. Adding the clock cycle numbers for the Chase and nested decoding parts, the overall worst-case latencies of the GII-BCH Chase nested decoding round 0 and 1 using our first proposed polynomial selection scheme are $1056 + 84 = 1140$ and $203 + 205 = 408$ clock cycles, respectively. Since $\eta_2 = \eta_3 = 0$, the latencies for nested decoding round 2 and 3 using our first design are the same as those for hard-decision GII-BCH decoding as listed in the last row of Table I.

Our second design pre-computes the remainders for the proposed pre-bit-flipping polynomial selection, and the overall decoding flow is summarized in Fig. 7. After the evaluation values are calculated in the first clock cycle, the joint KES architecture takes two more clock cycles to compute $\Lambda^{(1)}(x)$ and $\mathcal{B}^{(1)}(x)$ for vector σ_1. Then the remainder polynomials, $r_\Lambda^{(1)}(x)$ and $r_B^{(1)}(x)$, are derived in one clock cycle by adding the coefficients of $\Lambda^{(1)}(x)$ and $\mathcal{B}^{(1)}(x)$. Starting from the same clock cycle, one pair of the evaluation values for vectors σ_2 through $\sigma_{\eta+1}$ is calculated in each clock cycle. Once the evaluation values for a vector is available, the corresponding remainder polynomials are derived by linear combinations in 2 clock cycles using two sets of the proposed architecture in Fig. 10(b). The process from the derivation of $\Lambda^{(1)}(x)$ and $\mathcal{B}^{(1)}(x)$ to the remainder calculation for σ_2 through $\sigma_{\eta+1}$ takes $1+2+1+2\eta$ clock cycles for each sub-word. Once the evaluation values for σ_2 through $\sigma_{\eta+1}$ are available, the evaluation values for the rest $2^\eta - \eta - 1$ vectors can be derived by linear combinations. To reduce the hardware complexity, $P = 5$ pairs of evaluation values that do not have any data dependency are computed at a time by using P sets of the architecture in Fig. 10(a). Then the corresponding remainder polynomials are derived by using $2P$ sets of the architecture in Fig. 10(b). Since pipelining is applied between the architectures in Fig. 10(a) and 10(b), the remainder pre-computation for vectors $\sigma_{\eta+2}$ through σ_{2^η} takes $2 \times \lceil (2^\eta - \eta - 1)/P \rceil + 2$ clock cycles to complete for each sub-word.

When a zero remainder is found, the error-locator polynomial for the corresponding test vector is derived by using the proposed joint KES architecture. Since a vector has up to η bits flipped besides the pre-flipped bit compared to the hard-decision vector, this step takes up to 2η clock cycles. The roots of the error-locator polynomial is computed in $t_c'^{(i)} = 56$ $(i = 0, 1)$ clock cycles by sharing the reconfigurable Chien search architecture for the nested decoding. Since only one error-locator polynomial is actually computed for each sub-word that is correctable in the Chase decoding, the worst-case latencies of the Chase process in nested decoding round 0 and 1 using our second proposed approach are $m \times (3+1+2\eta_0 + 2\times\lceil(2^{\eta_0}-\eta_0-1)/P\rceil+2)+(m-v)\times(2\eta_0+t_c'^{(0)}) = 8\times(3+1+2\times5+2\times\lceil(2^5-5-1)/5\rceil+2)+5\times(2\times5+56) = 554$ and $(v-i+1)\times(3+1+2\eta_i+2\times\lceil(2^{\eta_i}-\eta_i-1)/P\rceil+2)+2\eta_i+t_c'^{(i)} = 3\times(3+1+2\times4+2\times\lceil(2^4-4-1)/5\rceil+2)+2\times4+56 = 124$ clock cycles, respectively. The worst-case latencies of the GII-BCH Chase nested decoding round 0 and 1 of our second design can be derived by adding up the latencies for the nested decoding parts as $84+554 = 638$ and $205+124 = 329$ clock cycles, respectively. For round 2 and 3, the latencies are the same as those for hard-decision GII-BCH decoding as listed in the last row of Table I since $\eta_2 = \eta_3 = 0$.

For comparisons, GII-BCH Chase decoding that adopts the one-pass Chase scheme in Algorithm 1 [16] and uses the Chien search to find the roots of every error-locator polynomial until the correct one is found is considered. Similar to the first design, it takes one clock cycle to compute one pair of the evaluation values and two clock cycles to derive the corresponding error-locator polynomial for each test vector. After the error-locator polynomial is computed, the Chien search is carried out and it takes $t_c'^{(i)} = 56$ $(i = 0, 1)$ clock cycles to compute the roots. The derivation of the error-locator polynomial for the next test vector is carried out simultaneously as the Chien search for the current test vector. In the worst-case, the Chien search is carried out for each of the $2^\eta - 1$ test vectors. Therefore, the latencies of the Chase decoding in nested decoding round 0 and 1 are $m\times(3+(2^{\eta_0}-1)\times t_c'^{(0)}) = 8\times(3+(2^3-1)\times56) = 3160$ and $(v-i+1)\times(3+(2^{\eta_i}-1)\times t_c'^{(i)}) = 3\times(3+(2^2-1)\times56) = 513$ clock cycles, respectively. As a result, the worst-case latencies of the GII-BCH Chase nested decoding round 0 and 1 using the Chien-search-based polynomial selection are $3160+84 = 3244$ and $513+205 = 718$ clock cycles, respectively.

The worst-case latencies of the GII-BCH Chase decoding using different polynomial selection schemes are summarized in Table I. Compared to the GII-BCH Chase decoding with Chien-search-based polynomial selection, our first design reduces the worst-case latencies of nested decoding round 0 and 1 in terms of the number of clock cycles by $(1-1140/3244)\times 100 = 65\%$ and $(1-408/718)\times100 = 43\%$, respectively, despite that larger η_i is used. By pre-computing the remainders for the proposed polynomial selection scheme, our second design further reduces the latencies by 15% and 11% for nested decoding round 0 and 1, respectively. For general cases, the formulas for calculating the worst-case latencies of the proposed designs are summarized in Table II.

TABLE II: Worst-case nested decoding and Chase decoding latencies for soft-decision $([m, v], n)$ GII-BCH Chase decoders with $\tau = [\tau_0, \tau_1, \cdots, \tau_v]$ and $\eta = [\eta_0, \eta_1, \cdots, \eta_v]$

	nested dec. round 0	nested dec. round i $(1 \le i \le v)$
soft-decision GII-BCH Chase dec. [16] w/ Chien-search-based poly. select.		
nested dec. latency	$3\tau_0$	$\tau_i - \tau_{i-1} + 3$ $+(v-i+1)\cdot(t_s^{(i)}+t_c^{(i)})$
chase dec. latency	$m\cdot(3+(2^{\eta_0}-1)\cdot t_c'^{(0)})$	$(v-i+1)\cdot(3+(2^{\eta_i}-1)\cdot t_c'^{(i)})$
proposed soft-decision GII-BCH Chase dec. w/ pre-bit-flipping poly. select.		
nested dec. latency	$3\tau_0$	$\tau_i - \tau_{i-1} + 3$ $+(v-i+1)\cdot(t_s^{(i)}+t_c^{(i)})$
chase dec. latency	$m\cdot(3\cdot2^{\eta_0}+1)$ $+(m-v)\cdot t_c'^{(0)}$	$(v-i+1)\cdot(3\cdot2^{\eta_i}+1)+t_c'^{(i)}$
proposed soft-decision GII-BCH Chase dec. w/ pre-bit-flipping poly. select. & remainder pre-comput.		
nested dec. latency	$3\tau_0$	$\tau_i - \tau_{i-1} + 3$ $+(v-i+1)\cdot(t_s^{(i)}+t_c^{(i)})$
chase dec. latency	$m\cdot(6+2\eta_0)+$ $m\cdot2\cdot\lceil(2^{\eta_0}-\eta_0-1)/P\rceil$ $+(m-v)\cdot(2\eta_0+t_c'^{(0)})$	$(v-i+1)\cdot(6+2\eta_i)+$ $(v-i+1)\cdot2\cdot\lceil(2^{\eta_i}-\eta_i-1)/P\rceil$ $+2\eta_i+t_c'^{(i)}$

$t_s^{(i)}$: number of clock cycles for syndrome computation in nested decoding round i.
$t_c^{(i)}$: number of clock cycles for the Chien search in nested decoding round i.
$t_c'^{(i)}$: number of clock cycles for the Chien search in Chase decoding in nested decoding round i.
P: parallelism of linear combination for remainder pre-computation of the second proposed soft-decision GII Chase decoder.

TABLE III: Average latencies of the Chase process in different nested rounds of ([8,3], 4095) GII-BCH Chase decoding with $\tau = [28, 32, 39, 58]$ and $\eta = [\eta_0, \eta_1, 0, 0]$ over $GF(2^{12})$ at $5.7dB$ E_b/N_o

Nested dec. round	0	1	0	1	0	1
	One-pass Chase [16]w/ chien-search -based poly. select. $([\eta_0, \eta_1] = [3, 2])$		Prop. one-pass Chase w/ pre-bit -flip. poly. select. $([\eta_0, \eta_1] = [5, 4])$		Prop. one-pass Chase w/ pre-bit -flip. poly. select. & rem. pre-comp. $([\eta_0, \eta_1] = [5, 4])$	
Average # sub-words carried out by Chase dec.	1.80	1.75	1.79	1.74	1.79	1.74
Average # sub-words correct. by Chase dec.	*0.98**	*0.86**	0.99	0.86	0.99	0.86
Average # vec. to test / sub-word	5.73	3.19	22.17	12.43	22.17	12.43
Average # clk cycles in Chase dec.(norm.)	482.18 (1.00)	219.87 (1.00)	176.28 (0.37)	114.78 (0.52)	107.25 (0.22)	89.21 (0.41)
Chase dec. act. prob.	4.7×10^{-5}	3.6×10^{-6}	4.7×10^{-5}	3.6×10^{-6}	4.7×10^{-5}	3.6×10^{-6}
Average # clk cycles in Chase dec. aggr. w/act.prob.	2.2×10^{-2}	7.9×10^{-4}	8.3×10^{-3}	4.1×10^{-4}	5.1×10^{-3}	3.2×10^{-4}

* The average number of sub-words corrected by the Chase decoding in the Chien-search-based scheme does not affect the average number of clock cycles in the Chase decoding since the Chien search is carried out for each test vector regardless of whether it is correctable.

B. Average decoding latency analysis

The average latencies of the Chase process using different designs are summarized in Table III. Based on simulations

over 10^7 samples at $E_b/N_o = 5.7dB$, data are collected for the probability of activating the Chase decoding, the average number of sub-words over which the Chase decoding is carried out, the average number of sub-words corrected by the Chase decoding, and the average number of vectors need to be tested for each sub-word in the Chase decoding. The average numbers of clock cycles needed by the Chase decoding in our proposed design can be calculated by replacing the worst-case numbers in the formulas of Table II with the average numbers in Table III. For example, the average numbers of clock cycles needed by the Chase decoding in our first proposed design can be calculated as $1.79 \times (3 \times 22.17 + 1) + 0.99 \times t'^{(0)}_c = 176.28$ and $1.74 \times (3 \times 12.43 + 1) + 0.86 \times t'^{(0)}_c = 114.78$ for nested decoding round 0 and 1, respectively. Accordingly, our first proposed design reduces the average Chase decoding latencies by 1-176.28/482.18=63% and 1-114.78/219.87=48% for nested decoding round 0 and 1, respectively, compared to the Chien-search-based design. The average latencies are further reduced by 15% and 11% for nested decoding round 0 and 1, respectively, using the second proposed design.

The contribution of the Chase decoding to the average number of clock cycles in the overall GII-BCH decoding process can be calculated by aggregating the average numbers of clock cycles needed by the Chase processes with their activation probabilities as $4.7 \times 10^{-5} \times 176.28 = 8.3 \times 10^{-3}$ and $3.6 \times 10^{-6} \times 114.78 = 4.1 \times 10^{-4}$ for nested decoding round 0 and 1, respectively. For higher E_b/N_0, the Chase decoding is activated with even lower probability. Hence the average latency of the overall GII-BCH Chase decoding of our designs is dominated by that of the sub-word decoding, which is around 84 clock cycles. This is also very similar to those of the Chien-search-based GII Chase decoding and hard-decision GII decoding. The critical path of a circuit determines the maximum achievable clock frequency. The proposed design has 13 gates in the critical path as listed in Table IV. From [22], a design with 11 gates can easily achieve 1Ghz clock frequency using TSMC $65nm$ process. Hence, it can be estimated that the proposed design can achieve $11/13 = 850Mhz$ clock frequency and accordingly $(4095 \times 8) \times 850Mhz/84 = 330Gbit/s$ throughput.

C. Hardware complexity analysis

The hardware complexities of the proposed $([8, 3], 4095)$ GII-BCH Chase decoders over $GF(2^{12})$ are analyzed in architectural level and summarized in Table IV. The complexities are estimated in terms of the number of XOR gates needed to implement the design. A 2-to-1 multiplexer requires the same area as an XOR gate. The silicon 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. A $GF(2^{12})$ general multiplier can be implemented by the same area as 201 XOR gates [4]. The hard and soft-decision GII-BCH Chase decoders have the same sub-word decoder. A sub-word decoder consists of m τ_0-error-correcting BCH decoders. As mentioned before, the parallelisms of the syndrome computation and Chien search architectures are adjusted so that they can be completed in $\tau_0 = 28$ clock cycles in order to improve hardware efficiency. Such high parallelism makes the sub-word decoder large.

TABLE IV: Hardware complexities of $([8, 3], 4095)$ GII-BCH Chase decoders with $\tau = [28, 32, 39, 58]$ and $\eta = [\eta_0, \eta_1, 0, 0]$ over $GF(2^{12})$

Sub-word decoder (# XORs)	Nested decoder w/ joint Chase KES (# XORs)	Eva. value computation (# XORs)	Remainder computation (# XORs)	Total (# XORs)	Critical path (# gates)
Hard-decision GII-BCH decoder [3], [6]					
3920k	281k	-	-	4201k (1)	13
GII-BCH Chase decoding [16] with Chien-search-based polynomial selection ($[\eta_0, \eta_1] = [3, 2]$)					
3920k	296k	15.9k	-	4231.9k (1.01)	13
Proposed GII-BCH Chase decoder w/ pre-bit-flipping polynomial selection ($[\eta_0, \eta_1] = [5, 4]$)					
3920k	312k	17.5k	0.5k	4250k (1.01)	13
Proposed GII-BCH Chase decoder w/ pre-bit-flipping polynomial selection & remainder pre-computation ($[\eta_0, \eta_1] = [5, 4]$)					
3920k	312k	21.9k	7k	4260.9k (1.01)	13

The complexities of the nested decoders are listed in the second column of Table IV. For GII-BCH Chase decoders, the complexities of the joint KES architectures are included in this column. Their nested decoders are larger than that of the hard-decision GII-BCH decoder. Besides, the Chien search architecture is designed to support longer polynomials as needed to incorporate the Chase decoding. Therefore, the nested decoders in our two proposed designs are larger than that of the Chien-search-based decoder since they use larger η. Longer error-locator polynomials also require more gates to compute the evaluation values. Besides, our second proposed design employs multiple sets of the architectures in Fig. 10 to compute the evaluation values and remainder polynomials. Hence, it requires larger area compared to our first design as shown in Table IV. Nevertheless, the overall area of the GII-BCH Chase decoder is dominated by that of the sub-word decoder. As a result, the two proposed designs only have 1% area overhead compared to the hard-decision decoder, while achieving significant coding gain as illustrated in Fig. 5. The area overheads of our two proposed designs over the Chien-search-based GII-BCH Chase decoder are even more negligible. Besides, in both GII-BCH and GII-BCH Chase decoders, the critical path lies in the nested KES architecture and it consists of one multiplier and four adders/multiplexers. Since a $GF(2^{12})$ general multiplier has 9 gates in the data path [4], the critical paths of our proposed GII-BCH Chase decoders have $9 + 4 = 13$ gates, which is the same as those of the hard-decision GII-BCH decoder and Chien-search-based GII-BCH Chase decoder.

D. Discussions

For longer codes, the Chien search takes more clock cycles. In this case, the two proposed designs would achieve more significant latency reduction. Besides, the second proposed design can achieve even further latency reduction with smaller overhead compared to the first one by using a larger parallelism for remainder pre-computation. Larger η leads to better error-correcting performance. Another advantage of our proposed designs is that the overall worst-case decoding latency

increases in a much slower pace with η compared to the Chien-search-based design.

For soft-decision GII-RS decoding, the proposed scheme can be also employed to integrate the Chase process into the GII decoding. Each symbol of a RS codeword over $GF(2^q)$ is a q-bit finite field element. To avoid applying the Chien search on each test vector, the proposed pre-bit-flipping polynomial selection can flip any bit in the first symbol of the RS codeword and the error-locator polynomial of a correctable test vector can still be detected by computing its remainder of the division by $x^2 - 1$. Also multiple remainders can be pre-computed to further reduce the decoding latency. To construct a joint nested and Chase KES architecture for RS codes, similar reformulations can be applied to the one-pass Chase algorithm for RS codes in [16] to reduce the data path to one multiplier and share intermediate results.

An alternative scheme to avoid the expensive Chien search on invalid error-locator polynomials is to only carry the Chien search when $\deg(\Lambda(x)) < \tau + l$, where l is the number of bits flipped in the Chase process. This alternative method also sacrifices one-bit error-correcting capability and needs to compute the evaluation values of $\Lambda(x)$ and $\mathcal{B}(x)$ in order to derive $\deg(\Lambda(x))$. Therefore, the proposed scheme and this alternative method have the same error-correcting performance as well as similar hardware complexity and decoding latency.

IX. Conclusions

For the first time, this paper considers soft-decision GII decoding. Different methods of incorporating the Chase process into the GII decoding are analyzed and compared to identify the best GII-BCH Chase scheme. Besides, a new polynomial selection scheme that pre-flips a pre-set bit is proposed to reduce the latency of the Chase process. The latency of the proposed polynomial selection is further reduced by pre-computing the remainder polynomials. Formulas for analyzing the error-correcting performance of the proposed designs are given. Furthermore, low-overhead hardware architectures are developed to efficiently implement the proposed GII Chase decoders. The proposed decoders can achieve significant coding gain over hard-decision decoders with negligible hardware overhead. Our designs also substantially reduce the worst-case decoding latency compared to the best alternative GII Chase decoder. Future research will address further optimizing the hardware implementation architectures of GII Chase decoders.

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] 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.

[4] 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.

[5] 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.

[6] Z. Xie and X. Zhang, "Scaled fast nested key equation solver for generalized integrated interleaved BCH decoders," *IEEE Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP)*, Toronto, ON, Canada, 2021, pp. 7883-7887.

[7] E. R. Berlekamp, *Algebraic Coding Theory*, New York, NY, USA: McGraw-Hill, 1968.

[8] D. Chase, "A class of algorithms for decoding block codes with channel measurement information," *IEEE Trans. on Info. Theory*, vol. 18, no. 1, pp. 170-182, Jan. 1972.

[9] M. Sudan, "Decoding of Reed-Solomon codes beyond the error-correction bound," *Journal of Complex.*, vol. 13, no. 1, pp. 180-193, 1997.

[10] V. Guruswami and M. Sudan, "Improved decoding of Reed-Solomon and algebraic-geometry codes," *IEEE Trans. on Info. Theory*, vol. 45, no. 6, pp. 1757-1767, Sept. 1999.

[11] Y. Wu, "New list decoding algorithms for Reed–Solomon and BCH codes," *IEEE Trans. on Info. Theory*, vol. 54, no. 8, pp. 3611-3630, Aug. 2008.

[12] G. Forney, "Generalized minimum distance decoding," *IEEE Trans. on Info. Theory*, vol. 12, no. 2, pp. 125-131, Apr. 1966.

[13] R. Koetter, "Fast generalized minimum-distance decoding of algebraic-geometry and Reed-Solomon codes," *IEEE Trans. on Info. Theory*, vol. 42, no. 3, pp. 721-737, May 1996.

[14] D. Agrawal and A. Vardy, "Generalized minimum distance decoding in Euclidean space: performance analysis," *IEEE Trans. on Info. Theory*, vol. 46, no. 1, pp. 60-83, Jan. 2000.

[15] N. Kamiya, "On algebraic soft-decision decoding algorithms for BCH codes," *IEEE Trans. on Info. Theory*, vol. 47, no. 1, pp. 45-58, Jan. 2001.

[16] Y. Wu, "Fast Chase decoding algorithms and architectures for Reed-Solomon codes," *IEEE Trans. on Info. Theory*, vol. 58, no. 1, pp. 109-129, Jan. 2012.

[17] X. Zhang, J. Zhu and Y. Wu, "Efficient one-pass chase soft-decision BCH decoder for multi-level cell NAND flash memory," *IEEE Int. Mid-west Symp. on Circuits and Systems (MWSCAS)*, Seoul, Korea (South), 2011.

[18] N. Zheng and P. Mazumder, "An efficient eligible error locator polynomial searching algorithm and hardware architecture for one-pass Chase decoding of BCH codes," *IEEE Trans. on Circuits and Syst.-II*, vol. 64, no. 5, pp. 580-584, May 2017.

[19] X. Zhang, Y. Wu, J. Zhu and Y. Zheng, "Novel interpolation and polynomial selection for low-complexity Chase soft-decision Reed-Solomon decoding," *IEEE Trans. on VLSI Syst.*, vol. 20, no. 7, pp. 1318-1322, July 2012.

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

[21] 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.

[22] Y. J. Tang and X. Zhang, "Low-complexity resource-shareable parallel generalized integrated interleaved encoder," *IEEE Trans. on Circuits and Syst.-I*, vol. 69, no. 2, pp. 694-706, Feb. 2022.

[23] Z. Xie and X. Zhang, "Miscorrection mitigation for generalized integrated interleaved BCH codes," *IEEE Commun. Letters*, vol. 25, no. 7, pp. 2118-2122, Apr. 2021.

[24] Z. Xie and X. Zhang, "Improved miscorrection detection for generalized integrated interleaved BCH codes," *Proc. IEEE Int. Conf. Commun.*, 2022.

[25] H. Cui, S. Song and Z. Wang, "An improved method for performance analysis of generalized integrated interleaved codes," *IEEE Comm. Letters*, vol. 25, no. 10, pp. 3166-3169, Oct. 2021.

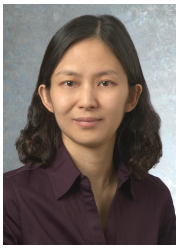
[26] M. P. C. Fossorier and Shu Lin, "Error performance analysis for reliability-based decoding algorithms," *IEEE Trans, on Info. Theory*, vol. 48, no. 1, pp. 287-293, Jan. 2002.

[27] X. Zhang and Y. Lao, "On the construction of composite finite fields for hardware obfuscation," *IEEE Trans. on Comp.*, vol. 68, no. 9, pp. 1353-1364, Sept. 2019.

[28] X. Zhang, *VLSI Architectures for Modern Error-Correcting Codes*, Boca Raton, FL, USA: CRC Press, 2015.

[29] 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 error-correcting codes.



Xinmiao Zhang received her Ph.D. degree in Electrical Engineering from the University of Minnesota. She is currently a Professor at the Ohio State University. She was a Senior Technologist at Western Digital/ SanDisk 2013-2017. Prior to that, she was a Timothy E. and Allison L. Schroeder Associate Professor at Case Western Reserve University. Prof. Zhang's research spans the areas of VLSI architecture design, digital storage and communications, cryptography, security, and signal processing.

Prof. Zhang is a recipient of the NSF CAREER Award 2009, the College of Engineering Lumley Research Award at The Ohio State University 2022, the Best Paper Award at ACM Great Lakes Symposium on VLSI 2004, and Best Paper Award at International SanDisk Technology Conference 2016. She authored "VLSI Architectures for Modern Error-Correcting Codes" (CRC Press, 2015), and co-edited "Wireless Security and Cryptography: Specifications and Implementations" (CRC Press, 2007). Prof. Zhang was elected the Vice President-Technical Activities of the IEEE Circuits and Systems Society (CASS) 2022-2023 and served on the Board of Governors of CASS 2019-2021. She was also the Chair (2021-2022) and a Vice-Chair (2017-2020) of the Data Storage Technical Committee (DSTC) of the IEEE Communications Society. She served on the technical program and organization committees of many conferences, including ISCAS, ICC, GLOBECOM, SiPS, GlobalSIP, MWSCAS, 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.