Efficient Sub-Codeword Key Equation Solver for Generalized Integrated Interleaved BCH Decoder

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

***Abstract*—Generalized integrated interleaved (GII) error-correcting codes nest sub-codewords to form codewords of more powerful codes. They can achieve hyper-speed decoding with good error correction capability. For GII codes built on BCH codes, the first decoding stage is to decode individual BCH sub-words. This stage largely determines the throughput and dominates the area of the overall decoder. Unlike that in traditional BCH decoding, longer polynomials need to be kept in the key-equation solver (KES) step of the first stage in order to continue the KES step in the second-stage nested decoding of GII codes. To take advantage of the very fast storage class memories (SCMs), GII codes with 3-error-correcting BCH sub-codewords can be utilized. This paper proposes a low-complexity and high-speed design for the KES of 3-error-correcting BCH sub-word decoding. Formulas are developed to compute the KES results directly instead of utilizing the traditional iterative process. More importantly, through analyzing the properties of the involved variables, the coefficients are scaled and reformulated to substantially reduced the complexity. Detailed hardware implementation architectures are also developed in this paper. Our design achieves three times throughput with 20.2% smaller area than the best prior design for a code over $GF(2^{10})$.**

***Index Terms*—BCH codes, Berlekamp algorithm, Generalized integrated interleaved codes, Key equation solver, Storage class memories**

I. Introduction

The new storage class memories (SCMs), such as phase change and resistive memories, have very short sensing latency, such as $100ns$. On the other hand, their error rate is much higher than that of DRAM. To realize the potential of SCMs, error-correcting codes with hyper-speed decoding and excellent correction capability are needed. Generalized integrated interleaved (GII) error-correcting codes [1], [2] that nest BCH sub-codewords to form stronger BCH codewords are among the best candidates. Due to the relatively short codeword length and low redundancy of SCMs, GII codes with 3-error-correcting BCH sub-codewords can be used to substantially increase the throughput and simplify the decoder.

GII-BCH decoding has two stages [2]. The first stage is the BCH decoding on individual sub-words, in which a key-equation solver (KES) computes an error locator polynomial, $\Lambda(x)$, using the syndromes. Only when the decoding fails or miscorrections [3] are detected on some sub-words, the second-stage nested decoding is carried out. The nested words are utilized to compute higher-order syndromes, from which the KES is carried out to correct more errors. Since the second stage is activated with low probability, it is essential to speed up and reduce the area of the first-stage sub-word decoding.

The KES for BCH decoding can be carried out using the Berlekamp algorithm [4], which has been reformulated to improve the hardware efficiency [5]–[7]. Traditionally, for a t-error-correcting BCH decoder, polynomial coefficients of degree up to t are computed in the KES implementation to reduce the complexity. However, the polynomial degrees may exceed t when there are more than t errors. For example, in 3-error-correcting BCH decoding, $\deg(\Lambda(x))$ may reach 5. To reduce the latency of the nested decoding, its KES for incorporating the higher-order syndromes should continue from the results of the KES in sub-word decoding. Reformulations have been developed to improve the efficiency of the nested KES [8], [9]. Nevertheless, to allow the continuation for nested KES, all coefficients of the polynomials instead of only those of degree up to t need to be kept in the KES of the sub-word decoding. Truncating the polynomials as in traditional BCH decoding will make the sub-words that can be corrected in the following nested decoding process uncorrectable and leads to substantial performance degradation [3].

The Peterson's algorithm [10] has lower complexity than the simplified Berlekamp algorithms [5]–[7] when the degree of $\Lambda(x)$ to compute is 3 [11]. However, it cannot compute the full-length polynomials based on which the KES can continue in the nested decoding and hence cannot be utilized for the sub-word decoding.

This paper proposes an efficient KES that computes full polynomials for 3-error-correcting GII-BCH sub-word decoding. By analyzing all possible updating of the polynomials and variables in the three iterations of the parallel inversionless Berlekamp algorithm (PIBA) [7], formulas are developed to directly derive the polynomials and variables that should be calculated at the end of the third iteration. Simplified control logic is also designed to decide which formulas to use based on the input syndromes. Additionally, by utilizing the properties of the variables and syndromes in different cases of updating, the polynomial and variable computation formulas are scaled and reformulated to substantially reduce the complexity. In addition, hardware architectures are developed to implement the proposed schemes. For an example GII code over $GF(2^{10})$, the proposed design achieves three times throughput with 20.2% smaller area than the PIBA architecture, which is the most efficient applicable existing design.

II. GII Decoding and KES Algorithm

Assume that $\mathcal{C}_v \subseteq \mathcal{C}_{v-1} \subseteq \cdots \subseteq \mathcal{C}_1 \subset \mathcal{C}_0$ are $v+1$ BCH codes of length n over $GF(2^q)$ with error correction capabilities $t_v \geq t_{v-1} \geq \cdots \geq t_1 > t_0$, respectively. A GII-BCH $[m,v]$ code can be defined as [2]

$$\mathcal{C} \triangleq \Big\{ [c_0(x), c_1(x), \cdots, c_{m-1}(x)] : c_i(x) \in \mathcal{C}_0, \\ \tilde{c}_l(x) = \sum_{i=0}^{m-1} \alpha^{il}(x) c_i(x) \in \mathcal{C}_{v-l}, 0 \leq l < v \Big\}, \tag{1}$$

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

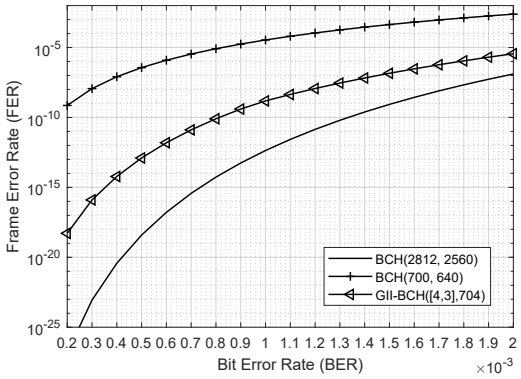

Fig. 1. FERs of BCH and GII-BCH [4,3] codes over binary symmetric channel

where α is a primitive element of $GF(2^q)$ and $\alpha^{il}(x)$ is the standard basis polynomial representation of α^{il}. $c_i(x)\in\mathcal{C}_0$ $(0\le i<m)$ and $\tilde{c}_l(x)\in C_{v-l}$ $(0\le l<v)$ are referred to as the sub-codewords and nested codewords, respectively.

GII-BCH decoding consists of two stages. First, t_0-error-correcting BCH decoding is carried out for each received sub-word $y_i(x)=c_i(x)+e_i(x)$, where $e_i(x)$ is the error polynomial. $2t_0$ syndromes are computed as $S_j^{(i)}=y_i(\alpha^{j+1})$ $(0\le j<2t_0)$. If some syndromes are nonzero, KES is carried out to find an error locator polynomial $\Lambda(x)$ using the $2t_0$ syndromes and the Chien search follows to find the roots, which are the inverse error locations.

Only when some sub-words fail the first-stage decoding or are miscorrected, the second-stage nested decoding is carried out to correct more errors. $2t$ syndromes are required to correct t errors. Let $\tilde{y}_l(x)=\sum_{i=0}^{m-1}\alpha^{il}(x)y_i(x)$ and $i_0, i_1, \cdots, i_{b-1}$ be the indices of the $b\le v$ sub-words that need extra error correction. Since the nested codeword $\tilde{c}_l(x)$ $(0\le l<b)$ is at least t_1-error-correcting, nested syndromes with order $2t_0\le j<2t_1$ can be computed as $\tilde{S}_j^{(l)}=\tilde{y}_l(\alpha^{j+1})$. According to (1), syndromes with order $2t_0\le j<2t_1$ for each of those sub-words with extra errors can be computed as

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

where A is a $b\times b$ matrix and $A_{k,w}=\alpha^{i_w k(j+1)}$ [2]. Once the higher-order syndromes are calculated, the KES follows to correct up to t_1 errors in each of those sub-words. If there are b' sub-words that remain to be corrected, then syndromes with order $2t_1\le j<2t_2$ are computed for each of them by using the higher-order nested syndromes in a similar way. This process is repeated for up to v rounds.

The frame error rates (FERs) of GII codes can be computed using formulas [2]. Consider codes that protect 2560-bit data with 256-bit parity for SCMs. The best FER of GII [4,3] codes with these overall codeword length and redundancy is achieved when $[t_0, t_1, t_2, t_3] = [3, 5, 6, 11]$. Its sub-codewords are (2560+256)/4=704-bit long. As shown in Fig. 1, it achieves 4-5 orders of magnitude lower FER compared to the (700, 640) BCH code whose rate is the same and length is similar to the sub-codewords. Although its FER is higher than that of the long (2812, 2560) BCH code, its decoder can achieve much higher throughput with lower complexity.

The KES algorithm of the lowest hardware complexity for general binary BCH decoding is the PIBA in Algorithm 1. The algorithm in [6] is only different in the initialization. The PIBA needs t iterations for t-error-correcting decoding and $\Lambda(x)$ is updated iteratively using an auxilliary polynomial $B(x)$. L_Λ and L_B are the lengths of $\Lambda(x)$ and $B(x)$, respectively. $\hat{\Delta}^{(r)}(x)=\Lambda^{(r)}(x)S(x)/x^{2r}$ and $\hat{\Theta}^{(r)}(x)=B^{(r)}(x)S(x)/x^{2r}$. 'even' is added as a subscript of these two polynomials since only their even coefficients are needed. The constant coefficient of $\hat{\Delta}^{(r)}(x)$, denoted by $\hat{\Delta}_0^{(r)}$, is referred to as the discrepancy coefficient of iteration r.

Algorithm 1: Parallel Inversionless Berlekamp Alg.

Input: syndromes S_i $(0 \le i < 2t)$
Initialization:
$\Lambda^{(0)}(x)=1$; $B^{(0)}(x)=x^{-1}$; $L_\Lambda^{(0)}=0$; $L_B^{(0)}=-1$; $\gamma^{(0)}=1$
$\hat{\Delta}_{even}^{(0)}(x)=S_0+S_2x^2+\cdots+S_{2t-2}x^{2t-2}$
$\hat{\Theta}_{even}^{(0)}(x)=S_1+S_3x^2+\cdots+S_{2t-1}x^{2t-2}$
for $r=0,1,\cdots,t-1$
1 $\Lambda^{(r+1)}(x)=\gamma^{(r)}\Lambda^{(r)}(x)+\hat{\Delta}_0^{(r)}x^2B^{(r)}(x)$
2 $\hat{\Delta}_{even}^{(r+1)}(x)=\gamma^{(r)}\hat{\Delta}_{even}^{(r)}(x)/x^2+\hat{\Delta}_0^{(r)}\hat{\Theta}_{even}^{(r)}(x)$
3 if ($\hat{\Delta}_0^{(r)}\neq 0$ and $L_B^{(r)}-L_\Lambda^{(r)}\ge -1$)
4 $B^{(r+1)}(x)=\Lambda^{(r)}(x)$; $\hat{\Theta}_{even}^{(r+1)}(x)=\hat{\Delta}_{even}^{(r)}(x)/x^2$
5 $\gamma^{(r+1)}=\hat{\Delta}_0^{(r)}$; $L_B^{(r+1)}=L_\Lambda^{(r)}$; $L_\Lambda^{(r+1)}=L_B^{(r)}+2$
6 else
7 $B^{(r+1)}(x)=x^2B^{(r)}(x)$; $\hat{\Theta}_{even}^{(r+1)}(x)=\hat{\Theta}_{even}^{(r)}(x)$
8 $\gamma^{(r+1)}=\gamma^{(r)}$; $L_B^{(r+1)}=L_B^{(r)}+2$; $L_\Lambda^{(r+1)}=L_\Lambda^{(r)}$

Compared to the KES in the sub-word decoding, the KES in the nested decoding just needs to handle more syndromes. Reformulations have been developed to efficiently incorporate higher-order syndromes into the results of Algorithm 1 [8], [9]. Accordingly, the nested KES can start from the results of the sub-word KES to substantially reduce the latency. The degree of the polynomials in Algorithm 1 may exceed t when there are more than t errors. Different from classic BCH decoding that only needs the coefficients of $\Lambda(x)$ with degree up to t from the KES, all the four polynomials, $\Lambda(x)$, $B(x)$, $\hat{\Delta}(x)$, and $\hat{\Theta}(x)$, with full length need to be computed from the KES of sub-word decoding in order to continue the nested KES. Therefore, the Peterson's algorithm cannot be utilized to simplify the KES in GII sub-word decoding even if t_0 is small, such as three.

III. Efficient KES for GII-BCH Sub-Word Decoding

In this section, a simplified method is proposed to carry out the KES in the sub-word decoding with t_0=3 for GII-BCH codes. Algorithm 1 only requires 3 iterations in the case of t_0=3 and the polynomials and variables can be updated in two different ways in each iteration. By analyzing all possible combinations of the updating in the three iterations, formulas are developed to directly derive the polynomials and variables that should be calculated at the end of the third iteration of Algorithm 1. Simplified control logic is also designed to decide which formula to use based on the syndromes. Additionally, by utilizing the properties of the discrepancy coefficients, variables, and syndromes in different cases of updating, the polynomial and variable computation formulas are scaled and reformulated to substantially reduce the complexity.

TABLE I
PATTERNS OF DISCREPANCY COEFFICIENTS AND PROPERTIES OF SYNDROMES FOR DIFFERENT EXECUTIONS OF ALGORITHM 1 FOR $t = 3$

branch vector	discrepancy coefficient pattern	$L_\Lambda^{(3)}$	$L_B^{(3)}$	properties
000	$\hat{\Delta}_0^{(0)}\neq0$, $\hat{\Delta}_0^{(1)}\neq0$, $\hat{\Delta}_0^{(2)}=\phi$, $\phi\neq0$	3	2	$S_0\neq0$
001	$\hat{\Delta}_0^{(0)}\neq0$, $\hat{\Delta}_0^{(1)}\neq0$, $\hat{\Delta}_0^{(2)}=\phi$, $\phi=0$	2	3	$S_0^6+S_2^2=S_0S_4+S_0^3S_2$
010	$\hat{\Delta}_0^{(0)}\neq0$, $\hat{\Delta}_0^{(1)}=0$, $\hat{\Delta}_0^{(2)}=\phi$, $\phi\neq0$	4	1	$S_0\neq0$, $S_2=S_0^3$
011	$\hat{\Delta}_0^{(0)}\neq0$, $\hat{\Delta}_0^{(1)}=0$, $\hat{\Delta}_0^{(2)}=\phi$, $\phi=0$	1	4	$S_0\neq0$, $S_4+S_0^2S_2=0$
101	$\hat{\Delta}_0^{(0)}=0$, $\hat{\Delta}_0^{(1)}\neq0$, $\hat{\Delta}_0^{(2)}=S_4$, $\phi\neq0$	3	2	$S_0=0$, $\hat{\Delta}_0^{(1)}=S_2$, $\phi=S_2^2$
110	$\hat{\Delta}_0^{(0)}=0$, $\hat{\Delta}_0^{(1)}=0$, $\hat{\Delta}_0^{(2)}=S_4$, $\phi=0$	5	0	$S_0=0$, $S_2=0$, $S_4\neq0$

A. Direct KES with $t = 3$

In each iteration of Algorithm 1, the polynomial and variable updating can be executed in two different ways according to Line 4-5 and 7-8. They are referred to as branch A and B, respectively, of the execution. A 3-bit vector, b=$[b_0b_1b_2]$, is used to represent the branches in the three iterations. If b_i='0', it means that branch A is executed in iteration r=i in Algorithm 1. Otherwise, branch B is executed in that iteration.

Table I lists all possible values of the branch vector. From initialization of Algorithm 1, $\hat{\Delta}_0^{(0)}$=S_0. Also $\hat{\Delta}^{(1)}(x)$=$\gamma^{(0)}\hat{\Delta}^{(0)}(x)/x^2$+$\hat{\Delta}_0^{(0)}\hat{\Theta}^{(0)}(x)$ according to Line 2 and hence $\hat{\Delta}_0^{(1)}$=S_2+S_0S_1. Similarly, tracing computations to iteration r=1, it can be derived that $\hat{\Delta}_0^{(2)}$ equals S_0^6+S_2^2+S_0S_4+$S_0^3S_2$, denoted by ϕ, or S_4 if branch A or branch B, respectively, is executed in iteration r=0. The KES is not carried out when all syndromes are zero, in which case $\hat{\Delta}_0^{(0)}$=$\hat{\Delta}_0^{(1)}$=$\hat{\Delta}_0^{(2)}$=0. Hence, the branch vector, b, can never be '111'. In addition, if branch B and branch A are executed in iteration r=0 and 1, respectively, it can be derived that $L_B^{(2)}$=0 and $L_\Lambda^{(2)}$=3. As a result, branch A will not be executed in iteration r=2. Hence, b cannot be '100' either. Therefore, a total of 6 possible values for the branch vector are listed in Table I.

Starting from the initial values of $L_\Lambda^{(0)}$ and $L_B^{(0)}$, by tracing the updating in branch A and B over the three iterations, the values of $L_\Lambda^{(3)}$ and $L_B^{(3)}$ can be derived as listed in Table I. The pattern of the discrepancy coefficients can be also derived for each possible execution of Algorithm 1 as listed in Table I using the values of $L_B^{(r)}$−$L_\Lambda^{(r)}$. Initially, $L_B^{(0)}$−$L_\Lambda^{(0)}$=−1. Therefore, branch A will be executed and hence b_0='0' iff $\hat{\Delta}_0^{(0)}\neq0$. Regardless of whether branch A or B is executed in iteration r=0, $L_B^{(1)}$−$L_\Lambda^{(1)}\geq-1$ in iteration 1. Similarly, b_1='0' iff $\hat{\Delta}_0^{(1)}\neq0$. If branch A is executed in iteration 0, it can be derived that $L_B^{(2)}$−$L_\Lambda^{(2)}\geq-1$ and $\hat{\Delta}_0^{(2)}$=ϕ. In this case, b_2='0' iff $\phi\neq0$. If branch B is executed in iteration 0, it means that $\hat{\Delta}_0^{(0)}$=S_0=0. Then ϕ is simplified to S_2^2. $\hat{\Delta}_0^{(1)}$=S_2 in this case. Accordingly, ϕ=0 iff $\hat{\Delta}_0^{(1)}$=0. From Table I, it can be observed that the six possible branch conditions correspond to distinct patterns of whether $\hat{\Delta}_0^{(0)}$, $\hat{\Delta}_0^{(1)}$, and ϕ are zero, which can be decided from the syndromes directly. Hence, the branch vector can be derived by simple combinational logic taking the syndromes as the inputs.

By tracing the computations over the three iterations of Algorithm 1, formulas for computing $\Lambda^{(3)}(x)$, $\hat{\Delta}_{even}^{(3)}(x)$, $B^{(3)}(x)$, $\hat{\Theta}_{even}^{(3)}(x)$, and $\gamma^{(3)}$ for each possible combination of branch executions can be derived as listed in Table II and III. In these formulas, $\hat{\Delta}_0^{(0)}$ is directly replaced by S_0.

TABLE II
FORMULAS FOR $\Lambda^{(3)}(x)$ AND $\hat{\Delta}_{even}^{(3)}(x)$ COMPUTATION

branch vector	$\Lambda^{(3)}(x)$
000	$S_0\hat{\Delta}_0^{(1)}+S_0^2\hat{\Delta}_0^{(1)}x+(S_0S_4+S_0^3S_2)x^2+S_0\phi x^3$
001	$S_0\hat{\Delta}_0^{(1)}+S_0^2\hat{\Delta}_0^{(1)}x+(S_0^6+S_2^2)x^2$
010	$S_0^2+S_0^3x+(S_0S_4+S_0^6)x^4$
011	$S_0^2+S_0^3x$
101	$S_2+S_4x^2+S_2^2x^3$
110	$1+S_4x^5$
branch vector	$\hat{\Delta}_{even}^{(3)}(x)$
000	$S_0^2S_2^2\hat{\Delta}_0^{(1)}+(S_0S_4+S_0^3S_2)S_4+S_0^5\phi+S_0S_2^2\phi x^2$
001	$S_0^2S_2^2\hat{\Delta}_0^{(1)}+(S_0^6+S_2^2)S_4$
010	$S_0^3S_2^2+(S_0S_4+S_0^6)S_2+(S_0S_4+S_0^6)S_4x^2$
011	$S_0^3S_2^2$
101	$S_4^2+S_2^4x^2$
110	0

TABLE III
FORMULAS FOR $B^{(3)}(x)$, $\hat{\Theta}_{even}^{(3)}(x)$, AND $\gamma^{(3)}$ COMPUTATION

branch vector	$B^{(3)}(x)$	$\hat{\Theta}_{even}^{(3)}(x)$	$\gamma^{(3)}$
000	$S_0+S_0^2x+\hat{\Delta}_0^{(1)}x^2$	$S_0^2S_2^2+S_4\hat{\Delta}_0^{(1)}$	ϕ
001	$x^2+S_0x^3$	$(S_4+S_0^5)+S_0S_2^2x^2$	$\hat{\Delta}_0^{(1)}$
010	$S_0+S_0^2x$	$S_0^2S_2^2$	$S_0S_4+S_0^6$
011	x^4	$S_2+S_4x^2$	S_0
101	x^2	S_4	S_2
110	1	0	S_4

B. Low-Complexity Reformulated and Scaled Polynomial and Variable Computations

Computing the polynomials by the formulas in Table II and III requires many multipliers. To reduce the complexity, the formulas are scaled and modified in this subsection to enable intermediate results sharing without changing decoding results.

In Algorithm 1, $\hat{\Delta}^{(r)}(x)$=$\Lambda^{(r)}(x)S(x)/x^{2r}$ and $\hat{\Theta}^{(r)}(x)$=$B^{(r)}(x)S(x)/x^{2r}$. Also it can be derived that $\gamma^{(r)}$ is the constant coefficient of $B^{(r)}(x)S(x)/x^{2r-2}$. Hence scaling $\Lambda^{(r)}(x)$ and $\hat{\Delta}^{(r)}(x)$ by a nonzero factor and scaling $\hat{\Theta}^{(r)}(x)$ and $B^{(r)}(x)$ by a different nonzero factor do not affect the linear combinations in Line 1 and 2 of Algorithm 1. The roots of $\Lambda^{(r)}(x)$ are not changed and decoding results are the same.

It can be seen that $S_0\neq0$ is a common factor in each coefficient of the first $\Lambda^{(3)}(x)$ formula in Table II. Eliminating this factor, the simplified $\Lambda^{(3)}(x)$ formula for the case of b='000' is listed in Table IV. From Table I, when b='001', ϕ=S_0^6+S_2^2+S_0S_4+$S_0^3S_2$=0. Accordingly, S_0^6+S_2^2 =S_0S_4+$S_0^3S_2$. Hence, the formula for $\Lambda^{(3)}(x)$ computation for the case of b='000' can be also used to compute $\Lambda^{(3)}(x)$ when b='001'. In the case of b='101', S_0=0. Hence, $\hat{\Delta}_0^{(1)}$=S_2+S_0S_1=S_2 and ϕ reduces to S_2^2. Therefore, the same formula also computes $\Lambda^{(3)}(x)$ when b='101'. Similarly, $S_0\neq0$ can be eliminated from each coefficient of the $\Lambda^{(3)}(x)$ formula for the case of b='010' in Table II. From Table I, $\hat{\Delta}_0^{(1)}$=S_2+S_0S_1=0 in the case of b='010'. For binary BCH codes, S_1=S_0^2 and hence S_2=S_0^3. Accordingly, S_4+S_0^5=S_4+$S_0^2S_2$ and the $\Lambda^{(3)}(x)$ formula can be simplified as shown in Table IV. This formula allows S_4+$S_0^2S_2$ to be

TABLE IV
SCALED AND SIMPLIFIED FORMULAS FOR $\Lambda^{(3)}(x)$ AND $\hat{\Delta}^{(3)}_{even}(x)$ COMPUTATION

branch vector	$\Lambda^{(3)}(x)$
000, 001, 101	$\hat{\Delta}_0^{(1)} + S_0\hat{\Delta}_0^{(1)}x + (S_4 + S_0^2S_2)x^2 + \phi x^3$
010, 011	$S_0 + S_0^2x + (S_4 + S_0^2S_2)x^4$
110	$1 + S_4x^5$
branch vector	$\hat{\Delta}^{(3)}_{even}(x)$
000, 001, 101	$S_0\hat{\Delta}_0^{(1)}S_2^2 + (S_4 + S_0^2S_2)S_4 + S_0^4\phi + S_2^2\phi x^2$
010, 011	$S_0^2S_2^2 + (S_4 + S_0^2S_2)S_2 + (S_4 + S_0^2S_2)S_4x^2$
110	0

TABLE V
SCALED AND SIMPLIFIED FORMULAS FOR $B^{(3)}(x)$, $\hat{\Theta}^{(3)}_{even}(x)$, AND $\gamma^{(3)}$ COMPUTATION

branch vector	$B^{(3)}(x)$	$\hat{\Theta}^{(3)}_{even}(x)$	$\gamma^{(3)}$
000,010,101	$S_0 + S_0^2x + \hat{\Delta}_0^{(1)}x^2$	$S_0^2S_2^2 + S_4\hat{\Delta}_0^{(1)}$	ϕ
001	$S_0x^2 + S_0^2x^3$	$(S_0S_4 + S_0^6) + S_0^2S_2^2x^2$	$S_0\hat{\Delta}_0^{(1)}$
011	x^4	$S_2 + S_4x^2$	S_0
110	1	0	S_4

shared with the first formula in Table IV. When b='011', $S_0{\neq}0$ can be eliminated from each coefficient of $\Lambda^{(3)}(x)$. Besides, $\phi{=}S_0^6{+}S_2^2{+}S_0S_4{+}S_0^3S_2{=}0$ and $\hat{\Delta}_0^{(1)}{=}S_2{+}S_0^3{=}0$ in this case. Hence $S_4{+}S_0^2S_2{=}(S_2^2{+}S_0^6)/S_0{=}(\hat{\Delta}_0^{(1)})^2/S_0{=}0$. As a result, when b='011', $\Lambda^{(3)}(x)$ can be computed by the same formula as for the case of b='010' as listed in Table IV. Using similar scaling and modifications, the $\hat{\Delta}^{(3)}(x)$ formulas are simplified as shown in Table IV.

The formulas in Table III for computing $B^{(3)}(x)$, $\hat{\Theta}^{(3)}(x)$ and $\gamma^{(3)}$ can be simplified as those shown in Table V. The formulas for the case of b='000' can be also used to compute $B^{(3)}(x)$, $\hat{\Theta}^{(3)}(x)$, and $\gamma^{(3)}$ when b='010', in which case $\hat{\Delta}_0^{(1)}{=}0$ from Table I and $\phi{=}S_0S_4{+}S_0^6{+}S_2S_0^3{+}S_2^2{=}S_0S_4{+}S_0^6{+}S_2\hat{\Delta}_0^{(1)}{=}S_0S_4{+}S_0^6$. When b='101', $\hat{\Delta}_0^{(1)}{\neq}0$ as shown in Table I. Hence $\hat{\Delta}_0^{(1)}$ can be multiplied to each formula of this case listed in Table III. Also $S_0{=}0$ for this case. Hence $\hat{\Delta}_0^{(1)}{=}S_2{+}S_0^3{=}S_2$ and $\phi{=}S_0S_4{+}S_0^6{+}S_2S_0^3{+}S_2^2{=}S_2^2{=}S_2\hat{\Delta}_0^{(1)}$. Accordingly, the same formulas for the case of b='000' can be also used for the computations when b='101'. For the case of b='001', the nonzero S_0 is multiplied to each polynomial and variable in Table III in order to share intermediate results with the computation of ϕ.

Compared to those in Table II and III, the formulas in Table IV and V are greatly simplified. They lead to not only a much smaller number of multipliers but also shorter critical path in the hardware implementation.

IV. HARDWARE ARCHITECTURES AND COMPARISONS

This section presents efficient architectures for implementing the proposed 3-error-correcting KES for GII sub-word decoding. Our architecture is also compared to that of Algorithm 1, which is the most efficient among existing KES designs.

According to Table IV, the architecture in Fig. 2 is developed to compute $\Lambda^{(3)}(x)$ and $\hat{\Delta}^{(3)}(x)$. Intermediate results are shared as much as possible to reduce the area. All the three sets of formulas in Table IV are implemented and three bits $[d_0, d_1, d_2]$ are used to choose from the three results for

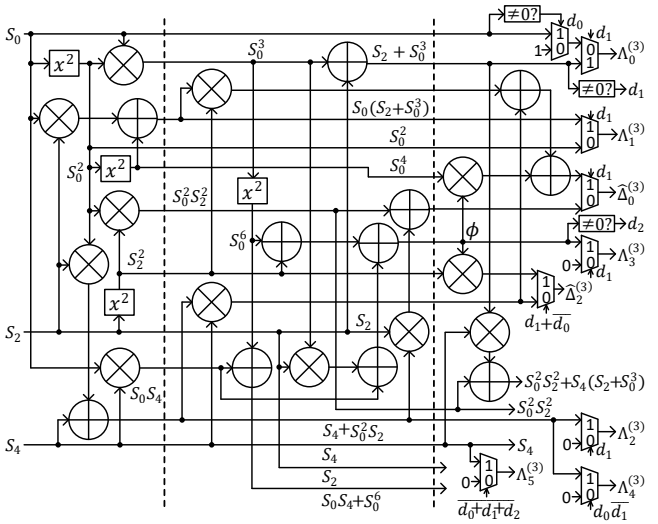

Fig. 2. $\Lambda^{(3)}(x)$ and $\hat{\Delta}^{(3)}(x)$ computation architecture

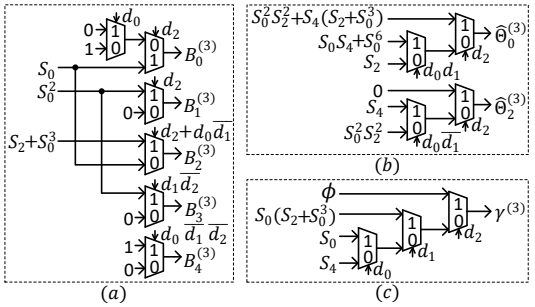

Fig. 3. Computation architectures for (a) $B^{(3)}(x)$; (b) $\hat{\Theta}^{(3)}(x)$; (c) $\gamma^{(3)}$

each polynomial coefficient. As mentioned previously, each possible branch vector corresponds to a distinct pattern of whether $\hat{\Delta}_0^{(0)}{=}S_0$, $\hat{\Delta}_0^{(1)}{=}S_2{+}S_0^3$, and ϕ are zero. $[d_0, d_1, d_2]$ are zero testing results of these three values, respectively. The architecture in Fig. 2 also computes additional intermediate results to be used in $\hat{\Theta}^{(3)}(x)$ computation.

The architectures for computing $B^{(3)}(x)$, $\hat{\Theta}^{(3)}(x)$, and $\gamma^{(3)}$ are shown in Fig. 3. All the inputs of these architectures are either the syndromes or values already computed in the architecture of Fig. 2. The architectures for deriving $L_\Lambda^{(3)}$ and $L_B^{(3)}$ are not shown in this paper. They can be derived by K-maps with $[d_0, d_1, d_2]$ as inputs and implemented by very simple combinational logic.

Our proposed architectures do not have feedback loops and are able to process the KES for one sub-word in each clock cycle. Besides, pipelining can be applied to reduce the critical path. Using normal basis representation, the squarer operation can be implemented by cyclical bit shifting [11]. When two sets of pipelining registers are inserted according to the vertical dashed lines in Fig. 2, the critical path of our architecture is reduced to one multiplier and two adders. The overall complexity of our proposed KES architecture is listed in Table VI. The registers are all used for pipelining purposes. Over $GF(2^{10})$, a normal basis multiplier can be implemented by the area of 174 XOR gates with 6 gates in the critical path. Each adder and register can be implemented with the area of 10 and 30 XOR gates, respectively [11]. Using these assumptions, the total area of the proposed design can be estimated. The simple

TABLE VI
COMPLEXITIES OF KES ARCHITECTURES FOR 3-ERROR-CORRECTING GII-BCH SUB-WORD DECODING

	Mult.	Add.	Reg.	Mux.	Sq.	Total Area (# XORs)	Crit. Path (# gates)	Latency (# clks)	# clks/ sub-word
PIBA	18	9	18	9	0	3852	7	3	3
proposed	12	11	26	22	4	3198	8	3	1

TABLE VII
SYNTHESIS RESULTS OF KES ARCHITECTURES USING TSMC $65nm$ PROCESS UNDER T=$1ns$ TIMING CONSTRAINT

	Total Area (μm^2)	Total Power (μW)
PIBA	13534	6449
proposed	10795	6116

logic for generating the multiplexer control signals, $L_{\Lambda}^{(3)}$, and $L_{B}^{(3)}$ is omitted since it accounts for a very small portion of the overall complexity.

For comparison, the complexity of the PIBA architecture [7] based on Algorithm 1 is included in Table VI. It is the simplest BCH KES architecture among existing designs and consists of $2t+1$ processing elements (PEs) for traditional t-error-correcting decoding. Each PE has two multipliers, one adder, two registers, and one multiplexer. The critical path consists of one multiplier and one adder. For 3-error-correcting sub-word decoding, $\deg(\Lambda(x))$ can be as high as 5. To keep the longer polynomial, (2×3+1)+(5-3)=9 PEs are needed.

Both the PIBA and proposed design are synthesized using TSMC $65nm$ process under T=$1ns$ timing constraint. The area and power requirements are listed in Table VII. The proposed KES architecture achieves (13534-10795)/13534=20.2% area reduction compared to the PIBA architecture for three-error-correcting BCH sub-word decoding under the same timing constraint. The area reduction is similar to the (3852-3198)/3852=17.0% analyzed from architectural level in Table VI. The proposed design also has lower power consumption as shown in Table VII. In addition, the PIBA architecture requires three clock cycles to finish the KES for each sub-word due to the iterative process. On the other hand, the proposed design is fully pipelined. After the initial three clock cycles of latency resulted from the pipelining, one sub-word is processed in each clock cycle. As a result, the proposed design also achieves three times higher throughput.

V. CONCLUSION

This paper proposes a new method to implement the KES for 3-error-correcting sub-word GII-BCH decoding. Full polynomials are kept in our KES to enable the continuation of the KES in the nested decoding. By analyzing all possible computations that can be carried out in the iterative PIBA, formulas are derived to directly compute the resulted polynomials and variables. More importantly, properties of the involved variables and syndromes are analyzed. Utilizing these properties, novel scaling and reformulations are developed to substantially simplify the polynomial and variable computation. The proposed architecture can achieve almost three times higher throughput with smaller area compared to the best previous design. Future work will study other components of GII-BCH decoders.

REFERENCES

[1] X. Tang and R. Koetter, "A novel method for combining algebraic decoding and iterative processing," in *Proc. IEEE Int. Symp. Inf. Theory*, Seattle, WA, USA, Jul. 2006, pp. 474-478.
[2] Y. Wu, "Generalized integrated interleaved codes," *IEEE Trans. Inf. Theory*, vol. 63, no. 2, pp. 1102-1119, Feb. 2017.
[3] Z. Xie and X. Zhang, "Mis-correction mitigation for generalized integrated interleaved BCH codes," *IEEE Commun. Letters*, in press.
[4] E. R. Berlekamp, *Algebraic Coding Theory*, McGraw-Hill, 1968.
[5] D. V. Sarwate and N. R. Shanbhag, "High-speed architecture for Reed-Solomon decoders," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 9, no. 5, pp. 641-655, Oct. 2001.
[6] W. Liu, J. Rho and W. Sung, "Low-power high-throughput BCH error correction VLSI design for multi-level cell NAND flash memories," *IEEE Workshop on Signal Processing Syst.*, 2006, pp. 303-308.
[7] Y. Wu, *Binary BCH decoders*, US patent, No. 8,132,08, 2012.
[8] Z. Xie and X. Zhang, "Reduced-complexity key equation solvers for generalized integrated interleaved BCH decoders," *IEEE Trans. Circuits Syst. I: Reg. Papers*, vol. 67, no. 12, pp. 5520-5529, Dec. 2020.
[9] Z. Xie and X. Zhang, "Scaled fast nested key equation solver for generalized integrated interleaved BCH decoders," in *Proc. of IEEE Int. Conf. on Acoustics, Speech, and Signal Processing*, Jun. 2021.
[10] W. W. Peterson, "Encoding and error-correction procedures for the Bose-Chaudhuri codes," *IEEE Trans. Inf. Theory*, vol. 6, no. 4, pp. 459-470, Sep. 1960.
[11] X. Zhang and Z. Wang, "A low-complexity three-error-correcting BCH decoder for optical transport network," *IEEE Trans. Circuits Syst. II: Exp. Briefs*, vol. 59, no. 10, pp. 663-667, Oct. 2012.