Modified Euclidean Algorithms for Decoding Reed-Solomon Codes

Dilip V. Sarwate

Department of Electrical and Computer Engineering
and the Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801 USA.
Email: `sarwate@illinois.edu`

Zhiyuan Yan

Department of Electrical and Computer Engineering
Lehigh University
Bethlehem, Pennsylvania 18015 USA.
Email: `yan@lehigh.edu`

Abstract—The extended Euclidean algorithm (EEA) for polynomial greatest common divisors is commonly used in solving the key equation in the decoding of Reed-Solomon (RS) codes, and more generally in BCH decoding. For this particular application, the iterations in the EEA are stopped when the degree of the remainder polynomial falls below a threshold. While determining the degree of a polynomial is a simple task for human beings, hardware implementation of this stopping rule is more complicated. This paper describes a modified version of the EEA that is specifically adapted to the RS decoding problem. This modified algorithm requires no degree computation or comparison to a threshold, and it uses a fixed number of iterations. Another advantage of this modified version is in its application to the errors-and-erasures decoding problem for RS codes where significant hardware savings can be achieved via seamless computation.

I. INTRODUCTION

Reed-Solomon (RS) codes are among the most widely used codes. Their applications range from consumer electronics such as Compact Disc (CD) and Digital Versatile Disc (DVD) players to optical communication and data storage systems. Most high-throughput RS codec architectures are based on reformulated versions [10], [18] of either the Berlekamp-Massey algorithm [2], [4] or the extended Euclidean algorithm (EEA) [3], [11], [12], [13], [15], [17]. A variable number of iterations—at most $2t$ for correcting up to t errors—are used in most EEA-based decoders. One exception is the errors-only hypersystolic Reed-Solomon decoder proposed by Berlekamp *et al.* [3] that uses exactly $2t$ iterations regardless of the number of errors. Of course, decoders based on the Berlekamp-Massey algorithm also use exactly $2t$ iterations.

In this paper, new modifications of the EEA are proposed for both errors-only (EO) decoding and errors-and-erasures (EE) decoding. First, a new modification of the EEA for errors-only decoding is proposed based on the ideas in [2] and [3]. The new algorithm also uses *exactly* $2t$ iterations, thus eliminating the degree computation and comparison in most variants of the EEA (see, for example, [1]). One particular hardware implementation (not described in this paper) of this modified algorithm turns out to be the *same* circuit as that obtained by implementing the RiBM algorithm of [10], with the difference that in one implementation the polynomials enter and leave

the circuit in ascending order of coefficients while in the other implementation the polynomials enter and leave in descending order of coefficients! The RiBM algorithm is based on the Berlekamp-Massey algorithm, and this result gives yet *another* equivalence between the Berlekamp-Massey algorithm and the EEA, different from those that have been described previously in [6] and [7].

The modified EEA proposed in this paper can be extended to errors-and-erasures decoding, and used to derive an errors-and-erasures decoding algorithm that also iterates *exactly* $2t$ times. This modified algorithm also combines the erasure-locator polynomial computation and the solution of the modified key equation in a seamless way. Hardware implementation of this algorithm eliminates not only the degree computation and comparison (see, e.g., [8]) but also the separate block used for computing the erasure-locator polynomial (see, for example, [18], [9]), thus leading to considerable savings.

II. NOTATION AND PRELIMINARIES

The codewords in a t-error-correcting cyclic Reed-Solomon code [4], [2], [16] of block length n over $GF(2^m)$ are the polynomials $C(z) = C_{n-1}z^{n-1} + C_{n-2}z^{n-2} \cdots + C_1 z + C_0$, with the property that the $2t$ successive powers $\alpha^{b_0}, \alpha^{b_0+1}, \ldots, \alpha^{b_0+2t-1}$ of α, a primitive n-th root of unity in $GF(2^m)$, are roots of $C(z)$. Here, b_0 can be any integer, but is often chosen to be 0 or 1 for ease of implementation. The code has $n - 2t$ information symbols.

A. Errors-only Decoding Algorithms for Reed-Solomon Codes

Suppose the codeword polynomial $C(z)$ is transmitted and the received word, corrupted by errors, is $R(z) = C(z) + E(z)$ where $E(z) = \sum_{i=0}^{n-1} E_i z^i$. The decoder computes the *syndromes* of the error polynomial $E(z)$: $S_j = R(\alpha^{b_0+j}) = C(\alpha^{b_0+j}) + E(\alpha^{b_0+j}) = E(\alpha^{b_0+j})$, $0 \le j < 2t$. The *syndrome polynomial* is defined as $S(z) = S_0 + S_1 z + \cdots + S_{2t-1}z^{2t-1}$. If ν errors have occurred, the error polynomial $E(z)$ can be written as $E(z) = Y_1 z^{i_1} + Y_2 z^{i_2} + \cdots + Y_\nu z^{i_\nu}$ where $Y_1, Y_2, \ldots, Y_\nu$, called the *error values*, are said to have occurred at the *error locations* $X_1 = \alpha^{i_1}, X_2 = \alpha^{i_2}, \ldots, X_\nu = \alpha^{i_\nu}$ respectively. The *error-locator polynomial* $\Lambda(z)$ of degree

ν is defined to be

$$\Lambda(z) = \prod_{j=1}^{\nu} (1 - X_j z) = 1 + \sum_{i=1}^{\nu} \Lambda_i z^i \qquad (1)$$

while the *error-evaluator polynomial* $\Omega(z)$ of degree less than ν is defined as

$$\Omega(z) = \sum_{i=1}^{\nu} Y_i X_i^{b_0} \prod_{j=1, j \neq i}^{\nu} (1 - X_j z) = \sum_{i=0}^{\nu-1} \Omega_i z^i. \qquad (2)$$

The error-locator and error-evaluator polynomials defined above are related to the syndrome polynomial by the *key equation:*

$$\Lambda(z) S(z) \equiv \Omega(z) \bmod z^{2t}. \qquad (3)$$

Note that $S(z)$ is known to the decoder, while $\Lambda(z)$ and $\Omega(z)$ are not. As the name suggests, solving the key equation for both $\Lambda(z)$ and $\Omega(z)$ is the most difficult part of the decoding process. In this paper, we focus on the EEA algorithm [4], [13] for solving the key equation.

After the key equation is solved, the errors can be corrected by finding the error locations and computing the error values. The error locations can be found via the *Chien search*: for each j, $0 \leq j \leq n-1$, the decoder tests whether or not $\Lambda(\alpha^{-j}) = 0$. If $\Lambda(\alpha^{-j}) = 0$, $E_j \neq 0$, that is, $j \in \{i_1, i_2, \cdots, i_\nu\}$. The value of the j-th transmitted symbol is computed via *Forney's formula*:

$$C_j = R_j + \left. \frac{z^{b_0} \Omega(z)}{z \Lambda'(z)} \right|_{z = \alpha^{-j}} \qquad (4)$$

where $\Lambda'(z) = \lambda_1 + 2\lambda_2 z + 3\lambda_3 z^2 + \cdots = \lambda_1 + \lambda_3 z^2 + \cdots$ is the formal derivative of $\Lambda(z)$. It is worth noting that most implementations of RS decoders compute and use $\beta\Lambda(z)$ and $\beta\Omega(z)$ where β is a nonzero scalar whose value is immaterial: $\beta\Lambda(z)$ has the same roots as $\Lambda(z)$ and so the decoder finds the same error locations, and β cancels out in (4) and so the decoder finds the same error values. Henceforth, we ignore such scalar factors in $\Lambda(z)$ and $\Omega(z)$.

B. Errors-and-Erasures Decoding of Reed-Solomon Codes

In some cases, the received words enter the decoder with some symbols specially marked as being highly unreliable and hence more likely to be in error than other symbols. These marked symbols are called erasures. For a code with minimum Hamming distance $d_{\min}$, any pattern of μ erasures and ν errors can be corrected [4] as long as $2\nu + \mu < d_{\min}$. Let $X_{1,e}, \cdots, X_{\nu,e}$ denote the (unknown) error locations and $X_{1,\epsilon}, \cdots, X_{\mu,\epsilon}$ denote the known *erasure locations*. As before, the error-locator polynomial is defined as

$$\Lambda_e(z) = \prod_{j=1}^{\nu} (1 - X_{j,e} z)$$

while the *erasure-locator polynomial* is defined to be

$$\Lambda_\epsilon(z) = \prod_{j=1}^{\mu} (1 - X_{j,\epsilon} z).$$

Note that $\Lambda_\epsilon(z)$ can be computed from the known erasure locations whereas $\Lambda_e(z)$ is unknown. Similarly, the error-evaluator polynomial is defined as

$$\Omega_e(z) = \sum_{i=1}^{\nu} Y_{i,e} X_{i,e}^{b_0} \prod_{j=1, j \neq i}^{\nu} (1 - X_{j,e} z)$$

and the *erasure-evaluator polynomial* is defined to be

$$\Omega_\epsilon(z) = \sum_{i=1}^{\mu} Y_{i,\epsilon} X_{i,\epsilon}^{b_0} \prod_{j=1, j \neq i}^{\mu} (1 - X_{j,\epsilon} z)$$

where $Y_{i,e}$ and $Y_{i,\epsilon}$ denote respectively the i-th error and *erasure values*. Also note that some of the erasure values might be zero.

If we define the *errata-locator polynomial* $\Lambda(z)$ of degree $\eta = \nu + \mu$ as $\Lambda_e(z)\Lambda_\epsilon(z)$ and the *errata-evaluator polynomial* $\Omega(z)$ as $\Lambda_e(z)\Omega_\epsilon(z) + \Lambda_\epsilon(z)\Omega_e(z)$, then the key equation (3) still holds for the errata-locator and errata-evaluator polynomials. Furthermore, the η errata locations can be obtained from the errata-locator polynomial by the Chien search and the correct values of the codeword symbols can be computed using Forney's formula. Note that the errors-only decoding is simply the special case of the errors-and-erasures decoding where $\Lambda_\epsilon(z) = 1$ and $\Omega_\epsilon(z) = 0$. As in errors-only decoders, typical implementations compute the same scalar multiple of all these polynomials, and the value of this scalar does not affect the results of any subsequent computations.

C. Structure of RS Decoders

As described above, the decoding of RS codes involve three successive stages—syndrome computation (SC), key equation solving (KES), and errata correction (EC). The implementation of syndrome computation and errata correction, which is described in, for example, [2]-[4] is generally straightforward and will not be discussed further in this paper. Here, we will focus on the implementation of key equation solving, which is the most difficult part of the decoding process.

III. MODIFIED EEA FOR ERRORS-ONLY DECODING

A. Key Equation Solution via the EEA

Sugiyama *et al.* [13] first pointed out that the extended Euclidean algorithm for computing the polynomial greatest common divisor (GCD) can be used to solve the key equation (3). The EEA, tailored to solving the key equation, can be stated as follows:

EEA for Errors-Only Decoding: The EO Algorithm

1) **Initialization:** Set $v^{(0)}(z) \leftarrow z^{2t}$, $v^{(1)}(z) \leftarrow S(z)$, $x^{(0)}(z) \leftarrow 0$, $x^{(1)}(z) \leftarrow 1$, and $j \leftarrow 1$.
2) **Iteration: While** $\deg\left[v^{(j)}(z)\right] \geq t$,
    Divide $v^{(j-1)}(z)$ by $v^{(j)}(z)$ to obtain both
    the quotient $q^{(j)}(z) \leftarrow \left\lfloor \frac{v^{(j-1)}(z)}{v^{(j)}(z)} \right\rfloor$ and the
    remainder $v^{(j+1)}(z) \leftarrow v^{(j-1)}(z) - q^{(j)}(z)v^{(j)}(z)$.
    Set $x^{(j+1)}(z) \leftarrow x^{(j-1)}(z) - q^{(j)}(z)x^{(j)}(z)$.
    Set $j \leftarrow j + 1$.
3) **Output:** $\Lambda(z) = x^{(j)}(z)$, $\Omega(z) = v^{(j)}(z)$.

Let k denote the value of j when the EO algorithm stops. Then, the outputs $x^{(k)}(z)$ and $v^{(k)}(z)$ are scalar multiples of $\Lambda(z)$ and $\Omega(z)$ as defined in (1) and (2) since $x^{(k)}(0)$ is not necessarily 1. Also, it can be shown that the polynomials $v^{(0)}(z), v^{(1)}(z), \ldots, v^{(k)}(z) = \Omega(z)$ computed by the EO algorithm have degrees d_i that form a strictly decreasing sequence with $d_0 = 2t$, $d_{k-1} = 2t - \nu$, and $d_k = \nu$.

The drawbacks to efficient implementation of the above algorithm are as follows.

- The degree $d_{j-1} - d_j$ of the quotient polynomial $q^{(j)}(z)$ can vary with j, and thus Step 2 of the EO algorithm requires a variable number of computations. This complicates the control mechanism. Furthermore, it is necessary to divide the coefficients of $v^{(j-1)}(z)$ by the leading coefficient of $v^{(j)}(z)$ in order to obtain the quotient polynomial $q^{(j)}(z)$.
- Determining the stopping condition $\deg\left[v^{(j)}(z)\right] < t$ is difficult since data needs to be gathered from many different cells in the circuit.

These two drawbacks have motivated many improvements.

B. Partial Division and Cross-Multiplication

Brent and Kung [5] proposed a systolic array implementation of the polynomial GCD algorithm in which each of the polynomial division operations involved is broken into a sequence of partial divisions, as humans often do in the "long division" method. In fact, this idea had been pointed out even earlier (see, for example, [2]). Brent and Kung also proposed using cross multiplications to avoid dividing one polynomial coefficient by another. These notions can be explained as follows. Let $U(z)$ and $V(z)$ denote polynomials of degrees r and s respectively where $r \geq s$. Then, in the "long division" of $U(z)$ by $V(z)$, the first step consists of subtracting $\frac{U_r}{V_s}z^{r-s}V(z)$ from $U(z)$ to cancel out the highest degree term in $U(z)$. If the remainder has degree at least s, a different multiple of $V(z)$ is subtracted to cancel out the highest degree term in the remainder, and so on. But,

$$\gcd(U(z), V(z)) = \gcd(U(z) - \frac{U_r}{V_s}z^{r-s}V(z), V(z)) \quad (5)$$

$$= \gcd(V_s U(z) - U_r z^{r-s}V(z), V(z)) \quad (6)$$

where (6) follows from (5) because $\gcd(A(z), B(z)) = \gcd(\beta A(z), B(z))$ for any nonzero scalar β. Thus, changing $U(z)$ to $V_s U(z) - U_r z^{r-s}V(z)$ instead of $U(z) - \frac{U_r}{V_s}z^{r-s}V(z)$ avoids a division while still zeroing out the highest degree term in $U(z)$ and still having the same GCD. Since the computation of $x^{(j+1)}(z)$ in the EO algorithm is of exactly the same form as the computation of $v^{(j+1)}(z)$, a similar calculation can be used to update these polynomials as well.

These two basic ideas have been used in different ways by many researchers to design different algorithms for GCD computation and RS decoding (see, for example, [3], [11], [12], [15], [17]). All these algorithms actually compute scalar multiples $a\Lambda(z)$ and $a\Omega(z)$ of the error-locator and error-evaluator polynomials defined in (1) and (2) respectively. Our architectures also use the ideas of Brent and Kung, but

compute $az^i\Lambda(z)$ and $az^i\Omega(z)$ where $i \geq 0$. Since the nonzero roots of $az^i\Lambda(z)$ are the same as those of $\Lambda(z)$ and the factors az^i cancel out in Forney's formula (4), such factors are inconsequential and can be ignored.

As noted before [2], [3], [5], a polynomial division can be broken up into a sequence of partial divisions for ease of implementation, and the cross-multiplication technique can be used to avoid divisions of field elements [3], [5], [11]. The same ideas can be adapted to eliminate the comparison of $\deg[v^{(j)}(z)]$ with t as well. Our modification of the EEA solves the key equation in *exactly* $2t$ steps; rather than in *at most* $2t$ steps as in previous work by others. When $\nu \leq t$ errors have occurred, our algorithm computes $z^\nu\Lambda(z)$ and $z^\nu\Omega(z)$ in 2ν steps instead of $\Lambda(z)$ and $\Omega(z)$. Our algorithm is also set up so that each of the additional $2t - 2\nu$ steps simply multiplies the results by z so that after a total of $2t$ steps, our modification of the EEA has computed $z^{2t-\nu}\Lambda(z)$ and $z^{2t-\nu}\Omega(z)$. These give the same error locations and error values as do $\Lambda(z)$ and $\Omega(z)$. The advantages to our approach are that the degree checking is avoided completely, and the key equation solution is produced with a fixed latency, both of which properties simplify the control mechanism in an implementation.

C. The Modified EEA

We claim that the following modified version of the EEA solves the key equation for RS decoding, producing polynomials $X(z) = az^{2t-\nu}\Lambda(z)$ and $V(z) = az^{2t-\nu}\Omega(z)$.

Algorithm I (Modified Euclidean Algorithm)

I.1 Initialization: $\delta \leftarrow 0$, $U(z) \leftarrow z^{2t}$, $V(z) \leftarrow S(z)$, $W(z) \leftarrow 0$, and $X(z) \leftarrow 1$.

I.2 Iteration: Repeat $2t$ times:
 a) **Set** $V(z) \leftarrow zV(z)$, $X(z) \leftarrow zX(z)$, $\delta \leftarrow \delta - 1$.
 b) **If** $V_{2t} \neq 0$ and $\delta < 0$,
  set $\delta \leftarrow -\delta$ and **swap** $U \leftrightarrow V$ and $W \leftrightarrow X$.
 c) **Set**
$$V(z) \leftarrow U_{2t}V(z) - V_{2t}U(z),$$
$$X(z) \leftarrow U_{2t}X(z) - V_{2t}W(z).$$

I.3 Output: $\Lambda(z) = X(z)$, $\Omega(z) = V(z)$, and δ.

If $\nu \leq t$ errors have occurred, then after 2ν iterations of Step I.2 in Algorithm I, $V(z) = z^{2t-d_{k-1}}v^{(j)}(z) = z^\nu\Omega(z)$ and $X(z) = z^{2t-d_{k-1}}x^{(j)}(z) = z^\nu\Lambda(z)$ where scalar factors are ignored. When Step I.2 is iterated $2t - 2\nu$ more times, $X(z)$ and $V(z)$ are multiplied by z (Step 1.2a) and the ignorable scalar factor U_{2t} (Step I.2c) each time. Hence, when Algorithm I ends, $X(z) = z^{2t-\nu}\Lambda(z)$, $V(z) = z^{2t-\nu}\Omega(z)$, and $\delta = 2\nu - 2t \leq 0$. If $\nu > t$, then Algorithm I terminates with $\delta > 0$. In practice, Steps I.2a-I.2c are not executed in succession but combined into a single calculation that computes a Boolean control variable SWAP $= (V_{2t-1} \neq 0) \wedge (\delta < 0)$ and then *simultaneously* sets

$$V(z) \leftarrow U_{2t}zV(z) - V_{2t-1}U(z),$$
$$X(z) \leftarrow U_{2t}zX(z) - V_{2t-1}W(z),$$

$$(U(z), W(z), \delta) \leftarrow \begin{cases} (zV(z), zX(z), -\delta - 1), & \text{if SWAP} = 1, \\ (U(z), W(z), \delta - 1), & \text{if SWAP} = 0. \end{cases}$$

Note also that δ must be initialized to -1 for this modified computation to work properly. We refer to this variation of Algorithm I as Algorithm I*. The following theorem summarizes the results of Algorithm I*.

Theorem 1: If $\nu \leq t$ errors have occurred, then when Algorithm I* terminates, $\delta = 2\nu - 2t - 1 < 0$ and

$$
\begin{aligned}
&(X_{2t}, \quad X_{2t-1}, \quad \ldots, \quad X_{2t-\nu}, \quad X_{2t-\nu-1} \quad \ldots, \quad X_0) \\
=\ &(\Lambda_\nu, \quad \Lambda_{\nu-1}, \quad \ldots, \quad \Lambda_0, \quad 0, \quad \ldots, \quad 0), \\
&(V_{2t}, \quad V_{2t-1}, \quad \ldots, \quad V_{2t-\nu}, \quad V_{2t-\nu-1} \quad \ldots, \quad V_0) \\
=\ &(0, \quad \Omega_{\nu-1}, \quad \ldots, \quad \Omega_0, \quad 0, \quad \ldots, \quad 0).
\end{aligned}
$$

If Algorithm I* terminates with $\delta \geq 0$, then more than t errors have occurred and the error pattern $E(z)$ is not correctable.

IV. ERRORS-AND-ERASURES DECODING

In errors-and-erasures decoding (see, for example, [14], [8], [4], [18], [9]), the key equation (3) relating the errata-locator polynomial $\Lambda(z) = \Lambda_e(z)\Lambda_\epsilon(z)$ and the errata-evaluator polynomial $\Omega(z)$ is usually solved via the following three steps executed in succession:

1. using the known erasure locations $X_{i,\epsilon}$, $1 \leq i \leq \mu$ to compute the erasure-locator polynomial $\Lambda_\epsilon(z)$ and the modified syndrome polynomial

$$\hat{S}(z) \equiv \Lambda_\epsilon(z)S(z) \bmod z^{2t},$$

2. solving the modified key equation

$$\Lambda_e(z)\hat{S}(z) \equiv \Omega(z) \bmod z^{2t}$$

for the *error*-locator polynomial $\Lambda_e(z)$ and the *errata*-evaluator polynomial $\Omega(z)$

3. multiplying $\Lambda_e(z)$ by $\Lambda_\epsilon(z)$ to obtain the errata-locator polynomial $\Lambda(z)$.

The computations of $\Lambda_\epsilon(z)$ and $\hat{S}(z)$ can be implemented as μ-iteration procedures in which initial values $\Lambda_\epsilon(z) = 1$ and $\hat{S}(z) = S(z)$ are multiplied successively by $(1 - X_{1,\epsilon}z)$, $(1 - X_{2,\epsilon}z)$, $\ldots$, $(1 - X_{\mu,\epsilon}z)$. Alternatively, $\Lambda_\epsilon(z)$ can be computed as described above and then the polynomial product $\Lambda_\epsilon(z)S(z)$ computed in $\mu + 1$ further iterations (cf. [18]). Of course, if there are no erasures, then these calculations do not need to be carried out. Next, the (modified) key equation is solved in *at most* $2t - \mu$ iterations via a slightly modified version of the extended Euclidean algorithm for errors-only decoding. A slightly modified version of the Berlekamp-Massey errors-only decoding algorithm also can be used for this purpose. Finally, the last of the three steps above is not strictly necessary, but is usually implemented (in fact, embedded into the second step) because it is more convenient to use $\Lambda(z)$ in computing errata values via Forney's formula.

It was pointed out by Blahut [4] that if the registers used to compute $\Lambda_e(z)$ are initialized to $\Lambda_\epsilon(z)$ instead of 1, then the iterations during the solution of the modified key equation produce $\Lambda(z) = \Lambda_e(z)\Lambda_\epsilon(z)$ directly and thus the third step above is in effect embedded into the key equation solution. Blahut [4] also noted that for the Berlekamp-Massey algorithm, it is unnecessary to compute the modified syndrome polynomial: if the registers used to compute $\Lambda_e(z)$ are initialized to

$\Lambda_\epsilon(z)$ instead of 1, then the "discrepancies" calculated in the Berlekamp-Massey algorithm are exactly those needed for solving the modified key equation, and the algorithm produces $\Lambda(z)$ directly instead of $\Lambda_\epsilon(z)$. Unfortunately, reformulated Berlekamp-Massey algorithms such as the riBM and RiBM algorithms of [10] as well as *all* key equation solvers that are based on the extended Euclidean algorithm *do* need $\hat{S}(z)$. However, these algorithms are able to embed the third step above into the key equation solution. Finally, it has been noted by several researchers that the operations used for the solution of the key equation can be adapted to the computation of $\Lambda_\epsilon(z)$ or $\hat{S}(z)$ or both. Thus, the same hardware can be used in these calculations, which reduces the number of finite-field multipliers required.

A. Reformulation of Errors-and-Erasures Decoding Algorithms

As pointed out in [14], the modified key equation $\Lambda_e(z)\hat{S}(z) \equiv \Omega(z) \bmod z^{2t}$ can be solved by using the extended Euclidean algorithm shown below:

EEA for Errors-and-Erasures Decoding

1) **Initialization: Set** $v^{(0)}(z) \leftarrow z^{2t}$, $v^{(1)}(z) \leftarrow \hat{S}(z)$, $x^{(0)}(z) \leftarrow 0$, $x^{(1)}(z) \leftarrow 1$, and $j \leftarrow 0$.

2) **Iteration: While** $\deg\left[v^{(j)}(z)\right] \geq t + \mu/2$,
   Divide $v^{(j-1)}(z)$ by $v^{(j)}(z)$ to obtain both the quotient $q^{(j)}(z) \leftarrow \left\lfloor \frac{v^{(j-1)}(z)}{v^{(j)}(z)} \right\rfloor$ and the remainder $v^{(j+1)}(z) \leftarrow v^{(j-1)}(z) - q^{(j)}(z)v^{(j)}(z)$.
   Set $x^{(j+1)}(z) \leftarrow x^{(j-1)}(z) - q^{(j)}(z)x^{(j)}(z)$.
   Set $j \leftarrow j + 1$.

3) **Output:** $\Lambda_e(z) = x^{(j)}(z)$, $\Omega(z) = v^{(j)}(z)$.

This algorithm is clearly similar to the EO algorithm for errors-only decoding. In fact, the only differences between the two algorithms are the initial values of $v^{(1)}(z)$ and the stopping condition. Hence, direct implementation based on the above algorithm suffers the same problems we described in Section III.A. Using the same reformulation steps as in Section III, the above algorithm can be modified to a $(2t - \mu)$-iteration algorithm that eliminates the degree checking and produces $z^c\Lambda_e(z)$ and $z^c\Omega(z)$ instead of $\Lambda_e(z)$ and $\Omega(z)$ respectively.

As mentioned above, the operations used to compute $\Lambda(z) = \Lambda_e(z)\Lambda_\epsilon(z)$ in the third step can be embedded in the modified Euclidean algorithm by initializing $x^{(0)}(z)$ and $x^{(1)}(z)$ to scaled values $0 \cdot \Lambda_\epsilon(z)$ and $1 \cdot \Lambda_\epsilon(z)$ respectively. Note that the updates of $x^{(j)}(z)$ depend on $q^{(j)}(z)$, which are not at all affected by the change in the inital values. Thus, each $x^{(j)}(z)$ is scaled by $\Lambda_\epsilon(z)$, leading to a final output $\Lambda_e(z)\Lambda_\epsilon(z) = \Lambda(z)$.

In order to combine the computation of $\Lambda_\epsilon(z)$ and $\hat{S}(z)$ with the modified Euclidean algorithm (with the computation of $\Lambda(z)$ embedded) into a single algorithm with $2t$ iterations, we introduce a new polynomial $\psi(z) = \sum_{i=1}^{\mu} \psi_i z^{i-1}$ of degree $\mu - 1$ where the ψ_i's are the erasure locations. This polynomial can be formed easily during the syndrome computation stage by saving the marked erasure locations. Note that we allow for the possibility that more than $2t$ erasures have occurred, even

though such an errata pattern is not decodable. Our reformulated EEA for errors-and-erasures decoding is as follows:

Algorithm II

1) **Initialization:** $\delta \leftarrow -1$, $U(z) \leftarrow z^{2t}$, $V(z) \leftarrow S(z)$, $X(z) \leftarrow 1$, $W(z) \leftarrow 0$, and $\psi(z) = \sum_{i=1}^{\mu} \psi_i z^{i-1}$.

2) **Iteration: Repeat** $2t$ times:

Set FIRST $\leftarrow (\psi_0 \neq 0)$,

SWAP $\leftarrow (\neg$FIRST$) \wedge (V_{2t-1} \neq 0) \wedge (\delta < 0)$,

$$(\xi, \gamma) \leftarrow \begin{cases} (V_{2t-1}, U_{2t}), & \text{if FIRST } = 0, \\ (1, \psi_0), & \text{if FIRST } = 1. \end{cases}$$

$$\delta \leftarrow \begin{cases} -\delta - 1, & \text{if SWAP } = 1, \\ \delta - 1, & \text{if SWAP } = 0 \text{ and FIRST } = 0, \\ \delta, & \text{if FIRST } = 1, \end{cases}$$

$$\psi(z) \leftarrow \left\lfloor \frac{\psi(z)}{z} \right\rfloor,$$

$$V(z) \leftarrow \gamma \cdot zV(z) - \begin{cases} \xi \cdot U(z), & \text{if FIRST } = 0, \\ \xi \cdot V(z), & \text{if FIRST } = 1, \end{cases}$$

$$X(z) \leftarrow \gamma \cdot zX(z) - \xi \cdot W(z)$$

$$U(z) \leftarrow \begin{cases} zV(z), & \text{if SWAP } = 1, \\ U(z), & \text{if SWAP } = 0. \end{cases}$$

$$W(z) = \begin{cases} zX(z), & \text{if SWAP } = 1, \\ W(z), & \text{if SWAP } = 0 \text{ and FIRST } = 0, \\ \gamma \cdot zX(z) - \xi \cdot W(z), & \text{if FIRST } = 1. \end{cases}$$

3) **Output:** $\Lambda(z) = X(z)$, $\Omega(z) = V(z)$, and δ.

This algorithm uses a Boolean control variable FIRST that has value 1 only when the erasure locations are being processed to compute $\Lambda_\epsilon(z)$ and $\hat{S}(z)$. During this time, SWAP is always 0 and γ is set to the erasure location being processed currently. For each erasure location γ, Algorithm II sets $V(z)$ to $V(z)[1 - \gamma z]$ and both $X(z)$ and $W(z)$ to $X(z)[1 - \gamma z]$, thus obtaining $\hat{S}(z)$ and $\Lambda_\epsilon(z)$ after all the μ erasure locations have been processed one by one. Note that the update $\psi(z) \leftarrow \lfloor \psi(z)/z \rfloor$ discards the erasure location that was just processed and replaces it by the next erasure location to be processed, and thus FIRST becomes zero after μ iterations. From this point onwards, the updates of all the polynomials and of δ are the same as in Algorithm I*, and thus solve the modified key equation in $2t - \mu$ iterations. We remark that the received words with no erasures can be decoded correctly by Algorithm II as well. In fact, Algorithm I* corresponds to the special case of Algorithm II where $\mu = 0$ and FIRST is always 0. Similar to Theorem 1, we have

Theorem 2: Suppose that ν errors and μ erasures have occurred where $2\nu + \mu \leq 2t$. Let $\eta = \nu + \mu$ denote the total number of errata. Then when Algorithm II terminates, $\delta < 0$ and

$$\begin{aligned} & (X_{2t}, \quad X_{2t-1}, \quad \ldots, \quad X_{2t-\eta}, \quad X_{2t-\eta-1} \quad \ldots, \quad X_0) \\ = \ & (\Lambda_\eta, \quad \Lambda_{\eta-1}, \quad \ldots, \quad \Lambda_0, \quad 0, \qquad \ldots, \quad 0), \\ & (V_{2t}, \quad V_{2t-1}, \quad \ldots, \quad V_{2t-\eta}, \quad V_{2t-\eta-1} \quad \ldots, \quad V_0) \\ = \ & (0, \qquad \Omega_{\nu-1}, \quad \ldots, \quad \Omega_0, \quad 0, \qquad \ldots, \quad 0). \end{aligned}$$

If Algorithm II terminates with $\delta \geq 0$ or with $\psi_0 \neq 0$, then the errata pattern is not correctable.

V. Concluding Remarks

In this paper, modified Euclidean algorithms that use *fixed* numbers of iterations are proposed for both errors-only and errors-and-erasures decoding of RS codes. The salient feature of fixed numbers of iterations leads to simpler control mechanisms and hence hardware savings. The new algorithm for errors-and-erasures decoding seamlessly combines the three steps typically used in previously proposed architectures into one procedure, leading to hardware savings.

References

[1] J. H. Baek, J. Y. Kang and M. H. Sunwoo, "Design of a high-speed Reed-Solomon decoder," *Proceedings of the 2002 IEEE International Symposium on Circuits and Systems, ISCAS 2002,* May 2002, vol. 5, pp. 793–796.

[2] E. R. Berlekamp, *Algebraic Coding Theory*, McGraw-Hill, New York, 1968.

[3] E. Berlekamp, G. Seroussi, and P. Tong, "A hypersystolic Reed-Solomon decoder," in *Reed-Solomon Codes and Their Applications*, S. B. Wicker and V. K. Bhargava, Eds., chapter 10. IEEE Press, Piscataway NJ, 1994.

[4] R. E. Blahut, *Theory and Practice of Error-Control Codes*, Addison-Wesley, Reading MA, 1983.

[5] R. P. Brent and H. T. Kung, "Systolic VLSI arrays for polynomial GCD computation," *IEEE Transactions on Computers*, vol. C-33, pp. 731-736, 1984.

[6] J. L. Dornstetter, "On the equivalence between Berlekamp's and Euclid's algorithms," *IEEE Transactions on Information Theory*, vol. IT-33, pp. 428–431, May 1987.

[7] A. E. Heydtmann and J. M. Jensen, "On the equivalence of the Berlekamp-Massey and the Euclidean algorithms for decoding," *IEEE Transactions on Information Theory*, vol. 46, pp. 2614–2624, November 2000.

[8] C.-T. Huang and C.W. Wu, "VLSI design of a high speed pipelined Reed-Solomon CODEC," *Proceedings of the International Symposium on Multi-Technology Information Processing (ISMIP),* Hsinchu, December 1996, pp. 517–522.

[9] S. Kwon and H. Shin, "An area-efficient VLSI architecture of a Reed-Solomon decoder/encoder for digital VCRs," *IEEE Transactions on Consumer Electronics*, vol. 43, no. 4 pp. 1019–1027, November 1997.

[10] D. V. Sarwate and N. R. Shanbhag, "High-speed architectures for Reed-Solomon decoders," *IEEE Transactions on VLSI Systems*, vol. 9, pp. 941–955, October 2001.

[11] H. M. Shao, T. K. Truong, L. J. Deutsch, J. H. Yuen, and I. S. Reed, "A VLSI design of a pipeline Reed-Solomon decoder," *IEEE Transactions on Computers*, vol. C-34, pp. 393–403, 1985.

[12] H. M. Shao and I. S. Reed, "On the VLSI design of a pipeline Reed-Solomon decoder using systolic arrays," *IEEE Transactions on Computers*, vol. 37, pp. 1273–1280, 1988.

[13] Y. Sugiyama, M. Kasahara, S. Hirasawa, and T. Namekawa, "A method for solving key equation for decoding Goppa codes," *Information and Control*, vol. 27, pp. 87–99, January 1975.

[14] Y. Sugiyama, M. Kasahara, S. Hirasawa, and T. Namekawa, "An errors-and-erasures decoding algorithm for Goppa codes," *IEEE Transactions on Information Theory*, vol. IT-22, pp. 238–241, March 1976.

[15] T. K. Truong, W. L. Eastman, I. S. Reed, and I. S. Hsu, "Simplified procedure for correcting both errors and erasures of Reed-Solomon code using Euclidean algorithm," in *Proc. IEE, Part E*, November 1988, vol. 135, pp. 318–324.

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

[17] W. Wilhelm, "A new scalable VLSI architecture for Reed-Solomon decoders," *IEEE Journal of Solid-State Circuits*, vol. 34, no. 3, pp. 388–396, March 1999.

[18] T. Zhang and K. K. Parhi, "On the high-speed implementation of errors-and-erasures correcting Reed-Solomon decoders," *Proc. 12th Great Lakes Symposium on VLSI,* 2002.