Fast Nested Key Equation Solvers for Generalized Integrated Interleaved Decoder

Zhenshan Xie and Xinmiao Zhang, *Senior Member, IEEE,*

***Abstract*—Generalized integrated interleaved (GII) codes nest Reed-Solomon (RS) or BCH sub-codewords to generate codewords belonging to stronger RS or BCH codes. Their hyper-speed decoding and good error-correction capability make them one of the best candidates for next-generation terabit/s digital storage and communications. The key equation solver (KES) in the nested decoding stage causes clock frequency bottleneck and takes a large portion of the GII decoder area. Recent architectures reduce the critical path to two multipliers and rely on the application of the slow-down technique to further reduce it to one. The slow-down technique requires two sub-codewords to be interleaved in the nested KES. However, most of the time, the nested decoding only needs to be carried out on one sub-codeword and half of the clock cycles are wasted. This paper proposes two fast nested KES algorithms, both of which have one multiplier in the critical path without applying slow-down and accordingly reduce the latency of the nested KES to almost a half. The short critical path is achieved by algorithmic reformulations that enable the pre-computation of the scalars in parallel with polynomial updating. Our second design adopts scaled versions of the polynomials to enable product term sharing so that the number of multipliers in each pair of processing elements is reduced from 8 as in the first design to 4. Novel scaling and combined scalar computations are developed to keep the critical path one multiplier. For an example GII code over $GF(2^8)$ that has 3 nested codewords, our designs achieve 49.9% reduction on the number of clock cycles needed in the nested KES compared to prior designs. Besides, our second design requires 22% less area than the first one under the same timing constraint.**

***Index Terms*—Error-correcting codes, Generalized integrated interleaved codes, Nested decoding, Key equation solver, Reed-Solomon codes.**

I. Introduction

Next-generation terabit/s storage and digital communications, such as Flash memories, storage class memories, and optical communications, require error-correcting codes with not only hyper-speed decoding but also good correction capability. These requirements can be addressed by the generalized integrated interleaved (GII) codes [1]–[3], which nest a set of Reed-Solomon (RS) or BCH [4] sub-codewords to generate codewords of stronger RS or BCH codes. Most of the time, the errors are corrected by decoding individual sub-codewords and very high throughput can be reached. Besides, the nested codewords can be utilized to correct more errors. The GII decoding locality can be further improved by three-layer [5] nesting schemes.

The authors are with the Department of Electrical and Computer Engineering, The Ohio State University, Columbus, OH 43210 USA. This work is supported in part by Kioxia Corporation and the National Science Foundation under Award No. 2011785.

The GII decoding includes two stages [2]. The first stage is the traditional RS or BCH decoding on individual sub-codewords. The key equation solver (KES) step of the decoding iteratively computes error locator and evaluator polynomials using syndromes. Each iteration of the Berlekamp-Massey (BM) [6] KES algorithm first computes a discrepancy coefficient using a large multiplier-adder tree. Then the discrepancy coefficient is used for polynomial updating, which introduces another multiplier and an adder to the data path. The achievable clock frequency is limited by the long critical path. The reformulated inversionless Berlekamp-Massey (riBM) algorithm [7] initializes a discrepancy polynomial using syndromes and updates this polynomial simultaneously with the other polynomials. The discrepancy coefficient is just a coefficient of the discrepancy polynomial. As a result, the critical path is reduced to one multiplier and one adder.

In the second stage, higher-order syndromes acquired from the nested codewords are utilized to correct more errors. Although the KES of the nested decoding can continue from the KES results of sub-codeword decoding [2] to incorporate higher-order syndromes if the BM algorithm is used, the riBM algorithm cannot be applied to the nested KES, since the higher-order syndromes are not available at the very beginning of sub-codeword decoding to initialize the discrepancy polynomial. It was proposed in [8] to re-initialize the discrepancy polynomial at the beginning of the nested KES. This approach requires many large multiplier-adder trees and also leads to long critical path. The discrepancy coefficient computation was reformulated in [9] to incorporate one higher-order syndrome in each iteration so that the KES for the nested decoding can continue from the previous results and the critical path is reduced to two multipliers and two adders. The area is reduced by the scaled nested KES algorithm in [10] through scaling the polynomials and sharing product terms. Further area reduction can be achieved by eliminating the higher processing elements (PEs) of the nested KES with negligible degradation on the error-correcting performance [11].

It was proposed in [9]–[11] to apply the slow-down and retiming architectural transformation techniques [12] to reduce the critical path of the nested KES architecture by a half to one multiplier and a couple of adders. The application of the slow-down technique requires to interleave the decoding of two sub-codewords. However, with very high probability, there is only one sub-codeword with extra errors that requires the nested decoding, in which case dummy zeros are inserted and half of the clock cycles are wasted.

This paper proposes two fast nested KES algorithms that indeed have one multiplier in the critical path and do not rely on the slow-down technique. Using our new algorithms, the latency of the nested KES is reduced to almost a half. In previous designs [9]–[11], one multiplier in the critical path is used to compute the polynomial scalars, and the other is contributed by multiplying the scalars to the polynomials. By analyzing the data dependencies, algorithmic reformulations are developed in this paper to pre-compute the scalars. Moreover, the pre-computation is carried out with one multiplier in the data path in parallel with the polynomial updating of previous iterations. As a result, the critical path is reduced to one multiplier. In our second design, a scaled fast nested KES algorithm is proposed to enable polynomial product term sharing and accordingly reduce the area substantially. Different from the scaling scheme in [10], the scalar pre-computation with one multiplier in the data path brings many new challenges. Alternative scaling strategies are developed to simplify the combined scalars and modified scalars are adopted to enable the pre-computation using one multiplier in the critical path without large area overhead. Additionally, the methodologies for adjusting the polynomial updating scalars in the cases that the scalars introduced for product term sharing are zero are re-designed. Besides, a novel modification is proposed to enable the usage of the results from our scaled algorithm in the Horiguchi-Koetter (HK) formula [13] for error magnitude computation, which helps to further reduce the number of PEs in the scaled nested KES architecture with very small error-correcting performance degradation.

In summary, this paper has the following major contributions.

- Algorithmic reformulations are proposed to pre-compute the scalars in the nested KES with one multiplier in the data path in parallel with the polynomial updating of previous iterations. As a result, the critical path is reduced to one multiplier and two adders without using architectural transformations.
- Novel scaling schemes and combined scalar pre-computations are developed to enable the sharing of product terms. Accordingly, the number of multipliers in each PE pair is reduced from 8 to 4 with one multiplier and three adders/multiplexers in the critical path.
- Non-trivial methodologies are proposed to adjust the polynomial updating in the cases of zero scalars and make up for the scalars of the polynomials to be used in the following error magnitude computation.
- Efficient architectures are developed to implement the proposed schemes and detailed hardware complexity analyses are carried out.

For an example GII code over $GF(2^8)$ that has 8 sub-codewords and 3 nested codewords, both of the proposed designs reduce the average nested KES latency to almost a half in terms of the number of clock cycles compared to the designs in [9], [10]. Moreover, synthesis results show that the second design achieves 22% area reduction over the first one and has smaller area than the prior design [9] under the same timing constraint.

This paper is organized as follows. Section II introduces the decoding of GII codes. The proposed fast nested KES algorithm and architecture are detailed in Section III. The scaled fast nested KES algorithm and architecture are described in Section IV. Section V provides hardware complexity analyses and comparisons. Conclusions are drawn in Section VI.

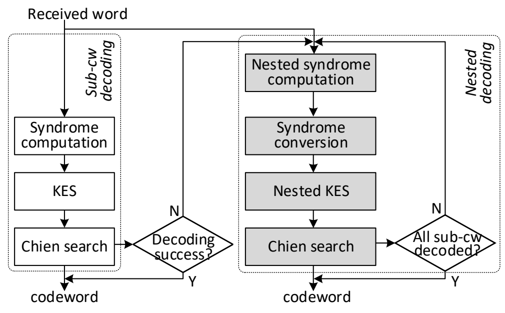


Fig. 1. Data flow of GII decoding.

II. GII Codes and Decoding

Let $\mathcal{C}_v \subseteq \mathcal{C}_{v-1} \subseteq \cdots \subseteq \mathcal{C}_1 \subset \mathcal{C}_0$ be $v+1$ RS or BCH codes over $GF(2^q)$. A GII codeword consists of m sub-codewords $c_0, c_1, \cdots, c_{m-1} \in \mathcal{C}_0$, and v codewords $\tilde{c}_l \in \mathcal{C}_{v-l}$ of higher correction capabilities are formed by nesting the m sub-codewords. The GII code can be defined as [1], [2]

$$\mathcal{C} \triangleq \left\{ c = [c_0, \cdots, c_{m-1}] : c_i \in C_0, \tilde{c}_l = \sum_{i=0}^{m-1} \alpha^{il} c_i \in \mathcal{C}_{v-l}, 0 \leq l < v \right\}, \tag{1}$$

where α is a primitive element of $GF(2^q)$. Although GII codes have been alternatively described by the parity check matrix [14], the decoding complexity is much higher.

GII encoding data flow and architectures can be found in [15], [16]. The GII decoding includes two stages as shown in Fig. 1. Let the error-correction capability of $\mathcal{C}_i$ be t_i $(0 \leq i \leq v)$. The first stage is the traditional RS or BCH decoding on individual sub-codewords, which can correct up to t_0 errors in each sub-codeword. Denote a received sub-codeword by $y(x)$. In traditional RS decoding, first, $2t_0$ syndromes are computed as $S_j = y(\alpha^{j+1})$ $(0 \leq j < 2t_0)$. If all $2t_0$ syndromes are zero, $y(x)$ has no error. Otherwise, a KES, such as the BM algorithm [6], takes the $2t_0$ syndromes and iteratively determines an error locator polynomial $\Lambda(x)$ and an error evaluator polynomial $\Omega(x)$ satisfying $\Omega(x) \equiv \Lambda(x)S(x) \mod x^{2t_0}$, where $S(x) = \sum_{j=0}^{2t_0-1} S_j x^j$. Then the error locations and magnitudes can be calculated by carrying out the Chien search on $\Lambda(x)$ and $\Omega(x)$.

Define a sub-codeword with more than t_0 errors as an exceptional sub-codeword. The second-stage nested decoding is activated when the number of exceptional sub-codewords, b, is between one and v. $2t$ syndromes are required by the BM algorithm to correct t errors. To correct more than t_0 errors in a sub-codeword, higher-order syndromes can be derived from the nested codewords since they belong to more powerful codes. Syndromes of the nested codewords are computed as $\tilde{S}_j^{(l)} = \tilde{y}_l(\alpha^{j+1}) = \sum_{i=0}^{m-1} \alpha^{il} y_i(\alpha^{j+1})$ $(0 \leq l < v, 0 \leq j < 2t_{v-l})$, where $y_i(x)$ is the i-th received sub-codeword. Since all nested codewords are at least t_1-error-correcting, $2t_1 - 2t_0$ higher-order syndromes can be computed for each of the nested codewords. Syndromes are evaluation

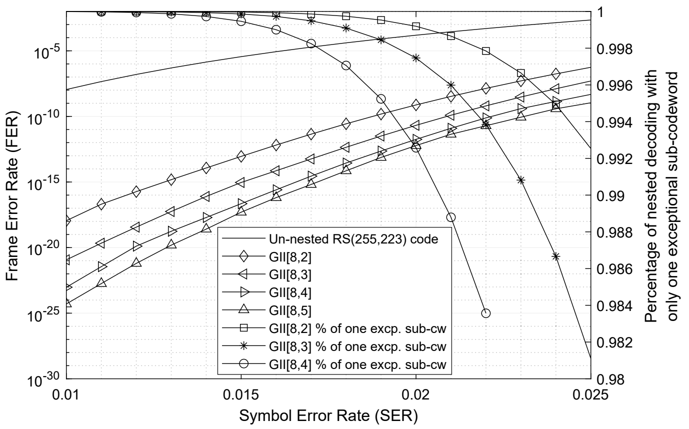


Fig. 2. FERs of GII codes over $GF(2^8)$ with $m = 8$, $v = 2, 3, 4$ and percentage of nested decoding with only one exceptional sub-codeword.

values of the codewords. Hence, the syndromes of the nested codewords are connected to those of the sub-codewords by the same nesting defined in (1). As a result, $2t_1 - 2t_0$ higher-order syndromes for each of the b exceptional sub-codewords can be converted from the nested syndromes as

$$\left[S_j^{(i_0)}, S_j^{(i_1)}, \cdots, S_j^{(i_{b-1})}\right]^T = A^{-1}\left[\tilde{S}_j^{(0)}, \tilde{S}_j^{(1)}, \cdots, \tilde{S}_j^{(b-1)}\right]^T, \tag{2}$$

where 'T' denotes transpose and $i_0, i_1, \cdots, i_{b-1}$ are the indices of the $b \leq v$ exceptional sub-codewords. The entry of A in the p-th row and q-th column is α^{pi_q} according to the nesting defined in (1). Then the KES step computes the error locator and evaluator polynomials according to the $2t_1$ syndromes of the sub-codewords, followed by the Chien search for error location and magnitude calculations. Assume that after the sub-codewords with up to t_1 errors are corrected, $b' < b$ exceptional sub-codewords remain. Next $2t_2 - 2t_1$ higher-order syndromes, $\tilde{S}_j^{(l)}$ $(0 \leq l < b', 2t_1 \leq j < 2t_2)$ are computed utilizing nested codewords $\tilde{c}_l$. These nested syndromes are converted to sub-codeword syndromes using an equation similar to (2) to correct the exceptional sub-codewords with up to t_2 errors. This process is repeated for up to $b \leq v$ rounds until all exceptional sub-codewords are corrected.

Sort the number of errors in the exceptional sub-codewords as $e_0 \geq e_1 \geq \cdots \geq e_{b-1}$. The nested decoding succeeds if $e_i \leq t_{v-i}$ for $0 \leq i < b$ [1], [2]. Fig. 2 plots the frame error rates (FERs) of example GII codes over $GF(2^8)$ with $m = 8$ and $v = 2, 3, 4$ over a range of input symbol error rates (SERs). The length of each sub-codeword is 255 symbols and the overall code rate is 87.45% for all codes in this figure. For reference, the FER of a conventional un-nested RS (255, 223) code of the same rate is also plotted. It can be observed that the FERs of the GII codes are several orders of magnitude lower compared to that of the conventional RS code. The overall decoder hardware complexities have been compared in [9] for the GII code with $v = 3$ and RS code shown in Fig. 2. It was found that, besides the several orders of magnitude lower FER, the GII decoder also achieves 18% higher throughput at the cost of only less than 30% area overhead.

The KES is the most complicated step of the sub-codeword decoding. In the r-th iteration of the BM algorithm [6], a discrepancy coefficient $\delta^{(r)} = \sum_{i=0}^{L_\Lambda^{(r)}} \Lambda_i^{(r)} S_{r-i}$, where $L_\Lambda^{(r)}$ is the length of $\Lambda^{(r)}(x)$, is first computed. Then the polynomials are updated using $\delta^{(r)}$. A large multiplier-adder tree is needed to compute $\delta^{(r)}$ and the following polynomial updating contributes to another multiplier and an adder in the data path. Such a long data path limits the achievable clock frequency of the overall decoder. For sub-codeword decoding, the riBM algorithm [7] reduces the critical path of the KES through introducing a discrepancy polynomial $\hat{\Delta}^{(r)}(x) = \Lambda^{(r)}(x)S(x)/x^r$, which is initialized as $\hat{\Delta}^{(0)}(x) = S(x)$. $\delta^{(r)}$ simply equals the constant coefficient of $\hat{\Delta}^{(r)}(x)$, denoted by $\hat{\Delta}_0^{(r)}$. Besides, $\hat{\Delta}^{(r)}(x)$ is updated in parallel with the other polynomials as a linear combination. As a result, the critical path is reduced to one multiplier and one adder. To cover $2t$ syndromes, $2t$ iterations are needed in both the BM and riBM algorithms.

The KES of the second-stage nested decoding does not have to restart. Using the BM algorithm, it can continue from the results of the sub-codeword decoding and only more iterations are needed to incorporate the higher-order syndromes. However, the critical path of the BM algorithm is long. On the other hand, the riBM algorithm cannot be used to continue the iterations since the higher-order syndromes are not available at the very beginning of the sub-codeword decoding to initialize the discrepancy polynomial. Although the discrepancy polynomial can be re-initialized according to the higher-order syndromes as proposed in [8], such re-initialization requires multiple large multiplier-adder trees, which lead to long critical path and large area overhead.

Algorithm 1: Nested KES Algorithm

```
input: $\Lambda^{(u)}(x)$, $B^{(u)}(x)$, $\hat{\Delta}^{(u)}(x)$, $\hat{\Theta}^{(u)}(x)$, $\gamma^{(u)}$, $k^{(u)}$ from previous KES; $S_i$ $(u \leq i < w)$; $S_w = 0$
initialization:
$\hat{\Delta}^{(u)}(x) = \hat{\Delta}^{(u)}(x) + S_u\Lambda^{(u)}(x)$
$\hat{\Theta}^{(u)}(x) = \hat{\Theta}^{(u)}(x) + S_uB^{(u)}(x)$
for $r = u, u+1, \cdots, w-1$
1).  $\Lambda^{(r+1)}(x) = \gamma^{(r)}\Lambda^{(r)}(x) + \hat{\Delta}_0^{(r)}xB^{(r)}(x)$
2).  $\hat{\Delta}^{(r+1)}(x) = \gamma^{(r)}(\hat{\Delta}^{(r)}(x)/x + S_{r+1}\Lambda^{(r)}(x))$
         $+\hat{\Delta}_0^{(r)}(\hat{\Theta}^{(r)}(x) + S_{r+1}xB^{(r)}(x))$
     if ($\hat{\Delta}_0^{(r)} \neq 0$ and $k^{(r)} \geq 0$)
3).      $B^{(r+1)}(x) = \Lambda^{(r)}(x)$
4).      $\hat{\Theta}^{(r+1)}(x) = \hat{\Delta}^{(r)}(x)/x + S_{r+1}\Lambda^{(r)}(x)$
5).      $\gamma^{(r+1)} = \hat{\Delta}_0^{(r)}$; $k^{(r+1)} = -k^{(r)} - 1$
     else
6).      $B^{(r+1)}(x) = xB^{(r)}(x)$
7).      $\hat{\Theta}^{(r+1)}(x) = \hat{\Theta}^{(r)}(x) + S_{r+1}xB^{(r)}(x)$
8).      $\gamma^{(r+1)} = \gamma^{(r)}$; $k^{(r+1)} = k^{(r)} + 1$
```

Reformulations on the discrepancy coefficient computation for the nested KES have been proposed in [9] to continue from the riBM results of the sub-codeword decoding or previous nested decoding round without requiring multiplier-adder trees. This reformulated nested KES algorithm is listed in Algorithm 1. It incorporates higher-order syndromes S_i $(u \leq i < w)$ one at a time into the discrepancy polynomial. $B(x)$ and $\hat{\Theta}(x)$ are auxiliary polynomials used to

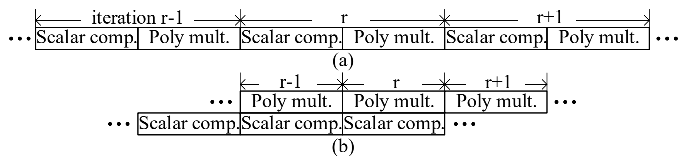


Fig. 3. Dependency of data in (a) previous nested KES; (b) proposed fast nested KES.

update $\Lambda(x)$ and $\hat{\Delta}(x)$, respectively. The scalars for $\Lambda^{(r)}(x)$ and $B^{(r)}(x)$ in Line 2 of Algorithm 1 are $\gamma^{(r)}S_{r+1}$ and $\hat{\Delta}_0^{(r)}S_{r+1}$, respectively. Hence the critical path of the hardware architecture implementing Algorithm 1 has a few adders and two multipliers, one for computing the combined scalars and the other for multiplying the scalars to the polynomials. The complexity of Algorithm 1 is reduced by adopting scaled versions of the polynomials to enable product term sharing in [10], which also has two multipliers in the critical path. It was proposed in [9], [10] to apply the slow-down technique [12] with a factor of two so that each register in the architecture is replaced by two registers, and then move the registers around by the re-timing technique [12] to cut the critical path to a half.

III. Fast Nested KES Algorithm

The previous nested KES architectures in [9], [10] rely on the application of the slow-down technique to reduce the critical path to one multiplier and a couple of adders. The application of the slow-down requires two sub-codewords to be interleaved. In the case that there is only one sub-codeword requiring nested decoding, dummy zeros are inserted after each data symbol. This means that half of the clock cycles of the nested KES are wasted. On the other hand, the majority of the nested decoding only involves one sub-codeword. The percentage of the cases that there is only one exceptional sub-codeword over all the cases that the nested decoding is activated is also plotted in Fig. 2 for example GII codes over $GF(2^8)$ with $m=8$ and $v=2,3,4$. It can be observed that the cases with only one exceptional sub-codeword dominate all cases of nested decoding. To further reduce the latency of the nested KES, it is essential to develop architectures that have short critical path without applying the slow-down technique so that no clock cycle is wasted when there is only one exceptional sub-codeword.

In this section, a fast nested KES algorithm is proposed to break the data dependency and reduce the critical path to one multiplier and two adders, which is just one adder longer compared to that of the slowed-down and re-timed nested KES architecture in [9]. As mentioned previously, there are two multipliers in the critical path for the nested KES algorithm in Algorithm 1. One is resulted from computing the combined scalars, such as $\gamma^{(r)}S_{r+1}$ in Line 2, and the other is contributed by multiplying the scalars to the polynomials. This data dependency is shown in Fig. 3(a). Essentially, our idea is to pre-compute the combined scalars by algorithmic reformulations, so that all the combined scalars needed for iteration r are ready before the beginning of iteration r as illustrated in Fig. 3(b). Then the polynomial scaling is completed in iteration r with one multiplier in the data path. Besides, the scalar pre-computation is done in parallel with the polynomial updating and should also have at most one multiplier in the data path although it can be accomplished in multiple clock cycles. In this case, the overall critical path would have only one multiplier. Such reformulation and pre-computation are made possible by analyzing the data dependencies and utilizing the fact that any computation on the syndromes can be carried out in advance since all syndromes are available before the nested KES starts.

In Algorithm 1, the scalars of polynomials for iteration r include $\gamma^{(r)}$, $\hat{\Delta}_0^{(r)}$, S_{r+1}, $\gamma^{(r)}S_{r+1}$, and $\hat{\Delta}_0^{(r)}S_{r+1}$. The first three are always available at the beginning of iteration r. Next, we will find out how to compute the other two, which are combined scalars, before iteration r with no more than one multiplier in the data path. From Lines 5 and 8 of Algorithm 1, it can be observed that $\gamma^{(r)}$ equals either $\gamma^{(r-1)}$ or $\hat{\Delta}_0^{(r-1)}$, which is available at the beginning of iteration $r-1$. Hence, $\gamma^{(r)}S_{r+1}$ can be computed as either $\gamma^{(r-1)}S_{r+1}$ or $\hat{\Delta}_0^{(r-1)}S_{r+1}$ in iteration $r-1$ with one multiplier in the data path in parallel with the polynomial updating in iteration $r-1$. From Line 2 of Algorithm 1,

$$\hat{\Delta}_0^{(r)}S_{r+1}=\gamma^{(r-1)}S_{r+1}\hat{\Delta}_1^{(r-1)}+\gamma^{(r-1)}S_rS_{r+1}\Lambda_0^{(r-1)} + \hat{\Delta}_0^{(r-1)}S_{r+1}\hat{\Theta}_0^{(r-1)}. \tag{3}$$

The three components on the right side of (3) can be computed as follows with one multiplier in the data path:
i) Compute $\gamma^{(r-1)}S_{r+1}$ in iteration $r-2$ as either $\gamma^{(r-2)}S_{r+1}$ or $\hat{\Delta}_0^{(r-2)}S_{r+1}$. Then the product is multiplied with $\hat{\Delta}_1^{(r-1)}$ in iteration $r-1$.
ii) S_rS_{r+1} can be calculated in advance. Similarly, $\gamma^{(r-1)}S_rS_{r+1}$ is derived in iteration $r-2$ as either $\hat{\Delta}_0^{(r-2)}(S_rS_{r+1})$ or $\gamma^{(r-2)}(S_rS_{r+1})$. In iteration $r-1$, $\gamma^{(r-1)}S_rS_{r+1}$ is multiplied to $\Lambda_0^{(r-1)}$.
iii) If Lines 3-5 of Algorithm 1 were executed in iteration $r-2$, $S_{r+1}\hat{\Theta}_0^{(r-1)}=S_{r+1}\hat{\Delta}_1^{(r-2)}+(S_{r+1}S_{r-1})\Lambda_0^{(r-2)}$. If Lines 6-8 were executed, $S_{r+1}\hat{\Theta}_0^{(r-1)}=S_{r+1}\hat{\Theta}_0^{(r-2)}$. In either case, $S_{r+1}\hat{\Theta}_0^{(r-1)}$ can be computed in iteration $r-2$ with one multiplier in the data path. Then it is multiplied to $\hat{\Delta}_0^{(r-1)}$ in iteration $r-1$.
The three components are added up in iteration $r-1$ to derive $\hat{\Delta}_0^{(r)}S_{r+1}$. Although the above scalar pre-computations are spread out over iterations $r-2$ and $r-1$, the data path only consists of one multiplier and two adders. Accordingly, our fast nested KES algorithm is developed as in Algorithm 2.

Algorithm 2 has the same inputs and outputs as Algorithm 1 and also requires $w-u$ iterations to incorporate $w-u$ syndromes. In this algorithm, $\gamma_S^{(r)}=\gamma^{(r)}S_{r+1}$ and $\delta_S^{(r)}=\hat{\Delta}_0^{(r)}S_{r+1}$. They are computed according to the explanations above with the assistance of intermediate values $h_1^{(r-1)}$, $h_2^{(r-1)}$, $\theta_S^{(r-1)}$, $g_1^{(r-1)}$, and $g_2^{(r-1)}$. All the scalars and intermediate results are updated in parallel with the polynomials. None of the lines in Algorithm 2 requires more than one multiplier and two adders in the data path.

The overall architecture for implementing the fast nested KES of Algorithm 2 is shown in Fig. 4. Since the degrees of the polynomials are at most t_v, t_v+1 pairs of PEs are employed to update the polynomials. PE0s are used to update

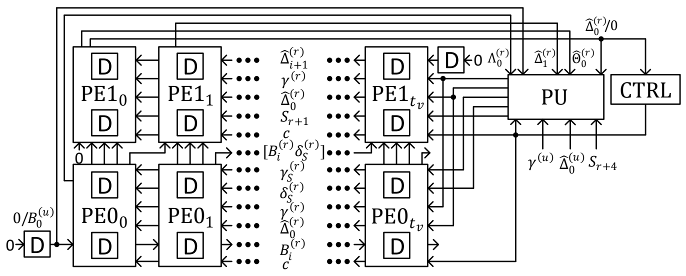


Fig. 4. Overall architecture for fast nested KES.

$\Lambda(x)$ and $B(x)$ and PE1s handle $\hat{\Delta}(x)$ and $\hat{\Theta}(x)$. All the scalars are computed in the pre-processing unit (PU) and one single PU is needed. The register in the bottom left corner is initialized with $B_0^{(u)}$ and then feeds '0' to PE0_0 and PU in later clock cycles. This architecture requires $w-u+1$ clock cycles to finish the nested KES for one sub-codeword. One clock cycle is used for initialization, and the others are spent on the $w-u$ iterations. Comparatively, prior designs [9], [10] take $2(w-u)+2$ clock cycles. It was found in [11] that a number of higher PE1s can be eliminated with negligible degradation on the error-correcting performance if the HK formula is used for the following error magnitude computation. This technique directly applies to our fast nested KES architecture.

Algorithm 2: Fast Nested KES Algorithm

```
input: $\Lambda^{(u)}(x)$, $B^{(u)}(x)$, $\hat{\Delta}^{(u)}(x)$, $\hat{\Theta}^{(u)}(x)$, $\gamma^{(u)}$, $k^{(u)}$ from
        previous KES; $S_i$ $(u \le i < w)$; $S_{w+i}=0$ $(0 \le i \le 3)$
initialization:
$\gamma_S^{(u)} = \gamma^{(u)} S_{u+1}$
$\delta_S^{(u)} = S_{u+1}\hat{\Delta}_0^{(u)} + S_{u+1}S_u\Lambda_0^{(u)}$
$\theta_S^{(u)} = S_{u+2}\hat{\Theta}_0^{(u)} + S_{u+2}S_u B_0^{(u)}$
$g_1^{(u)} = S_{u+1}S_{u+3}$; $g_2^{(u)} = S_{u+2}S_{u+3}$
$h_1^{(u)} = \gamma^{(u)}S_{u+2}$; $h_2^{(u)} = \gamma^{(u)}S_{u+1}S_{u+2}$
$\hat{\Delta}^{(u)}(x) = \hat{\Delta}^{(u)}(x) + S_u\Lambda^{(u)}(x)$
$\hat{\Theta}^{(u)}(x) = \hat{\Theta}^{(u)}(x) + S_u B^{(u)}(x)$
for $r = u, u+1, \cdots, w-1$
1).     $\Lambda^{(r+1)}(x) = \gamma^{(r)}\Lambda^{(r)}(x) + \hat{\Delta}_0^{(r)} x B^{(r)}(x)$
2).     $\hat{\Delta}^{(r+1)}(x) = \gamma^{(r)}\hat{\Delta}^{(r)}(x)/x + \gamma_S^{(r)}\Lambda^{(r)}(x)$
                $+\hat{\Delta}_0^{(r)}\hat{\Theta}^{(r)}(x) + \delta_S^{(r)} x B^{(r)}(x)$
3).     $\delta_S^{(r+1)} = h_1^{(r)}\hat{\Delta}_1^{(r)} + h_2^{(r)}\Lambda_0^{(r)} + \hat{\Delta}_0^{(r)}\theta_S^{(r)}$
4).     $g_1^{(r+1)} = S_{r+2}S_{r+4}$; $g_2^{(r+1)} = S_{r+3}S_{r+4}$
        if ($\hat{\Delta}_0^{(r)} \neq 0$ and $k^{(r)} \ge 0$)
5).         $B^{(r+1)}(x) = \Lambda^{(r)}(x)$
6).         $\hat{\Theta}^{(r+1)}(x) = \hat{\Delta}^{(r)}(x)/x + S_{r+1}\Lambda^{(r)}(x)$
7).         $\theta_S^{(r+1)} = S_{r+3}\hat{\Delta}_1^{(r)} + g_1^{(r)}\Lambda_0^{(r)}$
8).         $\gamma^{(r+1)} = \hat{\Delta}_0^{(r)}$; $\gamma_S^{(r+1)} = \hat{\Delta}_0^{(r)} S_{r+2}$
9).         $h_1^{(r+1)} = \hat{\Delta}_0^{(r)} S_{r+3}$; $h_2^{(r+1)} = \hat{\Delta}_0^{(r)} g_2^{(r)}$
10).        $k^{(r+1)} = -k^{(r)} - 1$
        else
11).        $B^{(r+1)}(x) = xB^{(r)}(x)$
12).        $\hat{\Theta}^{(r+1)}(x) = \hat{\Theta}^{(r)}(x) + S_{r+1}xB^{(r)}(x)$
13).        $\theta_S^{(r+1)} = S_{r+3}\hat{\Theta}_0^{(r)}$
14).        $\gamma^{(r+1)} = \gamma^{(r)}$; $\gamma_S^{(r+1)} = \gamma^{(r)} S_{r+2}$
15).        $h_1^{(r+1)} = \gamma^{(r)} S_{r+3}$; $h_2^{(r+1)} = \gamma^{(r)} g_2^{(r)}$
16).        $k^{(r+1)} = k^{(r)} + 1$
```

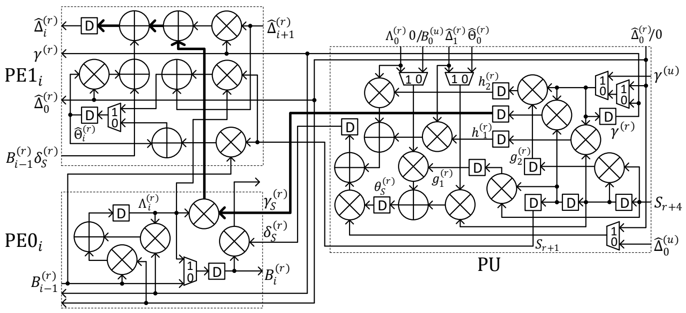


Fig. 5. PE and PU architectures for fast nested KES.

The details of the i-th PEs and the PU are illustrated in Fig. 5. The syndromes are available in advance and $S_u, S_{u+1}, \cdots$ are sent into the PU serially starting three clock cycles ahead of the nested KES. In the initialization clock cycle of the nested KES, the polynomial coefficients $\Lambda_i^{(u)}$ and $B_{i+1}^{(u)}$ are loaded into the registers in PE0_i, and $\hat{\Delta}_{i-1}^{(u)}$ and $\hat{\Theta}_i^{(u)}$ are loaded into the registers in PE1_i. The polynomial and scalar initialization are carried out by routing $\gamma^{(u)}$ and $\hat{\Delta}_0^{(u)}$ through the two multiplexers on the right side of PU and setting the registers in the PU according to the initialization lines of Algorithm 2. The other multiplexers in the PU and PEs are controlled by the 'c' signal, which is '1' when $\hat{\Delta}_0^{(r)} \neq 0$ and $k^{(r)} \ge 0$ and is '0' otherwise. At the beginning of the r-th iteration of the nested KES, $\gamma^{(r)}$, $\hat{\Delta}_0^{(r)}$, $\gamma_S^{(r)}$, and $\delta_S^{(r)}$ are available in the PU. They are sent to the PEs to compute $\hat{\Delta}_i^{(r+1)}$, $\hat{\Theta}_i^{(r+1)}$, $\Lambda_i^{(r+1)}$ and $B_i^{(r+1)}$ according to Lines 1, 2, 5, 6, 11, and 12 of Algorithm 2. At the same time, $\gamma^{(r+1)}$, $\gamma_S^{(r+1)}$, and $\delta_S^{(r+1)}$ are generated in the PU according to Lines 3, 4, 7-9, and 13-15 of Algorithm 2 to be used in iteration $r+1$. In the PU architecture, the signals $g_1^{(r)}$, $g_2^{(r)}$, $h_1^{(r)}$, and $h_2^{(r)}$ are labeled so that the correspondence between the architecture and Algorithm 2 can be easily traced. The critical path of the fast nested KES architecture has one multiplier and two adders as highlighted by the thicker wires in Fig. 5. It is only one adder longer compared to the critical path of the slowed-down and re-timed nested KES architecture in [9] and is the same as that of the design in [10]. When there is only one sub-codeword that requires nested decoding, which happens in most cases, the proposed architecture reduces the nested KES latency to a half in terms of the number of clock cycles, since no dummy zero needs to be inserted.

IV. Scaled Fast Nested KES Algorithm

The fast nested KES architecture developed in the previous section reduces the critical path by around a half to one multiplier and two adders without applying the slow-down technique. However, it has 8 multipliers in each pair of PEs instead of 4 as in the scaled nested KES design [10]. Finite field multipliers are area-consuming. Borrowing the idea from [10], the number of multipliers can be reduced by utilizing scaled polynomials to enable the sharing of intermediate results among polynomial updating. However, this inevitably makes the scalars more complicated and consist of more components. Different from the scaling scheme in [10], the scalars need to be pre-computed and the data path can only have one multiplier to achieve fast KES. The data dependency and the more complicated scalars make the scalar pre-computation very challenging.

This section proposes a scaled fast nested KES algorithm that reduces the number of multipliers in each pair of PEs from 8 to 4 with only one more gate in the critical path. Different from the scaling scheme in [10], our new scaling scheme was developed to not only enable product term sharing but also allow the combined scalars to be pre-computed with one multiplier in the critical path. Alternative scaling strategies and modified scalars are developed to enable such a scaled algorithm. In the following, subsection A) and B) present the scaling of the polynomials in two categories since different approaches are utilized to keep the critical path short. Subsection C) presents the scaled fast nested KES algorithm and necessary adjustments to handle zero scalars and enable the application of the HK formula. The implementation architectures are detailed in Subsection D).

A. Scaling by Syndromes

It can be observed that the sum of $\gamma_S^{(r)}\Lambda^{(r)}(x)$ and $\delta_S^{(r)}xB^{(r)}(x)$ in Line 2 of Algorithm 2 is $S_{r+1}\Lambda^{(r+1)}(x)$. Hence, if the scaled $\Lambda'^{(r+1)}(x) = S_{r+1}\Lambda^{(r+1)}(x)$ is used, the two multipliers in each PE0 for computing the products in Line 1 of Algorithm 2 can be eliminated. To keep the decoding results the same, every polynomial needs to be multiplied by the same overall scalar. The approach adopted in [10] is to multiply additional scalars to the polynomials to make up for the difference and these additional scalar multiplications are put off until the next iteration to save multipliers. As a result, a scalar in [10] for iteration r is combined from three components: i) the scalar introduced to enable product term sharing for iteration r; ii) the additional scalar needed to make the overall scaling the same for every polynomial from iteration $r-1$; iii) the scalar that was originally in the algorithm for iteration r. The data dependency already made the scalar pre-computation complicated as shown in Section III. Adding more components to the combined scalars will make the calculations even more difficult. Fortunately, all computations on the syndromes, include inverse, can start in advance and do not pose bottlenecks on the critical path. Therefore, to simplify the scalar pre-computation, S_{r+1}^{-1} is multiplied back to $\Lambda(x)$ in the next iteration to reverse the scaling. Since $B(x)$ and $\Lambda(x)$ are linearly combined in the polynomial updating, $B(x)$ needs to be scaled by the same factor as $\Lambda(x)$. Hence, $B'^{(r+1)}(x) = S_{r+1}B^{(r+1)}(x)$ is utilized and the inverse scalar S_{r+1}^{-1} is multiplied back to $B(x)$ in the next iteration. Since the overall scalars for $\Lambda(x)$ and $B(x)$ are '1', no additional scalars need to be multiplied to $\hat{\Theta}(x)$ and $\hat{\Delta}(x)$, whose combined scalar computations are the most complicated.

From the above discussions, the polynomials can be updated as follows.

$$\Lambda'^{(r+1)}(x) = (\gamma_S^{(r)}S_r^{-1})\Lambda'^{(r)}(x) + (\delta_S^{(r)}S_r^{-1})xB'^{(r)}(x) \quad (4)$$

$$\begin{aligned}\hat{\Delta}^{(r+1)}(x) = \gamma^{(r)}\hat{\Delta}^{(r)}(x)/x + (\gamma_S^{(r)}S_r^{-1})\Lambda'^{(r)}(x) \\ + \hat{\Delta}_0^{(r)}\hat{\Theta}^{(r)}(x) + (\delta_S^{(r)}S_r^{-1})xB'^{(r)}(x)\end{aligned} \quad (5)$$

TABLE I
SCALARS FOR POLYNOMIALS IN THE FAST NESTED KES

	$\hat{\Delta}'^{(r+1)}(x)$	$\hat{\Theta}'^{(r+1)}(x)$	$\Lambda'^{(r+1)}(x)$	$B'^{(r+1)}(x)$
scalars introduced to enable term sharing	1	$\gamma^{(r)}$ or $\hat{\Delta}_0^{(r)}$	S_{r+1}	$\gamma^{(r)}S_{r+1}$ or $\hat{\Delta}_0^{(r)}S_{r+1}$
additional scalars	$\gamma^{(r)}$ or $\hat{\Delta}_0^{(r)}$	1	$\gamma^{(r)}S_{r+1}^{-1}$ or $\hat{\Delta}_0^{(r)}S_{r+1}^{-1}$	S_{r+1}^{-1}

If $\hat{\Delta}_0^{(r)} \neq 0$ and $k^{(r)} \geq 0$

$$\begin{aligned}\hat{\Theta}^{(r+1)}(x) &= \hat{\Delta}^{(r)}(x)/x + (S_{r+1}S_r^{-1})\Lambda'^{(r)}(x) \\ B'^{(r+1)}(x) &= (S_{r+1}S_r^{-1})\Lambda'^{(r)}(x)\end{aligned} \quad (6)$$

else

$$\begin{aligned}\hat{\Theta}^{(r+1)}(x) &= \hat{\Theta}^{(r)}(x) + (S_{r+1}S_r^{-1})xB'^{(r)}(x) \\ B'^{(r+1)}(x) &= (S_{r+1}S_r^{-1})xB'^{(r)}(x)\end{aligned} \quad (7)$$

To keep one multiplier in the critical path, the combined scalars $(\gamma_S^{(r)}S_r^{-1})$ and $(\delta_S^{(r)}S_r^{-1})$ in (4) and (5) need to be computed before iteration r starts. $(\gamma_S^{(r)}S_r^{-1}) = \gamma^{(r)}(S_{r+1}S_r^{-1})$ and can be computed in iteration $r-1$ with one multiplier in the data path as either $\hat{\Delta}_0^{(r-1)}(S_{r+1}S_r^{-1})$ or $\gamma^{(r-1)}(S_{r+1}S_r^{-1})$, since $(S_{r+1}S_r^{-1})$ can be made available before this iteration. Also $(\delta_S^{(r)}S_r^{-1}) = \hat{\Delta}_0^{(r)}(S_{r+1}S_r^{-1})$ and it is calculated in a similar way as $\hat{\Delta}_0^{(r)}S_{r+1}$ as explained in Section III. There are 6 different product terms in (4), (5), (6), and (7). Therefore, using these formulas, the number of multipliers in each pair of PEs can be reduced from 8 as in the fast nested KES architecture presented in Section III to 6.

B. Scaling by Discrepancy Coefficients

To further reduce the complexity of the PEs, more polynomials need to be computed by sharing product terms. Note that $\gamma_S^{(r)} = \gamma^{(r)}S_{r+1}$ and $\delta_S^{(r)} = \hat{\Delta}_0^{(r)}S_{r+1}$. Hence the $\hat{\Theta}^{(r+1)}(x)$ in (6) scaled by $\gamma^{(r)}$ and that in (7) scaled by $\hat{\Delta}_0^{(r)}$ equal the sums of the first two and last two terms, respectively, on the right side of (5). Similarly, the $B'^{(r+1)}(x)$ in (6) and (7) do not need extra product terms to compute if it is scaled by $\gamma^{(r)}$ and $\hat{\Delta}_0^{(r)}$, respectively. If these scaled polynomials are used, then each pair of PEs would only need 4 multipliers. As explained previously, the overall scalar for every polynomial needs to be the same. However, multiplying back the inverse of $\gamma^{(r)}$ or $\hat{\Delta}_0^{(r)}$ in the next iteration as in the approach of Subsection A) will lead to long critical path. This is because that the inversions of $\gamma^{(r)}$ or $\hat{\Delta}_0^{(r)}$ cannot start early and get pipelined as those of the syndromes. Therefore, $\gamma^{(r)}$ or $\hat{\Delta}_0^{(r)}$ needs to be multiplied to the other polynomials as additional scalars to make up for the difference. The scalars introduced to enable product term sharing and the additional scalars for making the overall scaling the same are listed in the first and second rows of Table I, respectively.

To differentiate the notations, '$'$' is added to all polynomials and related scalars in the following for the scaled algorithm. As mentioned previously, the additional scalars for iteration $r-1$ are combined with the scalars needed to enable product term sharing for iteration r and those already in the polynomial updating formulas for iteration r. Take the scalar of $\Lambda'(x)$ as an example. If (6) was executed in every previous iteration, it can be derived that the combined scalar for $\Lambda'^{(r)}(x)$ on the right side of (5) accumulated over the iterations is $(\gamma_S'^{(r)} S_r^{-1})\gamma'^{(r-1)} \cdots \gamma'^{(u)} = \gamma'^{(r)}\gamma'^{(r-1)} \cdots \gamma'^{(u)} S_{r+1}S_r^{-1}$. To keep one multiplier in the critical path, the multiplication of $\gamma'^{(r)}, \gamma'^{(r-1)}, \cdots$ should be taken care of one at a time in iteration $r-1, r-2, \cdots$. This means that $\gamma'^{(u)} S_{r+1}S_r^{-1}$ should be computed at initialization. When there are $w-u$ higher-order syndromes and hence $w-u$ iterations in the nested KES, $w-u$ such values need to be computed at the initialization. Moreover, they are multiplied by $\gamma'^{(u+1)}, \gamma'^{(u+2)} \cdots$ over the iterations until the combined scalar is consumed in iteration r. This initialization and updating bring large area overhead. Such an issue does not exist in the design of [10] since it has two multipliers in the critical path. One is used to multiply the latest γ and the other is used to multiply the syndrome.

To simplify the combined scalar computation, we propose to drop $\gamma'^{(r-2)} \cdots \gamma'^{(u)}$ and use $\rho_S^{(r)} = \gamma'^{(r)}\gamma'^{(r-1)} S_{r+1}S_r^{-1}$ as the accumulated scalar of $\Lambda'^{(r)}(x)$ for iteration r. $\gamma'^{(r)}$ and $\gamma'^{(r-1)}$ cannot be further eliminated because they are the original scalar for iteration r and additional scalar from iteration $r-1$, respectively. Define $c_u^{r-2} = \gamma'^{(r-2)} \cdots \gamma'^{(u)}$. If (7) was executed in iteration $i < r$, then the $\gamma'^{(i)}$ in c_u^{r-2} is replaced by $\hat{\Delta}_0'^{(i)}$ and c_u^{r-2} can still be eliminated from the scalar. Besides, following Lines 8 and 14 of Algorithm 2, it can be derived that $\gamma'^{(r)}\gamma'^{(r-1)}$ always equals to $\hat{\Delta}_0'^{(r-1)}\gamma'^{(r-1)}$ no matter whether (6) or (7) was executed. Hence

$$\rho_S^{(r)} = \hat{\Delta}_0'^{(r-1)}\gamma'^{(r-1)} S_{r+1}S_r^{-1}. \tag{8}$$

To keep the decoding results unchanged, the four terms on the right side of (5) for $\hat{\Delta}'(x)$ updating should be scaled by the same factor. Hence, c_u^{r-2} also needs to be eliminated from the other three scalars. This also ensures that the other polynomial updatings are done correctly since the other formulas are partial results from (5). From (6), the difference between the scalars for $\hat{\Delta}'^{(r)}(x)/x$ and $\Lambda'^{(r)}(x)$ is $S_{r+1}S_r^{-1}$. Hence, if c_u^{r-2} is removed from the scalar of $\Lambda'^{(r)}(x)$, the modified scalar for $\hat{\Delta}'^{(r)}(x)/x$ should be

$$\rho^{(r)} = \hat{\Delta}_0'^{(r-1)}\gamma'^{(r-1)}. \tag{9}$$

The additional scalar for $\hat{\Theta}'(x)$ is always '1' as shown in Table I. From (5), it can be derived that $\hat{\Delta}_0'^{(r)} = \gamma'^{(r-1)}\gamma'^{(r-2)} \cdots \gamma'^{(u)}\hat{\Delta}_1'^{(r-1)} + \gamma'^{(r-1)}\gamma'^{(r-2)} \cdots \gamma'^{(u)} S_r S_{r-1}^{-1}\Lambda_0'^{(r-1)} + \hat{\Delta}_0'^{(r-1)}\hat{\Theta}_0'^{(r-1)}$ if (6) was executed in every previous iteration. Define $\phi^{(r-1)} = \hat{\Theta}_0'^{(r-1)}/(\gamma'^{(r-2)} \cdots \gamma'^{(u)})$. Then the scalar for $\hat{\Theta}'^{(r)}(x)$ after c_u^{r-2} is eliminated is

$$\xi^{(r)} = \gamma'^{(r-1)}\hat{\Delta}_1'^{(r-1)} + \gamma'^{(r-1)} S_r S_{r-1}^{-1}\Lambda_0'^{(r-1)} + \hat{\Delta}_0'^{(r-1)}\phi^{(r-1)} \tag{10}$$

From (7), the scalars for $\hat{\Theta}'^{(r)}(x)$ and $xB'^{(r)}(x)$ are different by a factor of $S_{r+1}S_r^{-1}$. Hence, the modified scalar for $xB'^{(r)}(x)$ should be

$$\begin{aligned}\xi_S^{(r)} &= \xi^{(r)} S_{r+1}S_r^{-1} \\ &= \gamma'^{(r-1)}(S_{r+1}S_r^{-1})\hat{\Delta}_1'^{(r-1)} + \gamma'^{(r-1)} S_{r+1}S_{r-1}^{-1}\Lambda_0'^{(r-1)} \\ &\qquad + \hat{\Delta}_0'^{(r-1)}(S_{r+1}S_r^{-1})\phi^{(r-1)}\end{aligned} \tag{11}$$

In the case that (7) was executed in some previous iterations, similar analyses apply and the same scalar formulas as in (10) and (11) are derived.

All the four modified combined scalars in (8), (9), (10), and (11) for $\Lambda'^{(r)}(x)$, $\hat{\Delta}'^{(r)}(x)/x$, $\hat{\Theta}'^{(r)}(x)$ and $xB'^{(r)}(x)$, respectively, can be pre-computed before iteration r with one multiplier in the critical path. Specifically, for (8), $\gamma'^{(r-1)} S_{r+1}S_r^{-1}$ is computed as either $\gamma'^{(r-2)}(S_{r+1}S_r^{-1})$ or $\hat{\Delta}_0'^{(r-2)}(S_{r+1}S_r^{-1})$ in iteration $r-2$ with one multiplier in the data path. Then the multiplication with $\hat{\Delta}_0'^{(r-1)}$ is completed in iteration $r-1$. The computation of (9) is even simpler. Besides, both (10) and (11) are in a similar format as (3). Hence, they can be also computed in advance with one multiplier in the critical path as explained in Subsection A).

C. Scaled Fast Nested KES Algorithm

Algorithm 3: Scaled Fast Nested KES Algorithm

```
input: $\Lambda'^{(u)}(x)$, $B'^{(u)}(x)$, $\hat{\Delta}'^{(u)}(x)$, $\hat{\Theta}'^{(u)}(x)$, $\gamma'^{(u)}$, $k^{(u)}$, $\rho^{(u)}$,
       $\xi^{(u)}$, $\lambda^{(u)}$, $\phi^{(u)}$, $\beta^{(u)}$, $\zeta^{(u)}$, $\eta^{(u)}$ from previous KES;
       $S_i$ $(u \le i < w)$; $S_{w+i} = 0$ $(0 \le i \le 4)$
       set $S'_i = S_i$ if $S_i \ne 0$, set $S'_i = 1$ otherwise $(u \le i \le w+4)$
initialization:
$\hat{\Delta}'^{(u)}(x) = \hat{\Delta}'^{(u)}(x) + S_u\Lambda'^{(u)}(x)$
$\hat{\Theta}'^{(u)}(x) = \hat{\Theta}'^{(u)}(x) + S_u B'^{(u)}(x)$
$\Lambda'^{(u)}(x) = S'_u\Lambda'^{(u)}(x)$
$B'^{(u)}(x) = S'_u B'^{(u)}(x)$
execute Subroutine 1
for $r = u, u+1, \cdots, w-1$
1).   if ($\hat{\Delta}_0'^{(r)} = 0$) set $\rho^{(r)} = 1$, $\rho_S^{(r)} = g_1^{(r)}$, $\xi^{(r)} = \xi_S^{(r)} = 0$
2).   $\Lambda'^{(r+1)}(x) = \rho_S^{(r)}\Lambda'^{(r)}(x) + \xi_S^{(r)} xB'^{(r)}(x)$
3).   $\hat{\Delta}'^{(r+1)}(x) = \rho^{(r)}\hat{\Delta}'^{(r)}(x)/x + \rho_S^{(r)}\Lambda'^{(r)}(x)$
                 $+\xi^{(r)}\hat{\Theta}'^{(r)}(x) + \xi_S^{(r)} xB'^{(r)}(x)$
                 (set $\rho_S^{(r)} = \xi_S^{(r)} = 0$ if $S_{r+1} = 0$)
4).   execute Subroutine 2
      if ($\hat{\Delta}_0'^{(r)} \ne 0$ and $k^{(r)} \ge 0$)
5).       $B'^{(r+1)}(x) = \rho_S^{(r)}\Lambda'^{(r)}(x)$
6).       $\hat{\Theta}'^{(r+1)}(x) = \rho^{(r)}\hat{\Delta}'^{(r)}(x)/x + \rho_S^{(r)}\Lambda'^{(r)}(x)$
                 (set $\rho_S^{(r)} = 0$ if $S_{r+1} = 0$)
7).       execute Subroutine 3
8).       $k^{(r+1)} = -k^{(r)} - 1$
      else
9).       if ($\hat{\Delta}_0'^{(r)} = 0$) set $\xi^{(r)} = 1$, $\xi_S^{(r)} = g_1^{(r)}$
10).      $B'^{(r+1)}(x) = \xi_S^{(r)} xB'^{(r)}(x)$
11).      $\hat{\Theta}'^{(r+1)}(x) = \xi^{(r)}\hat{\Theta}'^{(r)}(x) + \xi_S^{(r)} xB'^{(r)}(x)$
                 (set $\xi_S^{(r)} = 0$ if $S_{r+1} = 0$)
12).      execute Subroutine 4
13).      $k^{(r+1)} = k^{(r)} + 1$
```

Applying the scaling schemes discussed in the above two subsections, the proposed scaled fast nested KES algorithm is developed as shown in Algorithm 3. The initialization and updating of the scalars are detailed in the four subroutines.

```
Subroutine 1: Scalar initialization
1). $g_1^{(u)} = S'_{u+1}S'^{-1}_u$; $g_2^{(u)} = S'_{u+2}S'^{-1}_{u+1}$; $g_3^{(u)} = S'_{u+3}S'^{-1}_{u+1}$
    $g_4^{(u)} = S'_{u+3}S'^{-1}_{u+2}$; $g_5^{(u)} = S'_{u+4}S'^{-1}_{u+3}$; $g_6^{(u)} = g_4^{(u)}g_1^{(u)}$
2). $h_1^{(u)} = \gamma'^{(u)}S'_{u+1}S'^{-1}_u$; $h_2^{(u)} = \gamma'^{(u)}S'_{u+2}S'^{-1}_u$
    $h_3^{(u)} = \gamma'^{(u)}S'_{u+2}S'^{-1}_{u+1}$
3). $\rho_S^{(u)} = g_1^{(u)}\rho^{(u)}$
4). $\xi_S^{(u)} = g_1^{(u)}\xi^{(u)} + g_1^{(u)}S_u\lambda^{(u)}$
5). $\xi^{(u)} = \xi^{(u)} + S_u\lambda^{(u)}$
6). $\phi_S^{(u)} = g_2^{(u)}\phi^{(u)} + g_2^{(u)}S_u\beta^{(u)}$
7). $\phi^{(u)} = \phi^{(u)} + S_u\beta^{(u)}$
```

```
Subroutine 2: Scalar updating
1). $\rho^{(r+1)} = \begin{cases}\gamma'^{(r)}\hat{\Delta}_0'^{(r)} & \text{if } (\hat{\Delta}_0'^{(r)} \neq 0)\\ \gamma'^{(r)}\zeta^{(r)} & \text{otherwise}\end{cases}$
2). $\rho_S^{(r+1)} = \begin{cases}h_3^{(r)}\hat{\Delta}_0'^{(r)} & \text{if } (\hat{\Delta}_0'^{(r)} \neq 0)\\ h_3^{(r)}\zeta^{(r)} & \text{otherwise}\end{cases}$
3). $\xi^{(r+1)} = \begin{cases}\gamma'^{(r)}\hat{\Delta}_1'^{(r)} + h_1^{(r)}\Lambda_0'^{(r)} + \hat{\Delta}_0'^{(r)}\phi^{(r)} & \text{if } (S_{r+1} \neq 0)\\ \gamma'^{(r)}\hat{\Delta}_1'^{(r)} + \hat{\Delta}_0'^{(r)}\phi^{(r)} & \text{otherwise}\end{cases}$
4). $\xi_S^{(r+1)} = \begin{cases}h_3^{(r)}\hat{\Delta}_1'^{(r)} + h_2^{(r)}\Lambda_0'^{(r)} + \hat{\Delta}_0'^{(r)}\phi_S^{(r)} & \text{if } (S_{r+1} \neq 0)\\ h_3^{(r)}\hat{\Delta}_1'^{(r)} + \hat{\Delta}_0'^{(r)}\phi_S^{(r)} & \text{otherwise}\end{cases}$
5). $\zeta^{(r+1)} = \begin{cases}\rho^{(r)}\hat{\Delta}_0'^{(r)} & \text{if } (\hat{\Delta}_0'^{(r)} \neq 0)\\ \zeta^{(r)} & \text{otherwise}\end{cases}$
6). $\lambda^{(r+1)} = h_1^{(r)}\Lambda_0'^{(r)}$
7). $g_1^{(r+1)} = g_2^{(r)}$; $g_2^{(r+1)} = g_4^{(r)}$; $g_3^{(r+1)} = S'_{r+4}S'^{-1}_{r+2}$
    $g_4^{(r+1)} = g_5^{(r)}$; $g_5^{(r+1)} = S'_{r+5}S'^{-1}_{r+4}$; $g_6^{(r+1)} = g_5^{(r)}g_2^{(r)}$
```

```
Subroutine 3: Scalar updating in 'if'
1). $\gamma'^{(r+1)} = \hat{\Delta}_0'^{(r)}$
2). $h_1^{(r+1)} = \hat{\Delta}_0'^{(r)}g_2^{(r)}$; $h_2^{(r+1)} = \hat{\Delta}_0'^{(r)}g_3^{(r)}$; $h_3^{(r+1)} = \hat{\Delta}_0'^{(r)}g_4^{(r)}$
3). $\phi^{(r+1)} = \begin{cases}\hat{\Delta}_1'^{(r)} + g_1^{(r)}\Lambda_0'^{(r)} & \text{if } (S_{r+1} \neq 0)\\ \hat{\Delta}_1'^{(r)} & \text{otherwise}\end{cases}$
4). $\phi_S^{(r+1)} = \begin{cases}g_4^{(r)}\hat{\Delta}_1'^{(r)} + g_6^{(r)}\Lambda_0'^{(r)} & \text{if } (S_{r+1} \neq 0)\\ g_4^{(r)}\hat{\Delta}_1'^{(r)} & \text{otherwise}\end{cases}$
5). $\beta^{(r+1)} = g_1^{(r)}\Lambda_0'^{(r)}$
6). $\eta^{(r+1)} = \rho^{(r)}\hat{\Delta}_0'^{(r)}$
```

```
Subroutine 4: Scalar updating in 'else'
1). $\gamma'^{(r+1)} = \gamma'^{(r)}$
2). $h_1^{(r+1)} = \gamma'^{(r)}g_2^{(r)}$; $h_2^{(r+1)} = \gamma'^{(r)}g_3^{(r)}$; $h_3^{(r+1)} = \gamma'^{(r)}g_4^{(r)}$
3). $\phi^{(r+1)} = \phi^{(r)}$
4). $\phi_S^{(r+1)} = g_4^{(r)}\phi^{(r)}$
5). $\beta^{(r+1)} = 0$
6). $\eta^{(r+1)} = \xi^{(r)}\eta^{(r)}$
```

In Algorithm 3, ρ, ξ, λ, ϕ, and β are scaled versions of γ, $\hat{\Delta}_0$, Λ_0, $\hat{\Theta}_0$, and B_0 of Algorithm 2, respectively. ζ is used to adjust the polynomial updating scalars when the scalars introduced for product term sharing are zero. η is needed to enable error magnitude computation using the HK formula. More details about these two variables are provided later in this section. To show that all computations can be broken down to one multiplication at a time, $\phi_S^{(r)} = S_{r+2}S_{r+1}^{-1}\phi^{(r)}$, $g_i^{(r)}$ $(1 \leq i \leq 6)$, and $h_i^{(r)}$ $(1 \leq i \leq 3)$ are used to denote intermediate values in Algorithm 3. Essentially, $g_1^{(r)}$, $g_2^{(r)}$, $g_3^{(r)}$, $g_4^{(r)}$, $g_5^{(r)}$, and $g_6^{(r)}$ equal $S_r^{-1}S_{r+1}$, $S_{r+1}^{-1}S_{r+2}$, $S_{r+1}^{-1}S_{r+3}$, $S_{r+2}^{-1}S_{r+3}$, $S_{r+3}^{-1}S_{r+4}$, and $S_r^{-1}S_{r+1}S_{r+2}^{-1}S_{r+3}$, respectively, and $h_1^{(r)}$, $h_2^{(r)}$, and $h_3^{(r)}$ are $\gamma'^{(r)}S_r^{-1}S_{r+1}$, $\gamma'^{(r)}S_r^{-1}S_{r+2}$, and $\gamma'^{(r)}S_{r+1}^{-1}S_{r+2}$, respectively.

It can be observed that there are only four different polynomial product terms in Algorithm 3. As a result, the number of multipliers in each pair of PEs is reduced to 4 from 8 as needed by the fast nested KES algorithm (Algorithm 2). Although the scalars need to be computed and updated according to the many lines in the four subroutines, the calculations are done over individual coefficients instead of polynomials. Hence, the overall area requirement is reduced substantially.

Algorithm 3 can also start from the KES results of either the sub-codeword decoding or the previous nested decoding round. In the former case, $\Lambda'^{(u)}(x)$, $B'^{(u)}(x)$, $\hat{\Delta}'^{(u)}(x)$, $\hat{\Theta}'^{(u)}(x)$, and $\gamma'^{(u)}$ are set to $\Lambda^{(u)}(x)$, $B^{(u)}(x)$, $\hat{\Delta}^{(u)}(x)$, $\hat{\Theta}^{(u)}(x)$, and $\gamma^{(u)}$, respectively, from the sub-codeword decoding. $\rho^{(u)}$, $\xi^{(u)}$, $\lambda^{(u)}$, $\phi^{(u)}$, $\beta^{(u)}$, $\zeta^{(u)}$ and $\eta^{(u)}$ are set to $\gamma^{(u)}$, $\hat{\Delta}_0^{(u)}$, $\Lambda_0^{(u)}$, $\hat{\Theta}_0^{(u)}$, $B_0^{(u)}$, $\gamma^{(u)}$, and $\gamma^{(u)}$, respectively. For a later round of the nested KES, the inputs of Algorithm 3 are just the outputs of the previous round. S_{w+i} $(0 \leq i \leq 4)$ are not part of the higher-order syndromes that are available. They are set to 0 in Algorithm 3 to make the formulas uniform for each iteration.

As shown in Table I, $\gamma^{(r)}$, syndromes, and discrepancy coefficients are introduced as scalars of polynomials in our scaled fast nested KES algorithm. Following Algorithm 1, it can be easily derived that $\gamma^{(r)}$ is never zero. However, syndromes and discrepancy coefficients may be zero, in which case the corresponding scalars need to be adjusted to eliminate their effects on the polynomial updating. If a syndrome is not part of the original scalar in Algorithm 2, then it should be replaced by '1' in the corresponding lines of Algorithm 3 when it is zero. For example, S_{r+1} is not a scalar in Lines 1, 5, and 11 of Algorithm 2. Hence, if $S_{r+1} = 0$, then it should be replaced by '1' in the $\rho_S^{(r)}$ and $\xi_S^{(r)}$ scalars in Lines 2, 5, and 10 of Algorithm 3. The replacements are done through the four subroutines. On the other hand, the terms $\rho_S^{(r)}\Lambda'^{(r)}(x)$ and $\xi_S^{(r)}xB'^{(r)}(x)$ in Lines 3, 6, and 11 of Algorithm 3 should be zero when $S_{r+1} = 0$, since the computations in Lines 2, 6, and 12 of Algorithm 2 originally have the factor S_{r+1}.

When $\hat{\Delta}_0'^{(r)} = 0$, $\xi^{(r)}$ and $\xi_S^{(r)}$ in Lines 2 and 3 of Algorithm 3 should be zero since $\hat{\Delta}_0'^{(r)}$ is originally part of the corresponding scalars in Lines 1 and 2 of Algorithm 2. Similarly, if $\hat{\Delta}_0'^{(r)}$ is not originally part of a polynomial scalar, such as the $\xi^{(r)}$ and $\xi_S^{(r)}$ in Lines 10 and 11 of Algorithm 3, then the scalar should be adjusted. However, the other factors of $\xi^{(r)}$ and $\xi_S^{(r)}$ are not computed, and the adjusted scalars cannot be derived by setting $\hat{\Delta}_0'^{(r)}$ to '1' as in the adjustments of the scalars involving S_{r+1}. Instead, the adjusted scalars can be decided according to the requirement that all the product terms involved in a linear combination, such as that in Line 3 of Algorithm 3, should be scaled by the same factor in each iteration. Since $\xi^{(r)} = \xi_S^{(r)} = 0$ when $\hat{\Delta}_0'^{(r)} = 0$, and $\rho_S^{(r)} = S_{r+1}S_r^{-1}\rho^{(r)}$, this requirement can be satisfied by replacing the $\rho^{(r)}$ and $\rho_S^{(r)}$ in Lines 2 and 3 of Algorithm 3 with '1' and $g_1^{(r)} = S_{r+1}S_r^{-1}$, respectively. Similarly, the $\xi^{(r)}$ and $\xi_S^{(r)}$ in Lines 10 and 11 of Algorithm 3 are also replaced by '1' and $g_1^{(r)}$, respectively.

$\hat{\Delta}_0'^{(r)}$ is a factor of $\rho^{(r+1)}$ and $\rho_S^{(r+1)}$ for iteration $r+1$. These two scalars also need to be adjusted when $\hat{\Delta}_0'^{(r)} = 0$. $\xi^{(r+1)}$ and $\xi_S^{(r+1)}$ can be computed according to (10) and (11). Then $\rho^{(r+1)}$ and $\rho_S^{(r+1)}$ can be derived by satisfying the requirement that all product terms involved in the linear combination of Line 3 of Algorithm 3 should have the same factor. From (10) and the fact that $\hat{\Delta}'^{(r)}(x)$ and $S_r^{-1}\Lambda'^{(r)}(x)$ have the same scalar, it can be derived that $\xi^{(r+1)}/\hat{\Delta}_0^{(r+1)} = \gamma'^{(r)}\hat{\Delta}_0'^{(r)}/\hat{\Delta}_0^{(r)}$ holds. From Line 3 of Algorithm 3 and Line 3 of Subroutine 2, it can be derived that $\hat{\Delta}_0'^{(r)} = \rho^{(r-1)}/\gamma'^{(r-1)}\xi^{(r)}$. Accordingly, $(\xi^{(r+1)}/\hat{\Delta}_0^{(r+1)}) = \gamma'^{(r)}\rho^{(r-1)}/\gamma'^{(r-1)}(\xi^{(r)}/\hat{\Delta}_0^{(r)})$. To make the four product terms in Line 3 of Algorithm 3 scaled by the same factor, $(\rho^{(r+1)}/\gamma^{(r+1)}) = \gamma'^{(r)}\rho^{(r-1)}/\gamma'^{(r-1)}(\rho^{(r)}/\gamma^{(r)})$ should be satisfied. $\gamma^{(r+1)} = \gamma^{(r)}$ when $\hat{\Delta}_0'^{(r)} = 0$. Hence, the requirement is reduced to $\rho^{(r+1)} = \gamma'^{(r)}\rho^{(r-1)}/\gamma'^{(r-1)}\rho^{(r)}$. From (9), $\rho^{(r)} = \gamma'^{(r-1)}\hat{\Delta}_0'^{(r-1)}$. Hence $\rho^{(r+1)} = \gamma'^{(r)}\rho^{(r-1)}\hat{\Delta}_0'^{(r-1)}$. Both $\rho^{(r-1)}$ and $\hat{\Delta}_0'^{(r-1)}$ are available at iteration $r-1$. Hence their product can be computed at iteration $r-1$. Then $\gamma'^{(r)}$ is multiplied at iteration r. These computations are completed with only one multiplier in the critical path.

In the case that $\hat{\Delta}_0'^{(r+i)}$ is also '0' for $i = 1, 2, \cdots, \kappa$, it can be easily derived that $\rho^{(r+i+1)} = \gamma'^{(r+i)}\rho^{(r-1)}\hat{\Delta}_0'^{(r-1)}$. Define $\zeta^{(r)} = \rho^{(r-1)}\hat{\Delta}_0'^{(r-1)}$. Then $\rho^{(r+i+1)} = \gamma'^{(r+i)}\zeta^{(r)}$ for $i = 1, 2, \cdots, \kappa$, and it can be always computed with one multiplier in the critical path. Similarly, $\rho_S^{(r+i+1)}$ is computed as $S_{r+i+2}S_{r+i+1}^{-1}\rho^{(r+i+1)}$. These updatings are summarized in Lines 1, 2, and 5 of Subroutine 2.

Traditionally, the error magnitudes are calculated using $\Lambda(x)$ and $\hat{\Delta}(x)$ [7] according to the Forney's formula. The error magnitudes can be also computed by using $\Lambda(x)$ and $B(x)$ according to the HK formula. It also was discovered in [11] that the higher coefficients of $\hat{\Delta}(x)$ and $\hat{\Theta}(x)$ can be eliminated and hence the number of PEs for updating these two polynomials in the nested KES architecture can be reduced by a significant portion with negligible degradation on the error-correcting performance. However, due to the scalings of the polynomials in our scaled fast nested KES algorithm, further adjustments are necessary in order to enable the application of the HK formula.

For t-error-correcting decoding, the pair of $\Lambda(x)$ and $B(x)$ in the iteration that the length of $B(x)$ reaches t can be used to limit the length of $B(x)$ [17]. Assume this happens in iteration $r = K-1$. For unscaled algorithms, the HK formula for computing the error magnitude of the j-th position is

$$Y_j = (\gamma^{(K)}\Lambda_0^{(K)}\alpha^{-jK})/(B^{(K)}(\alpha^{-j})\Lambda_{odd}^{(K)}(\alpha^{-j})), \qquad (12)$$

where $\Lambda_{odd}(x)$ is the polynomial consisting of the odd-degree terms of $\Lambda(x)$. Λ_0 and γ are for the normalization of $\Lambda_{odd}(x)$ and $B(x)$, respectively. In our scaled fast nested KES algorithm, $\Lambda_0'^{(r)}$ and $\Lambda'_{odd}(x)$ are $\Lambda_0^{(r)}$ and $\Lambda_{odd}(x)$, respectively, scaled by the same factors. Hence $\Lambda'_{odd}(x)$ and $\Lambda_0'^{(r)}$ can be directly used in (12). However, $B'^{(r)}(x)$ is not always scaled by the same factor as $\gamma'^{(r)}$ in our scaled algorithm and its normalization needs to be reformulated.

In our scaled algorithm, $\hat{\Delta}'^{(r)}(x)$ has the same scalar as $S_r^{-1}\Lambda'^{(r)}(x)$. If Line 5 of Algorithm 3 is executed in iteration r, $B'^{(r+1)}(x) = \rho^{(r)}S_{r+1}(S_r^{-1}\Lambda'^{(r)}(x))$. The normalization coefficient of $B'^{(r+1)}(x)$ should be the constant coefficient of $B'^{(r+1)}(x)S(x)/x^r = \rho^{(r)}S_{r+1}(S_r^{-1}\Lambda'^{(r)}(x))S(x)/x^r$, where $S(x) = \sum_{j=0}^{r} S_j x^j$. Therefore, the coefficient for normalizing $B'^{(r+1)}(x)$ should be $\rho^{(r)}S_{r+1}\hat{\Delta}_0'^{(r)}$. In the case that Line 10 is executed in iteration r, $B'^{(r+1)}(x) = \xi_S^{(r)}xB'^{(r)}(x) = \xi^{(r)}S_{r+1}S_r^{-1}xB'^{(r)}(x)$. Hence, the normalization coefficient in this case is the normalization coefficient for iteration $r-1$ scaled by $\xi^{(r)}S_{r+1}S_r^{-1}$. Let $r^* < r$ be the last iteration that Line 5 was executed. Then the normalization coefficient for $B'^{(r+1)}(x)$ is $(\xi^{(r)}S_{r+1}S_r^{-1}) \times (\xi^{(r-1)}S_rS_{r-1}^{-1}) \times \cdots \times (\xi^{(r^*+1)}S_{r^*+2}S_{r^*+1}^{-1}) \times (\rho^{(r^*)}S_{r^*+1}\hat{\Delta}_0'^{(r^*)}) = S_{r+1}\xi^{(r)}\xi^{(r-1)}\cdots\xi^{(r^*+1)}\rho^{(r^*)}\hat{\Delta}_0'^{(r^*)}$. Let $\eta^{(r^*+1)} = \rho^{(r^*)}\hat{\Delta}_0'^{(r^*)}$. Then the normalization coefficient can be computed iteratively through using $\eta^{(r+1)} = \xi^{(r)}\eta^{(r)}$. These computations are carried out according to Line 6 of Subroutine 3 and 4, and S_{r+1} is multiplied at the end. In summary, if our scaled fast nested KES algorithm is adopted, the HK formula should be adjusted to

$$Y_j = (S_K\eta^{(K)}\Lambda_0'^{(K)}\alpha^{-jK})/(B'^{(K)}(\alpha^{-j})\Lambda_{odd}'^{(K)}(\alpha^{-j})). \qquad (13)$$

Similar to the zero scalar adjustment, if $S_K = 0$, then it is replaced by '1' in the above formula.

D. Scaled Fast Nested KES Architecture

The overall architecture of the scaled fast nested KES algorithm is very similar to that in Fig. 4. It consists of $t_v + 1$ pairs of PEs and one PU. The architectures of the PEs and PU for the scaled algorithm are illustrated in Fig. 6(a) and (b), respectively. PE1s update $\hat{\Delta}'(x)$ and $\hat{\Theta}'(x)$ and PE0s handle $\Lambda'(x)$ and $B'(x)$ according to Algorithm 3. The scalar initialization and updating in the four subroutines are carried out in the PU. Using these architectures, the nested KES also takes $w - u + 1$ clock cycles: one clock cycle is spent on the initialization and each iteration takes one clock cycle.

The syndromes are sent in serially to the PU four clock cycles ahead of the nested KES so that the computations on the syndromes in Subroutine 1 can start early. In the initialization clock cycle of the nested KES, the input polynomials are loaded into the registers of the PEs. $\lambda^{(u)}$, $\xi^{(u)}$, and $\gamma'^{(u)}$ are routed through the three multiplexers in the PU and the registers in the PU are set to accomplish the scalar and polynomial initialization according to Subroutine 1 and Algorithm 3. At the beginning of iteration r, the combined scalars $\rho^{(r)}$, $\rho_S^{(r)}$, $\xi^{(r)}$, and $\xi_S^{(r)}$ computed by the PU are sent to the PEs to update the polynomials according to the formulas in Lines 2, 3, 5, 6, 10, and 11 of Algorithm 3. Simultaneously, the combined scalars for the next iteration are generated in the PU based on

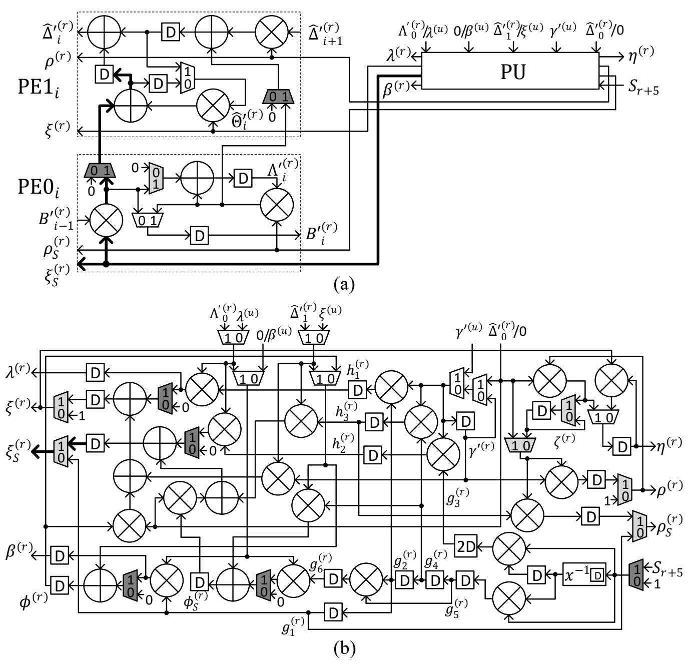


Fig. 6. Architectures for scaled fast nested KES: (a) PEs; (b) PU.

the polynomial coefficients, $\Lambda_0'^{(r)}$, $\hat{\Delta}_0'^{(r)}$, and $\hat{\Delta}_1'^{(r)}$, sent from PE0_0, PE1_0, and PE1_1, respectively. The $g_i^{(r)}$ $(1 \le i \le 6)$ signals as labeled in Fig. 6(b) are updated according to Line 7 of Subroutine 2. $h_1^{(r+1)}$, $h_2^{(r+1)}$, and $h_3^{(r+1)}$ are derived by multiplying $\hat{\Delta}_0'^{(r)}$ or $\gamma'^{(r)}$ to $g_2^{(r)}$, $g_3^{(r)}$, and $g_4^{(r)}$, respectively. The combined scalars are computed according to Subroutines 2, 3, and 4. Pipelining is applied to the finite field inverter in the bottom right corner of the PU so that its data path is no longer than that of the rest of the architecture. In the PE and PU architectures, the control signals of the white multiplexers, except the three multiplexers in the PU used for initialization, are set to '1' or '0' when the condition ($\hat{\Delta}_0'^{(r)} \neq 0$ & $k^{(r)} \ge 0$) is true or false, respectively. The gray multiplexers are used to adjust the scalars when S_{r+1} or $\hat{\Delta}_0'^{(r)}$ is '0'.

S_{r+1} is not part of the original scalars in Lines 1, 5, and 11 of Algorithm 2, which correspond to the $\rho_S^{(r)}$ and $\xi_S^{(r)}$ scalars in Lines 2, 5, and 10 of Algorithm 3. Hence, when $S_{r+1} = 0$, it is substituted by '1' in the $\rho_S^{(r)}$ and $\xi_S^{(r)}$ computations through the dark gray multiplexer in the bottom right corner of the PU architecture. On the other hand, S_{r+1} is a component of the original scalars corresponding to the $\rho_S^{(r)}$ and $\xi_S^{(r)}$ in Lines 3, 6, and 11 of Algorithm 3. Therefore, the corresponding product terms are eliminated from the polynomial updating by passing '0' through the dark gray multiplexers in the PEs of Fig. 6(a). S_{r+1} is also a part of $h_1^{(r)}$, $h_2^{(r)}$, $g_1^{(r)}$, and $g_6^{(r)}$. Similarly, when it is '0', the corresponding terms in Lines 3 and 4 of Subroutine 2 and the same lines of Subroutine 3 are eliminated by passing '0' through the dark gray multiplexers in the PU.

Since $\hat{\Delta}_0'^{(r)}$ is originally part of the scalars corresponding to the $\xi^{(r)}$ and $\xi_S^{(r)}$ in Lines 2 and 3 of Algorithm 3, the product terms multiplied by these two scalars should be excluded from the polynomial updating when $\hat{\Delta}_0'^{(r)} = 0$. For the polynomial updating in Line 2, this is done by passing '0' through the light gray multiplexer in PE0. For the updating in Line 3, the two product terms are excluded by setting the leftmost register in PE1 to '0' to save a multiplexer. The computations of $\xi^{(r)}$ and $\xi_S^{(r)}$ for Lines 10 and 11, as well as the calculations of the scalars that are dependent on whether $\hat{\Delta}_0'^{(r)} = 0$, are taken care of by using the light gray multiplexers in the PU architecture.

It can be observed that the number of multipliers is reduced by a half from 8 as in the pair of nested KES PEs in Fig. 5 to 4 in Fig. 6(a). Although the PU for the scaled algorithm in Fig. 6(b) has more multipliers compared to that in Fig. 5, only one PU is needed. Therefore, substantial area reduction is achieved. The critical path of the architectures in Fig. 6 is highlighted by the thicker wires. It consists of one multiplier and three adders/multiplexers and is only one multiplexer longer than that of the architecture in Fig. 5.

TABLE II
COMPLEXITIES OF NESTED KES ARCHITECTURES

	Add.	Mult.	Reg.	Mux.	Total # XORs	Crit. path # gates	Latency ave. # clks
Nested KES from [8]	429	474	90	45	52404	11	7.21
Nested KES [9]	116	174	240	58	24204	7	14.39
Scaled Nested KES [10]	116	121	274	147	20186	8	14.39
Fast Nested KES (Algorithm 2)	177	242	129	63	28732	8	7.21
Scaled Fast Nested KES (Algorithm 3)	122	135	170	161	19369	9	7.21

V. COMPLEXITY ANALYSES AND COMPARISONS

The hardware complexities of the proposed fast and scaled fast nested KES architectures are analyzed and compared with prior designs in this section using an example GII code over $GF(2^8)$ with $m = 8$, $v = 3$, $n = 255$, and error-correction capabilities $[t_0, t_1, t_2, t_3] = [13, 16, 19, 28]$. The FER of this code is plotted in Fig. 2 and this code is also used for complexity analyses in [9], [10].

The hardware complexities of our designs from architectural analysis are listed in Table II. For the fast nested KES (Algorithm 2) architecture shown in Fig. 5, each pair of PE0 and PE1 have 8 multipliers, 6 adders, 4 registers, and 2 multiplexers. The PU has 10 multipliers, 3 adders, 11 registers, and 5 multiplexers. As shown in Fig. 4, $t_v + 1 = 29$ pairs of PEs and one PU are included in the overall architecture. Each adder over $GF(2^8)$ is implemented by 8 XOR gates and each 8-bit multiplexer requires around the same area as 8 XORs to implement. An 8-bit register with preset and clear requires the area of around 24 XORs. A general multiplier over $GF(2^8)$ can be implemented by the area of 98 XOR gates with 6 gates in the critical path using composite field arithmetic [9]. Utilizing these hardware components, the logic complexity of the proposed fast nested KES architecture can be estimated as shown in Table II and its critical path has 8 gates. Since the control logic contributes to a negligible portion of the overall area, its complexity is omitted.

The scaled fast nested KES (Algorithm 3) architecture also consists of $t_v + 1$ pairs of PEs and one PU unit. However, as it is shown in Fig. 6(a), the number of multipliers in each pair of PEs is reduced by a half to 4 compared to that in the

TABLE III
SYNTHESIS RESULTS OF NESTED KES ARCHITECTURES USING TSMC $65nm$ PROCESS UNDER $T = 0.8ns$ TIMING CONSTRAINT

	Nested KES [9]	Scaled Nested KES [10]	Fast Nested KES (Alg. 2)	Scaled Fast Nested KES (Alg. 3)
area (μm^2)	90569	78385	109746	85461
power(μW)	176868	168852	177198	154636

fast nested KES PEs in Fig. 5. Also, the number of adders is reduced from 6 to 4. Although one more register and three more multiplexers are needed in each pair of PEs to adjust the polynomial scaling, they account for a small portion of the overall area. Moreover, each multiplexer with a '0' or '1' input takes only around one half the complexity of a general multiplexer since it can be implemented as bit-wise AND or OR. As a result, the complexity of the PEs is reduced significantly by the scaled fast nested KES scheme. Despite that the complexity of the PU in Fig. 6(b) is much larger than that in Fig. 5, only one PU is needed. Overall, the scaled fast nested KES architecture achieves substantial area reduction compared to the fast nested KES architecture at the cost of only one more gate in the critical path.

For comparisons, the complexities of a simplified version of the nested KES design in [8] and those from [9], [10] are also included in Table II. Instead of using m t_v-error-correcting riBM architectures shared between the sub-codeword and nested decoding as in [8], only $3(t_v - t_0)$ extra PEs for nested decoding is counted towards the complexity of the nested KES architecture from [8] in Table II. Even with this simplification, the nested KES design from [8] is more than twice larger and has much longer critical path due to the multiplier-adder trees needed for re-initialization. Slow-down and re-timing have been adopted in the nested KES architecture [9] and the scaled nested KES architecture [10] to reduce the critical paths to those listed in Table II. The application of the slow-down with a factor of two duplicates every register and requires the interleaving of two sub-codewords. Both our new scaled fast nested KES and the design in [10] have 4 multipliers in each pair of PEs. Although our scaled fast design has more multipliers in the PU, the larger area is offset by the duplicated registers in the design of [10]. As a result, our scaled fast design has similar area as that from [10]. Nevertheless, our new designs do not require the interleaving of two sub-codewords.

To further verify the complexity of the proposed designs, they are synthesized using TSMC $65nm$ process under $T = 0.8ns$ timing constraint. The total area and power consumption are listed in Table III. The ratio between the areas of the proposed fast nested KES and the scaled nested KES [10] is very similar to the ratio between the total gate numbers in Table II estimated from architectural level. The ratios between the areas of the scaled fast nested KES and other designs are larger than the ratios between the gate counts in Table II because the scaled fast nested KES has longer critical path. Compared to the fast nested KES design, the scaled fast nested KES architecture achieves 1-85461/109746=22% area reduction under the same timing constraint from the

TABLE IV
POSSIBLE NUMBERS OF SUB-CODEWORDS IN EACH ROUND OF THE NESTED KES AND CORRESPONDING PROBABILITIES AT SER=0.02

1st round	2nd round	3rd round
1 sub-CW (99.7468%)	1 sub-CW (2.67%)	1 sub-CW (1.58%)
		0 sub-CW (98.42%)
	0 sub-CW (97.33%)	
2 sub-CW (0.2528%)	2 sub-CW (0.07%)	failure (0.02%)
		1 sub-CW (3.09%)
		0 sub-CW (96.89%)
	1 sub-CW (5.21%)	1 sub-CW (1.57%)
		0 sub-CW (98.43%)
	0 sub-CW (94.72%)	
3 sub-CW (0.0004%)	failure (0.002%)	
	2 sub-CW (0.208%)	failure (0.02%)
		1 sub-CW (3.09%)
		0 sub-CW (96.89%)
	1 sub-CW (7.611%)	1 sub-CW (1.57%)
		0 sub-CW (98.43%)
	0 sub-CW (92.179%)	

TABLE V
NUMBER OF CLOCK CYCLES NEEDED IN THE NESTED KES ROUNDS

	1st round	2nd round	3rd round
	1 sub-CW	1 sub-CW	1 sub-CW
Designs in [9], [10]	4(t_1-t_0)+2	4(t_2-t_1)+2	4(t_3-t_2)+2
proposed designs	2(t_1-t_0)+1	2(t_2-t_1)+1	2(t_3-t_2)+1
	2 sub-CW	2 sub-CW	
Designs in [9], [10]	4(t_1-t_0)+2	4(t_2-t_1)+2	
proposed designs	4(t_1-t_0)+2	4(t_2-t_1)+2	
	3 sub-CW		
Designs in [9], [10]	8(t_1-t_0)+4		
proposed designs	6(t_1-t_0)+3		

synthesis reports. To evaluate the maximum achievable clock frequencies of the designs, syntheses with tighter timing constraints have also been carried out. The shortest timing constraints that can be met without negative slack are 0.57ns, 0.60ns, 0.66ns, and 0.66ns, for the designs in [9], [10], fast nested KES, and scaled fast nested KES, respectively. The minimum achievable clock period of the fast nested KES design is relatively longer than that analyzed based on the number of gates in the critical path. Possible reasons include i) the fast nested KES is larger and hence the wire delay is longer; ii) the adders in its critical path cannot be combined easily with other logic as the multiplexers in the critical paths of the scaled nested KES [10] and the proposed scaled fast nested KES architectures.

For the example GII code with $v = 3$, the nested KES will be activated when there are 1, 2, or 3 exceptional sub-codewords. Up to v nested decoding rounds are carried out. In the i-th round $(i = 1, 2, \cdots, v)$, if there are more than $v-i+1$ sub-codewords remain to be corrected, then the decoding will not be successful. In this case, decoding failure is declared and the rest decoding rounds are not carried out. Table IV lists the possible numbers of sub-codewords in each round of the nested KES. The probabilities of having each number of sub-codewords in each round at input SER=0.02 are also listed. As it is shown in this table and mentioned previously, the nested decoding is dominated by the case that there is only one exceptional sub-codeword. Besides, the exceptional sub-codewords most likely get corrected in the first nested decoding round.

The numbers of clock cycles needed to carry out the nested KES for different numbers of sub-codewords over the nested decoding rounds are listed in Table V. For both the proposed fast nested KES designs, $2(t_i - t_{i-1}) + 1$ clock cycles are needed to incorporate $2(t_i - t_{i-1})$ higher-order syndromes for each sub-codeword in the i-th nested decoding round. The additional one clock cycle is used for initialization. The designs of [9], [10] require two sub-codewords to be interleaved and two clock cycles for initialization. If there is only one sub-codeword remaining in a nested decoding round, then dummy zeros are inserted and the latency of the nested KES for one sub-codeword is the same as that for two sub-codewords in that round.

Multiplying the probabilities of each case in Table IV with the corresponding clock cycle numbers in Table V, the average latency of the nested KES can be derived as shown in the last column of Table II. Our proposed designs do not need to interleave sub-codewords and require half of the clock cycles compared to the designs in [9], [10] when there is one sub-codeword, which dominates the cases of the nested KES. As a result, our proposed designs reduce the average number of clock cycles needed in the nested KES to almost a half. Besides, the maximum achievable clock frequencies of the proposed designs are only slightly lower than those of the architectures in [10].

Essentially, the proposed designs require a smaller number of clock cycles than the designs in [9], [10] when the number of sub-codewords to be corrected in a nested decoding round is odd and have the same number of clock cycles when there are even sub-codewords remaining. Even if the number of exceptional sub-codewords is not odd in the beginning, the number of remaining sub-codewords will be odd in a later decoding round with high probability. Hence, the proposed designs also reduce the nested KES latency in most cases. Consider the worst case that there are three exceptional sub-codewords, and two and one sub-codewords remain to be corrected in the second and third nested decoding rounds, respectively. It can be calculated from Table V that the nested KES latency of such a case is 80 clock cycles using the designs from [9], [10]. Using the proposed designs, this worst-case latency is reduced to 54 clock cycles.

If $\Lambda(x)$ and $B(x)$ are used for error magnitude computation, the number of PE1s can be reduced at the cost of error-correcting performance degradation. Following the analysis in [11], it can be computed that the extra FER resulted by keeping 21 instead of 29 PE1s does not exceed 3.3×10^{-14} at input SER=0.02 for the example code. This extra FER is negligible compared to the FER of the code at this SER, which is 2.1×10^{-11} as shown in Fig. 2. This PE1-reduction scheme can be applied to both the proposed designs as well as the architectures from [9], [10]. In this case, the new scaled fast nested KES is still much smaller than the fast nested KES architecture and has similar or smaller gate counts than the designs in [9], [10] from architectural analyses.

The area saving achievable by the proposed scaled fast nested KES scheme (Algorithm 3) over the unscaled algorithm (Algorithm 2) depends on the numbers of PEs and the relative complexity of the multipliers compared to the other hardware components. The scaled algorithm reduces the complexity of each pair of PEs while having higher complexity in the single PU unit. Hence, for codes with larger error-correction capability t_v, the achievable area reduction is more significant, since the PU would account for a smaller percentage of the overall area. The complexity of finite field multipliers generally increases faster with field order compared to the complexities of the other arithmetic units. Besides, the multipliers account for a smaller portion of the overall area in the scaled fast nested KES architecture. Therefore, for codes constructed over larger fields, the scaled fast nested KES architecture achieves more significant area saving over the unscaled architecture. Additionally, the increase of one gate in the critical path accounts for a smaller percentage since the data path of higher-order finite field multipliers is longer. Although the percentage of cases with only one exceptional sub-codeword changes with the input SER as shown in Fig. 2, it is very high for the SER in practical range. Accordingly, both of the proposed fast designs achieve close to 50% average latency reduction in terms of the number of clock cycles compared to prior architectures for various SERs.

VI. Conclusions

Two fast nested KES algorithms and architectures are proposed in this paper. Novel algorithmic reformulations are developed to pre-compute the scalars simultaneously with the polynomial updating in the nested KES with one multiplier in the critical path. As a result, the latency of our proposed KES architectures is only about a half most of the time compared to that of the previous designs. Our second design keeps one multiplier in the critical path and substantially reduces the area requirement. The area reduction is achieved by developing new scaling schemes to enable product term sharing and novel methods to pre-compute the combined scalars. Additionally, non-trivial adjustments have been proposed to eliminate the effects of the introduced zero scalars and enable the application of alternative error magnitude computation formula for further area reduction. Future work will study the simplifications of the other components of GII decoders.

References

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

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

[3] Y. Wu, “Generalized integrated interleaving BCH codes,” *Proc. IEEE Intl. Symp. Info. Theory*, pp. 1098-1102, Barcelona, Spain, Jul. 2016.

[4] S. B. Wicker and V. K. Bhargava, Ed. *Reed-Solomon Codes and Their Applications*, IEEE Press, 1994.

[5] X. Zhang, “Generalized three-layer integrated interleaved codes,” *IEEE Commun. Lett.*, vol. 22, no. 3, pp. 442-445, Mar. 2018.

[6] E. Berlekamp, *Algebraic Coding Theory*, McGraw-Hill, 1968.

[7] D. V. Sarwate and N. R. Shanbhag, “High-speed architectures for Reed-Solomon decoders,” *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 9, no. 5, pp. 641-655, Oct. 2001.

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

[9] X. Zhang and Z. Xie, “Efficient architectures for generalized integrated interleaved decoder,” *IEEE Trans. Circuits and Syst.-I: Regular Papers*, vol. 66, no. 10, pp. 4018-4031, Oct. 2019.

[10] Z. Xie and X. Zhang, “Scaled nested key equation solver for generalized integrated interleaved decoder,” *IEEE Trans. Circuits and Syst.-II*, 2019.

[11] Z. Xie and X. Zhang, “Reduced-complexity key equation solvers for generalized integrated interleaved BCH decoders,” *IEEE Trans. Circuits and Syst.-I*, in press.

[12] K. K. Parhi, *VLSI Digital Signal Processing Systems: Design and Implementation*, John Wiley & Sons, 1999.

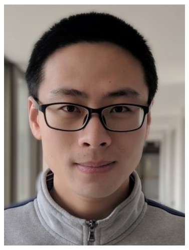
[13] R. E. Blahut, *Algebraic Codes for Data Transmission*, Cambridge, U.K.: Cambridge Univ. Press, 2003.

[14] M. Blaum, J. L. Hafner, and S. R. Hetzler, “Nested multiple erasure correcting codes for storage arrays,” U.S. Patent 8 433 979, Apr. 30, 2013.

[15] Y. Wu, “A new encoding method for integrated-interleaved codes,” *Proc. IEEE Intl. Symp. Info. Theory*, pp. 983-987, Austin, Texas, USA, Jun. 2010.

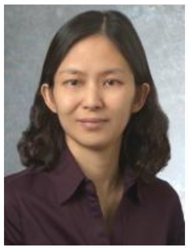
[16] X. Zhang, “Systematic encoder of generalized three-layer integrated interleaved codes,” *Proc. of IEEE Intl. Conf. on Commun.*, Shanghai, China, May 2019.

[17] Y. Wu, “New scalable decoder architectures for Reed-Solomon codes.” *IEEE Trans. Commun.*, vol. 63, no. 8, pp. 2741-2761, Aug. 2015.



Zhenshan Xie received the B.S. degree in information engineering from East China University of Science and Technology, Shanghai, China, in 2014, and the M.S. degree in communications and information system from University of Chinese Academy of Sciences, Beijing, China, in 2017. He is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, The Ohio State University, Columbus, OH, USA.

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

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