Journal of VLSI Signal Processing 39, 93–111, 2005

Towards a VLSI Architecture for Interpolation-Based Soft-Decision Reed-Solomon Decoders*

WARREN J. GROSS
Department of Electrical and Computer Engineering, McGill University, 3480 University Street, Montreal, Quebec, H3A 2A7, Canada

FRANK R. KSCHISCHANG
Edward S. Rogers Sr. Department of Electrical and Computer Engineering, University of Toronto, 10 King's College Road, Toronto, Ontario, M5S 3G4, Canada

RALF KOETTER
Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, Urbana, IL, 61801, USA

P. GLENN GULAK
Edward S. Rogers Sr. Department of Electrical and Computer Engineering, University of Toronto, 10 King's College Road, Toronto, Ontario, M5S 3G4, Canada

Received March 20, 2003; Revised July 11, 2003; Accepted November 13, 2003

First online version published in September, 2004

Abstract. The Koetter-Vardy algorithm is an algebraic soft-decision decoder for Reed-Solomon codes which is based on the Guruswami-Sudan list decoder. There are three main steps: (1) multiplicity calculation, (2) interpolation and (3) root finding. The Koetter-Vardy algorithm seems challenging to implement due to the high cost of interpolation. Motivated by a VLSI implementation viewpoint we propose an improvement to the interpolation algorithm that uses a transformation of the received word to reduce the number of iterations. We show how to reduce the memory requirements and give an efficient VLSI implementation for the Hasse derivative.

Keywords: Reed-Solomon decoders, soft-decision decoding, VLSI architectures, list decoding, Sudan's algorithm, Guruswami-Sudan algorithm, Koetter-Vardy algorithm, polynomial interpolation, Hasse derivative

1. Introduction

Reed-Solomon codes are powerful error-correcting codes that can be found in a wide variety of digital communications systems, from digital media to wireless communications and deep-space probes. The ubiquitous nature of these codes continues to fuel research into decoding algorithms some forty years after their introduction (see e.g. [1]).

Reed-Solomon codes have been employed in a wide spectrum of digital communications systems because they provide powerful error correction with only a small number of overhead symbols. Reed-Solomon codewords consist of non-binary symbols and therefore the correction of a single symbol could result in

*Portions of this work were presented at the 2002 IEEE Workshop on Signal Processing Systems, October 2002.

the correction of more than one of the constituent bits. For this reason, Reed-Solomon codes are well suited to the correction of burst errors.

Classical decoders for Reed-Solomon codes of length n and dimension k can correct up to $t = \lfloor d_{\min}/2 \rfloor$ errors where $d_{\min} = (n - k + 1)$ is the minimum distance of the code. Recently, a new class of *list decoding* algorithms has been introduced that can sometimes correct an even greater number of errors [2, 3]. The list decoding problem is to find the set of codewords at a Hamming distance of t' from the received word. If $t' > d_{\min}/2$ there might not be a unique codeword so the decoder returns a list of candidate codewords. The Guruswami-Sudan (GS) list decoding algorithm has t' as large as $n - \sqrt{nk}$ errors [3].

To improve the error-correction capability of a decoder even further, the decoder should take advantage of the *soft* reliability information available from the channel. Soft-decision decoders can provide an asymptotic gain of 2–3 dB on Gaussian channels [4] and 10 dB or more on Rayleigh fading channels. Traditional hard-decision Reed-Solomon decoding algorithms are efficient because they are algebraic; that is, they exploit the underlying algebraic structure of the code to generate a system of equations that is solved using finite field arithmetic. However, an algebraic decoder based on finite field arithmetic does not appear to be compatible with the real-valued, *soft* information available from the channel and therefore it has been a research challenge to develop an algebraic soft-decision Reed-Solomon decoder. Koetter and Vardy have recently proposed an algebraic soft-decision decoding algorithm by extending the list decoder of Guruswami and Sudan to include a method for converting soft information into algebraic conditions [5, 6]. The Koetter-Vardy (KV) algorithm can achieve up to about 4 dB of coding gain at a frame-error-rate (FER) of 10^{-3} on a Gaussian noise channel (with a practical range of 1–1.5 dB) and gains of 2–7 dB on a Rayleigh fading channel [7, 8].

The Koetter-Vardy soft-decision decoding procedure shows a lot of promise from the point of view of error correcting performance. At a first glance, the algorithm seems to be quite computationally complex and not straightforward to implement in VLSI. This paper aims to introduce techniques that reduce the complexity of interpolation-based decoders to the point where an efficient VLSI implementation is possible. Section 2 is a review of the GS and KV list-decoding algorithms. In Section 3 we describe techniques for significantly reducing the complexity and memory requirements of interpolation-based decoders. A VLSI architecture is then developed in Section 4 that reduces the complexity of evaluating the Hasse derivative, one of the main tasks in interpolation. We offer conclusions in Section 5.

2. Interpolation-Based List Decoding Algorithms

Say we want to transmit a message f. The bits of the message can be grouped into $\log_2(q)$-bit symbols chosen from the finite field with q elements, GF(q). An (n, k) Reed-Solomon code over GF(q) represents the k-symbol message, $f = (f_0, f_1, f_2, \ldots, f_{k-1})$, by an n-symbol codeword, $c = (c_0, c_1, c_2, \ldots, c_{k-1}, \ldots, c_{n-1})$, where $n > k$ and usually $n = q - 1$. The k symbols of the message f can be considered to be the coefficients of the up to degree $(k - 1)$ univariate message polynomial:

$$f(x) = f_0 + f_1 x + f_2 x^2 + \cdots + f_{k-1} x^{k-1}. \quad (1)$$

In this paper, we use the classical view of Reed-Solomon codes taken from the original definition in [9]. With this *evaluation map* encoding method, a codeword is formed by evaluating the message polynomial $f(x)$ at n elements of GF(q). If the set of evaluation elements is $X = \{x_0, x_1, \ldots, x_{n-1}\}$, the codeword c is:

$$c = (f(x_0), f(x_1), \ldots, f(x_{n-1})), \quad x_i \in X. \quad (2)$$

In this paper, we will always assume that $n = q - 1$ and the set of evaluation elements X is the set of nonzero elements of GF(q):

$$\begin{aligned} X &\triangleq \{x_0, x_1, x_2, \ldots, x_{n-1}\} \\ &\triangleq \{1, \alpha, \alpha^2, \ldots, \alpha^{n-1}\}, \end{aligned} \quad (3)$$

where α is a primitive element in GF(q). The evaluation map encoding method is useful because it provides insight leading to interpolation-based decoding algorithms.

2.1. The Guruswami-Sudan Algorithm

An *interpolation-based* decoder takes the point of view that a codeword is a message polynomial evaluated at points in a finite field and uses polynomial interpolation to try to reconstruct that polynomial. The Guruswami-Sudan (GS) algorithm is an interpolation-based list decoder for Reed-Solomon codes. To describe the algorithm, we will first need to review some notation

and facts about bivariate polynomials, which are the basic data structures in the algorithm. Consider the bivariate polynomial with coefficients chosen from a finite field:

$$P(x, y) = \sum_{a=0}^{\infty} \sum_{b=0}^{\infty} p_{a,b} x^a y^b \in \mathrm{GF}(q)[x, y]. \quad (4)$$

Let w_x and w_y be non-negative real numbers. The (w_x, w_y)-*weighted degree* of $P(x, y)$, $\deg^{(w_x,w_y)}(P)$, is defined as the maximum over all the numbers $aw_x + bw_y$ such that $p_{a,b} \neq 0$. Most often, the choice of finite field will be GF(2^ω). The (μ,ν)'th formal derivative of a polynomial $P(x, y)$ over GF(2^ω), $P^{(\mu,\nu)}(x, y)$, will vanish for $\mu \geq 2$ or $\nu \geq 2$. To be able to define these higher-order derivatives, we can use a related concept, the *Hasse derivative* [10]. The (μ, ν)'th *Hasse derivative* of a bivariate polynomial $P(x, y)$ is defined for integers $\mu, \nu \geq 0$ as:

$$P^{[\mu,\nu]}(x, y) = \sum_{a \geq \mu \wedge b \geq \nu} \binom{a}{\mu} \binom{b}{\nu} p_{a,b} x^{a-\mu} y^{b-\nu}. \quad (5)$$

To perform interpolation-based decoding, we will need to ensure that bivariate polynomials and their Hasse derivatives pass through certain points. We say that a bivariate polynomial $P(x, y)$ passes through a point (x_i, y_j) with multiplicity m if $P^{[\mu,\nu]}(x_i, y_j) = 0$ for all integers μ, ν, such that $\mu + \nu < m$.

Consider the received word $y = c + e$, where e is an error vector with components drawn from GF(q). Since each component of c was generated by evaluating $f(x)$ at a unique value of $x \in X$, a unique x_i can be associated with each received $y_i \in \mathrm{GF}(q)$ to form the list of points, $\mathbb{L} = \{(x_0, y_0), (x_1, y_1), \ldots, (x_{n-1}, y_{n-1})\}$. If there is no noise ($e = 0$), then $y_i = f(x_i),\ 0 \leq i < n$, and a bivariate polynomial, $P(x, y) = y - f(x)$, passes through all the points in $\mathbb{L}$ with a multiplicity of one. This suggests that an interpolation-based approach can be used to decode Reed-Solomon codes. In the presence of noise ($e \neq 0$), the interpolation polynomial will pass through some points that are not part of the codeword. The GS algorithm ensures that under certain conditions, the codeword polynomial "lives inside" the interpolation polynomial [2, 3]. The GS algorithm is an interpolation-based list decoder with two main steps:

1. Interpolation Step: Given the set of points $\mathbb{L}$ and a positive integer m, compute $P(x, y) \in \mathrm{GF}(q)[x, y]\backslash\{0\}$ of minimal $(1, k-1)$-weighted degree that passes through all the points in $\mathbb{L}$ with multiplicity at least m.
2. Factorization Step: Given the interpolation polynomial $P(x, y)$, identify all the factors of $P(x, y)$ of the form $y - f(x)$ with $\deg f(x) < k$. The output of the algorithm is a list of the codewords that correspond to these factors.

A complete factorization of $P(x, y)$ is not necessary since we are just looking for linear y-roots of degree $< k$. An appropriate root-finding algorithm is given in [11]. The multiplicity, m, functions as a user-selectable complexity parameter. The error-correcting ability of the GS algorithm increases as the value of m increases. Unfortunately, so does the decoding complexity.

2.2. *The Koetter-Vardy Algorithm*

Guruswami and Sudan hint at a possible soft-decision extension to their algorithm in [3] by allowing each point on the interpolated curve to have its own multiplicity. Koetter and Vardy proposed a method to perform soft-decision decoding by assigning unequal multiplicities to points according to their relative reliabilities. We note that all possible $(q \times n)$ transmitted/received symbol pairs are considered and not just the ones corresponding to the hard decisions. Once multiplicities have been assigned, the rest of the decoding proceeds according to the Guruswami-Sudan algorithm allowing for unequal multiplicities and hence a variable number of linear constraints for each point. A block diagram of the soft-decision algorithm is given in Fig. 1.

Consider a codeword whose symbols are drawn from a finite field $\mathrm{GF}(q) = \{\alpha_0, \alpha_1, \ldots, \alpha_{q-1}\}$ and transmitted across a memoryless channel. The optimum value of the *soft information* or *reliability* for a received value $\beta_j \in \mathrm{GF}(q)$ given that the symbol $\alpha_i \in \mathrm{GF}(q)$ was sent

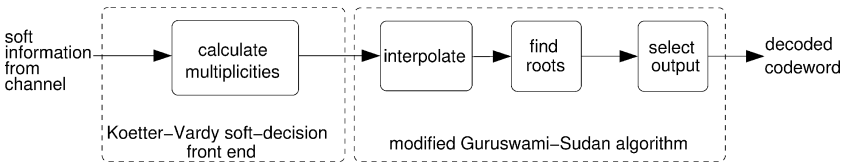

Figure 1. The Koetter-Vardy algorithm.

is the *a posteriori probability* (APP):

$$\pi_{i,j} = \Pr(\alpha_i \text{ sent} \mid \beta_j \text{ received}), \tag{6}$$

where $0 \le i < q$ and $0 \le j < n$. The soft information is therefore a real number that can take on any value between zero and one. In practice, it can be quantized to any finite number of levels. Soft information can be calculated directly from the received channel information if we have knowledge of the noise model. If the noise probability density function (PDF) is $\eta(\beta_j \mid \alpha_i)$ then by Bayes' Rule,

$$\begin{aligned}\pi_{i,j} &= \Pr(\alpha_i \mid \beta_j) \\ &= \frac{\eta(\beta_j \mid \alpha_i)\Pr(\alpha_i)}{\sum_{\alpha_k \in \mathrm{GF}(q)} \eta(\beta_j \mid \alpha_k)\Pr(\alpha_k)} \\ &= \frac{\eta(\beta_j \mid \alpha_i)}{\sum_{\alpha_k \in \mathrm{GF}(q)} \eta(\beta_j \mid \alpha_k)}.\end{aligned} \tag{7}$$

A $(q \times n)$ *reliability matrix* Π whose columns sum to unity can be constructed from the $\pi_{i,j}$. Each entry $\pi_{i,j}$ therefore represents the reliability of a point (x_j, y_i). A hard decision vector $r = (r_0, r_1, \ldots, r_{n-1})$ can be extracted from Π where

$$r_j = y_{\mathrm{argmax}(\pi_{i,j})}, \quad 0 \le j < n. \tag{8}$$

Ultimately, the information in the reliability matrix has to be translated to a set of weighted points in the form of nonnegative integers in a $(q \times n)$ *multiplicity matrix* M. Algorithm 1, proposed in [6], is an algorithm for generating a matrix M from Π subject to the constraint that the sum of the entries of M is an integer s: $\sum_{i=0}^{q-1} \sum_{j=0}^{n-1} m_{i,j} = s$. Algorithm 1 requires $s = O(n)$ passes through the $(q \times n)$ reliability matrix resulting in an $O(n^3)$ algorithm. A lower complexity $O(n^2)$ algorithm for implementing the KV front-end is proposed in [7, 8].

Algorithm 1. Algorithm for calculating M from Π subject to complexity constraint s (from [6]).

Choose a desired value for $s = \sum_{i=0}^{q-1} \sum_{j=0}^{n-1} m_{i,j}$
$\Pi^* \leftarrow \Pi$; $M \leftarrow 0$
While $s > 0$ **do**
  Find the position (i, j) of the largest entry $\pi^*_{i,j}$ in Π^*
  $\pi^*_{i,j} \leftarrow \frac{\pi_{i,j}}{m_{i,j}+2}$
  $m_{i,j} \leftarrow m_{i,j} + 1$
  $s \leftarrow s - 1$
end while

2.3. Bivariate Interpolation

The interpolation step consists of solving the linear system:

$$P^{[\mu,\nu]}(x_j, y_i) = 0, \quad \mu + \nu < m_{i,j}, \tag{9}$$

for all triples $(x_j, y_i, m_{i,j})$, $0 \le i < q$, $0 \le j < n$. Gaussian elimination could be used to solve this system with complexity $O(n^3)$ however if we exploit the special nature of this problem the complexity can be reduced. Fast algorithms for interpolation are described in [11–16]. We use the algorithm from [16] for the GS algorithm which is easily modified to handle unequal multiplicities (Algorithm 2). The *cost* of interpolation, C, is the number of linear equations that need to be satisfied for the interpolation. If an entry in M is increased from $m_{i,j}$ to $m_{i,j} + 1$, this introduces $m_{i,j} + 1$ additional linear constraints on the interpolation problem. The total cost of decoding with multiplicity matrix M is therefore:

$$C = \frac{1}{2} \sum_{i=0}^{q-1} \sum_{j=0}^{n-1} m_{ij}(m_{ij} + 1). \tag{10}$$

Algorithm 2 runs for C iterations. At each iteration, a maximum of $(d_y + 1)$ polynomials are updated where,

$$d_y = \left\lfloor \frac{1 + \sqrt{1 + \frac{8C}{k-1}}}{2} \right\rfloor - 1. \tag{11}$$

The maximum $(1, k-1)$-weighted degree of the polynomials is

$$d_x = \left\lfloor \frac{C}{d_y + 1} + \frac{d_y(k-1)}{2} \right\rfloor. \tag{12}$$

The bivariate polynomials will have maximum x-degree of d_x and a maximum y-degree of d_y. At the end of each iteration, each of the $(d_y + 1)$ polynomials in the set G satisfy all of the linear constraints considered up to that point. If we want to read out an answer, we choose the polynomial with the smallest $(1, k-1)$-weighted degree. To keep the cost down, we would like to keep the m_{ij} and hence s small, however, in general, the error-rate performance improves with increasing s. This gives the algorithm a tunable parameter s to trade-off performance with decoding complexity.

Algorithm 2. Interpolation algorithm.

$G \leftarrow \{g_0 = 1, g_1 = y, g_2 = y^2, \ldots, g_{d_y} = y^{d_y}\}$.
for each point (x_j, y_i) with multiplicity $m_{i,j} > 0$ **do**
  for $\alpha \leftarrow 0$ to $m_{i,j} - 1$ **do**
    for $\beta \leftarrow 0$ to $m_{i,j} - 1 - \alpha$ **do**
      $f \leftarrow \min_{\deg^{(1,k-1)}} \{g \in G$ such that $g^{[\alpha,\beta]}(x_j, y_i) \neq 0\}$
      for $g \in G$ such that $g \neq f$ **do**
        $g \leftarrow g \cdot f^{[\alpha,\beta]}(x_j, y_i) - f \cdot g^{[\alpha,\beta]}(x_j, y_i)$
      end for
      $f \leftarrow (x - x_j)f$
    end for
  end for
end for

2.4. An Example of Soft-Decision Decoding

We illustrate how the KV algorithm works with an example over a small field.

Example 1. Consider a (7,5) Reed-Solomon code over GF(8) where α is a root of the primitive polynomial $x^3 + x + 1$. A classical hard-decision decoder for this code can correct a single symbol error. Say we want to transmit the message:

$$f = (\alpha, \alpha^4, \alpha^4, \alpha^3, 1).$$

Then the message polynomial is $f(x) = \alpha + \alpha^4 x + \alpha^4 x^2 + \alpha^3 x^3 + x^4$ and the corresponding codeword using an evaluation map encoding is:

$$\begin{aligned} c &= (f(1), f(\alpha), f(\alpha^2), f(\alpha^3), f(\alpha^4), f(\alpha^5), f(\alpha^6)) \\ &= (0, \alpha^3, \alpha^3, 0, 1, \alpha^2, \alpha^5). \end{aligned}$$

The codeword is transmitted bit-by-bit using BPSK modulation over an AWGN channel. The soft-input to the KV algorithm is a $(q \times n)$ reliability matrix formed using the information received from the channel:

$$\Pi = \begin{bmatrix} 0.959796 & 0.214170 & 0.005453 & 0.461070 & 0.001125 & 0.000505 & 0.691729 \\ 0.001749 & 0.005760 & 0.000000 & 0.525038 & 0.897551 & 0.025948 & 0.000209 \\ 0.028559 & 0.005205 & 0.000148 & 0.003293 & 0.000126 & 0.018571 & 0.020798 \\ 0.000052 & 0.000140 & 0.000000 & 0.003750 & 0.100855 & 0.954880 & 0.000006 \\ 0.009543 & 0.736533 & 0.968097 & 0.003180 & 0.000000 & 0.000000 & 0.278789 \\ 0.000017 & 0.019810 & 0.000006 & 0.003621 & 0.000307 & 0.000003 & 0.000084 \\ 0.000284 & 0.017900 & 0.026295 & 0.000023 & 0.000000 & 0.000002 & 0.008382 \\ 0.000001 & 0.000481 & 0.000000 & 0.000026 & 0.000035 & 0.000092 & 0.000003 \end{bmatrix}.$$

Let's examine the hard-decision:

$$r = (0, \alpha^3, \alpha^3, \mathbf{1}, 1, \alpha^2, \mathbf{0}),$$

even though it is not explicitly needed for soft-decision decoding. Since r has errors in two positions (indicated in bold), it is uncorrectable by a classical hard-decision decoder. However, the soft-decision KV algorithm might be able to tackle the problem. Running Algorithm 1 on Π with $s = 12$ gives the multiplicity matrix:

$$M = \begin{bmatrix} 2 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 1 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

The interpolation algorithm (Algorithm 2) is run with $d_y = 2$ and $l = 9$ on the following input:

x	y	Multiplicity
1	0	2
α	α^3	1
α^2	α^3	2
α^3	0	1
α^3	1	1
α^4	1	2
α^5	α^2	2
α^6	0	1

This means that there are $(d_y + 1) = 3$ bivariate polynomials being updated with maximum x-degree of $d_x = 9$ and maximum y-degree of $d_y = 2$. At the end of $C = 16$ iterations, the interpolation polynomial of minimum weighted degree, obtained by using Algorithm 2 is:

$$\begin{aligned} P(x, y) = {} & \alpha^2 x^5 y + \alpha^2 x^9 + \alpha^4 y^2 + \alpha^5 x^4 y + \alpha^4 x^8 \\ & + \alpha^4 x^3 y + x^7 + \alpha^3 x^6 + \alpha^3 x y + \alpha^4 x^5 \\ & + \alpha^4 y + \alpha^3 x^4 + \alpha^5 x^3 + \alpha^2 x^2 + \alpha^2 x + \alpha. \end{aligned}$$

The Roth-Ruckenstein root-finding algorithm from [11] finds the linear y-roots of the form $y - f(x)$, $\deg(f(x)) < k$ in the factorization of $P(x, y)$. The output list contains two candidates for $f(x)$:

$$\begin{aligned} \hat{f}_1(x) &= \alpha + \alpha^4 x + \alpha^4 x^2 + \alpha^3 x^3 + x^4, \\ \hat{f}_2(x) &= \alpha^3 + \alpha^3 x + \alpha^4 x^2 + \alpha x^3 + \alpha^3 x^4. \end{aligned}$$

To choose from the list, we encode the candidate messages to find their corresponding codewords:

$$\begin{aligned} \hat{c}_1 &= (0, \alpha^3, \alpha^3, 0, 1, \alpha^2, \alpha^5), \\ \hat{c}_2 &= (\alpha^5, \alpha^5, 1, \alpha^2, \alpha^5, 0, 1). \end{aligned}$$

The probability of each candidate codeword can be obtained by multiplying the appropriate probabilities from Π. It is found that $\hat{c}_1$ is more likely and hence $\hat{f}_1 = (\alpha, \alpha^4, \alpha^4, \alpha^3, 1)$ is the correctly decoded message.

3. Fast Algorithms for Interpolation-Based Decoding

The Koetter-Vardy soft-decision decoding procedure shows a lot of promise from the point of view of error correcting performance. At first glance, the algorithm seems to be quite computationally complex and not straightforward to implement. This section introduces algorithmic techniques that reduce the complexity of interpolation-based decoders to the point where efficient software or VLSI implementations are possible. We will begin by looking at an example of a software implementation of a KV decoder.

Example 2. We implemented a software KV decoder. The soft-decision front end uses the reduced complexity algorithm from [7, 8]. Interpolation is performed using Nielsen's algorithm (Algorithm 2) and the factoring is carried out by the Roth-Ruckenstein root-finding algorithm from [11].

The software was implemented in C and the tests were run on a 2.4 GHz P4. The test case was a RS(255,239) code where the maximum multiplicity was set to $m = 4$. This setting of m is a compromise between decoding complexity and soft-decision gain. We ran a decoding trial on a test case which had nine errors in the corresponding hard-decision received word (this is a case that a classical decoder would not be guaranteed to correct). The total decoding time for one codeword was 543.4 ms. The interpolation took 452 ms (83% of the total) and the root finding took 91.4 ms (17% of the total). The other parts of the algorithm (soft-decision front end, output encoding and selection) contributed a negligible amount to the total.

An important measure of the complexity of decoding is the number of arithmetic operations required. We counted the number of Galois field additions and multiplications and found that there were 9.2×10^7 Galois field additions and 1.55×10^8 Galois field multiplications.

The decoder throughput in terms of decoded message bits per second is

$$\frac{k \log(q)}{543.4 \times 10^{-3}} = 3.5 \text{ kbps}.$$

A naive software implementation clearly is not fast enough for real-time applications. In this section we will explore techniques for reducing the complexity of the interpolation and root-finding algorithms. In Section 3.1 we describe techniques for significantly reducing the number of iterations and the memory requirements of the interpolation algorithm. Section 3.2 shows how the complexity reductions can be carried over into the factoring algorithm. Section 3.3 describes a software implementation of the Koetter-Vardy algorithm that applies the algorithmic improvements explored in this section.

3.1. Reduced-Complexity Interpolation

The interpolation algorithm is the most time-consuming component of KV decoding and it is essential to reduce its complexity if KV decoding is to be used in real-time applications. It is shown in [8] that if a multiplicity matrix has a maximum entry m, then the maximum interpolation cost would be the cost of hard-decision Guruswami-Sudan decoding with

multiplicity m:

$$C = \left(\frac{n}{2}\right) m(m+1).$$

For the remainder of this discussion, we will assume the worst-case where the cost of interpolation is the maximum possible cost, C. The interpolation algorithm needs to store $(d_y + 1)$ bivariate polynomials. Since a homogeneous linear system must have more unknowns than equations, the length (number of terms) of the polynomials must be at least C. The memory requirements of interpolation are $\approx (d_y + 1)C$ field elements. Therefore the complexity of interpolation in terms of the number of Galois field operations is $N_{\text{op,interp}} = O(d_y C^2)$.

Example 3. Consider a decoder for a RS(255,239) code with maximum multiplicity $m = \{1, 4, 16\}$. The interpolation cost, complexity and memory requirements are shown in Table 1.

From the example we see that the decoding complexity and memory requirements grow very quickly as the multiplicity increases. If the maximum multiplicity is fixed to deliver a desired error-rate, then to lower the cost and hence the complexity, the number of interpolation points (nonzero entries in the multiplicity matrix) that we apply the bivariate interpolation algorithm to must be reduced. We apply the trick of "reencoding" the received word [17–19] to reduce the interpolation complexity in the spirit of the Berlekamp-Welch (BW) algorithm [20–24].

3.1.1. Systematic Encoding. A *systematic* encoding is one where all the k input symbols to the encoder explicitly appear in the encoded codeword. If they appear in k consecutive positions then the encoding is called *strictly systematic*. Strictly systematic encoders, where the k message symbols appear as the last k symbols in a codeword, are easily implemented with a linear feedback shift register [25] and are commonly used.

Table 1. The maximum cost, complexity ($N_{\text{op,interp}}$) and memory requirements for the interpolation algorithm applied to a RS(255,239) code with maximum multiplicity m.

m	C	$N_{\text{op,interp}}$	Memory
1	255	1×10^5	512 bytes
4	2550	3×10^7	12 Kbytes
16	34680	2×10^{10}	576 Kbytes

Interpolation-based decoding algorithms rely on an evaluation map encoding, however it is more efficient to implement an encoder as a linear feedback shift register. We would also like to apply the decoder to existing Reed-Solomon transmission systems that use a systematic encoder. Therefore we would like to use a systematic encoding in place of the evaluation map encoding.

The standard Reed-Solomon codes in wide use can be considered as non-binary, cyclic, BCH codes. Cyclic codes are codes where for every codeword c, there exists another codeword that is a cyclic shift of c. The cyclic view of RS codes is useful because it leads to efficient algebraic decoding algorithms such as the Berlekamp-Massey algorithm or Euclid's algorithm. In general, the evaluation map encoding (which is the original view of RS codes) does not result in a cyclic code. Therefore, the cyclic RS codes are only a (small) subset of all possible RS codes. However, under the appropriate choice of evaluation values, a cyclic code is obtained by the evaluation map.

Theorem 1. *The Reed-Solomon code generated by an evaluation map with the fixed ordering* $\{1, \alpha, \alpha^2, \ldots, \alpha^{q-2}\}$ *over GF(q) is cyclic.*

Proof: Consider a message polynomial $f(x) = f_0 + f_1 x + f_2 x^2 + \cdots + f_{k-1} x^{k-1}$. The codeword generated by the evaluation map with the ordering $\{1, \alpha, \alpha^2, \ldots, \alpha^{q-2}\}$ over GF(q) is

$$c = (f(1), f(\alpha), f(\alpha^2), \ldots, f(\alpha^{q-2})). \tag{13}$$

We have $f(x) = \sum_{i=0}^{k-1} f_i x^i$ and therefore

$$c = \sum_{i=0}^{k-1} f_i u_i, \tag{14}$$

where $u_i = (1, \alpha^i, \alpha^{2i}, \alpha^{3i}, \ldots, \alpha^{(q-2)i})$. We have to show that a cyclic shift of c is also a codeword. The cyclic shift of c is

$$\begin{aligned} c' &= (f(\alpha^{q-2}), f(1), f(\alpha), \\ &\qquad f(\alpha^2), \ldots, f(\alpha^{q-4}), f(\alpha^{q-3})) \\ &= \sum_{i=0}^{k-1} f_i v_i, \end{aligned} \tag{15}$$

where $v_i = (\alpha^{(q-2)i}, 1, \alpha^i, \alpha^{2i}, \ldots, \alpha^{(q-3)i})$. The vector v_i can be written as

$$\begin{aligned} v_i &= \left(\alpha^{(q-2)i}, 1, \alpha^i, \alpha^{2i}, \ldots, \alpha^{(q-3)i}\right) \\ &= \alpha^{(q-2)i}\left(1, \alpha^{-(q-2)i}, \alpha^{-(q-3)i}, \ldots, \alpha^{-2i}, \alpha^{-i}\right) \end{aligned}$$

$$= \alpha^{(q-2)i}\left(1, \left(\alpha^{-(q-2)}\right)^i, \left(\alpha^{-(q-3)}\right)^i, \ldots, (\alpha^{-2})^i, (\alpha^{-1})^i\right). \quad (16)$$

But from the properties of GF(q), $\alpha^{-j} = \alpha^{q-1-j}$, and therefore

$$\begin{aligned} v_i &= \alpha^{(q-2)i}\left(1, \alpha^i, \alpha^{2i}, \ldots, \alpha^{(q-3)i}, \alpha^{(q-2)i}\right) \\ &= \alpha^{(q-2)i} u_i. \end{aligned} \quad (17)$$

Therefore,

$$\begin{aligned} c' &= \sum_{i=0}^{k-1} \left(f_i \alpha^{(q-2)i}\right) u_i \\ &= \sum_{i=0}^{k-1} g_i u_i, \end{aligned} \quad (18)$$

where $g_i = \alpha^{(q-2)i} f_i$. Define $g(x) = g_0 + g_1 x + g_2 x^2 + \cdots + g_{k-1} x^{k-1}$. Then

$$c' = (g(1), g(\alpha), g(\alpha^2), \ldots, g(\alpha^{q-2})), \quad (19)$$

and is therefore a codeword in the same code as c. □

Therefore the standard shift register encoder, which can generate strictly systematic encodings, can be used in place of the evaluation map encoding with evaluation values $\{1, \alpha, \alpha^2, \ldots, \alpha^{q-2}\}$. The decoder will decode to a different codeword in the same code which is subtracted from the received hard-decision word to obtain an estimate of the error vector. The estimated error vector can be used to recover an estimate of the original codeword.

We are also interested in generating systematic encodings where the information appears in *arbitrary* positions in an encoded codeword. If these positions can change for every encoder use, then an efficient way of implementing this systematic encoder is with an *erasures-only* Reed-Solomon decoder [25]. Since Reed-Solomon codes are maximum-distance-separable (MDS), a codeword may be perfectly recovered from any k of its symbols. An erasures-only decoder is much simpler than an error-and-erasures decoder since the erasure locations are known a-priori. Therefore, the expensive iterative Berlekamp-Massey algorithm for solving the key equation and the Chien search root-finding can be skipped [26]. We will see in Section 3.2, that an errors-only Berlekamp-Massey algorithm (BMA) is needed as part of the implementation and therefore a single implementation of an errors-and-erasures decoder can be used as both an errors-only decoder and an erasures-only decoder.

3.1.2. Reencoding. The idea of reencoding is to transform the interpolation problem into one that is easier to solve. The codeword c is transmitted through a noisy channel. The hard decision vector, $r = (r_0, r_1, \ldots, r_{n-1})$, which can be extracted from the reliability matrix Π, is $r = c + e$, where e is an error vector. The first step is to partition the received symbols in r into two sets, U ("unreliable") and R ("reliable"). The set R consists of the k most reliable symbols, where the reliability information can be derived from the multiplicity matrix M. The set of positions of the symbols in R (labeled from 1 to n corresponding to $\{\alpha^1, \alpha^2, \ldots, \alpha^n\}$) is the set R_k. Now systematically encode the symbols in R so that they appear in the *reencoded codeword*, ψ, in the same positions that they appeared in r. As discussed in Section 3.1.1, this can be done efficiently for k arbitrary positions with an erasures-only decoder. Taking the difference between r and ψ we get:

$$\begin{aligned} r' &= r - \psi \\ &= (c + e) - \psi \\ &= (c - \psi) + e, \end{aligned} \quad (20)$$

which is a codeword (by the linearity of the code) that is corrupted by the same error pattern as r. However, r' has a very interesting property; since the reencoding of r is systematic, k symbols of r' are zero. These zero symbols correspond to k interpolation points with a zero y-component:

$$V = \{(\alpha^i, 0)\}, \quad i \in R_k. \quad (21)$$

We assume for the moment that the k "reliable" points in R all have the same multiplicity, m. In Section 3.1.4 we will consider the case where this is not true. An interpolation polynomial for the k points in V is $v(x)^m$ where $v(x)$ is found through a simple univariate interpolation:

$$v(x) = \prod_{i \in R_k} (x - \alpha^i). \quad (22)$$

The advantage for high-rate codes is that we have found an interpolation polynomial for most of the points

without having to use the expensive bivariate interpolation algorithm. The calculation of $v(x)$ requires a single polynomial to be updated k times instead of $(d_y + 1)$ polynomials being updated $(k/2)(m^2 + m)$ times. Now that we have $v(x)$, the bivariate interpolation algorithm (Algorithm 2) is run at decode time starting from the initial polynomial set:

$$G = \left\{ v(x)^m, v(x)^{m-1}y, \ldots, v(x)^{m-d_y}y^{d_y} \right\}, \quad (23)$$

and run for at most:

$$\begin{aligned} C' &= \left(\frac{n-k}{2}\right)(m)\,(m+1) \\ &= \left(1 - \frac{k}{n}\right)C \end{aligned} \quad (24)$$

iterations, where C is the maximum cost of the bivariate interpolation without reencoding. The reduced cost, C', gets smaller as the code rate k/n increases.

We note that if reencoding was applied to a received word where there was no reliability information (hard-decision decoding), then it is convenient to choose the last k consecutive positions in r as R_k. Then the systematic encoder can be implemented with a linear feedback shift register. Since R_k is known a-priori, $v(x)$ can be calculated offline once and stored in memory.

3.1.3. Reducing the Memory Requirements. The memory requirements for interpolation can be very large since the maximum length of the bivariate polynomials is at least C terms. The polynomials can be shortened by factoring out the polynomial $v(x)$. It is shown in [18, 19] that if reencoding is used then the interpolation polynomial $P(x, y)$ can be written as:

$$P(x, y) = \sum_{j=0}^{d_y} w_j(x) \prod_{i \in R_k} (X - \alpha^i)^{m_i - j} T_j(x) y^j, \quad (25)$$

where,

$$T_j(x) = \prod_{i \in R_k} (x - \alpha^i)^{\max(j - m_i, 0)}. \quad (26)$$

This is the most general way to decompose $P(x, y)$ and it allows any k symbols to be chosen for R. However, it seems that the most logical choice for R is to choose the k symbols with the largest multiplicities to achieve the maximum complexity reduction. To obtain the shortest possible polynomials (keeping VLSI implementations in mind), we make the assumption (which will be justified in Section 3.1.4) that R consists of k points that have the maximum possible multiplicity $m = d_y$. Then $T_j(x) = 1$, $j = 0, \ldots, d_y$, and (25) reduces to [17]:

$$P(x, y) = \sum_{j=0}^{d_y} w_j(x) v(x)^{m-j} y^j. \quad (27)$$

which means that common factors of $v(x)$ are being carried around needlessly, wasting memory. It would be nice to factor out the powers of $v(x)$ and only have to calculate the $w_j(x)$ in real-time. The interpolation polynomial can be written as

$$\begin{aligned} P(x, y) &= \sum_{j=0}^{d_y} w_j(x) v(x)^{m-j} y^j \\ &= v(x)^m \sum_{j=0}^{d_y} w_j(x) \left(\frac{y}{v(x)}\right)^j. \end{aligned} \quad (28)$$

The decoding algorithm takes the transformed word r' as input and tries to estimate the transformed codeword $c' = c - \psi$. Therefore, if the decoding is successful, a message polynomial $f'(x)$ corresponding to c' will be a linear y-root of $P(x, y)$, i.e.:

$$P(x, y) = (y - f'(x))A(x, y), \quad (29)$$

or,

$$P(x, f'(x)) = 0. \quad (30)$$

From (28),

$$\begin{aligned} v(x)^m \sum_{j=0}^{d_y} w_j(x) \left(\frac{f'(x)}{v(x)}\right)^j &= 0 \\ \sum_{j=0}^{d_y} w_j(x) \left(\frac{f'(x)}{v(x)}\right)^j &= 0. \end{aligned} \quad (31)$$

Define the *reduced interpolation polynomial*:

$$\tilde{P}(x, \tilde{y}) = \sum_{j=0}^{d_y} w_j(x) \tilde{y}^j, \quad (32)$$

where $\tilde{y} = y/v(x)$. Then if $f'(x)$ is a linear y-root of $P(x, y)$ it follows that $f'(x)/v(x)$ is a linear $\tilde{y}$-root of the reduced interpolation polynomial $\tilde{P}(x, \tilde{y})$. A *simplified interpolation* can be carried out to find $\tilde{P}(x, \tilde{y})$ which is much shorter than $P(x, y)$ since the degree k polynomial $v(x)$ has been factored out in advance.

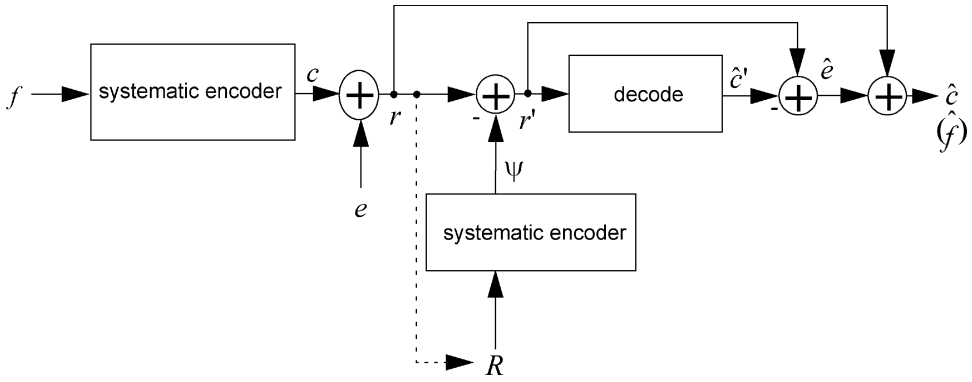

Figure 2. The decoding problem can be transformed into an easier one by reencoding.

To implement the simplified interpolation, consider the original set of polynomials, $G_1 = \{1, y, \ldots, y^{d_y}\}$. After applying the reencoding technique, the starting polynomial set for decoding is:

$$\begin{aligned} G' &= \{v(x)^m, v(x)^{m-1}y, \ldots, v(x)^{m-d_y}y^{d_y}\} \\ &= v(x)^m\left\{1, \frac{y}{v(x)}, \left(\frac{y}{v(x)}\right)^2, \ldots, \left(\frac{y}{v(x)}\right)^{d_y}\right\}. \end{aligned} \tag{33}$$

After a change of variables $\tilde{y} = y/v(x)$, we have $\tilde{G} = \{1, \tilde{y}, \ldots, \tilde{y}^{d_y}\}$. Note that the weighted degree of the new variable $\tilde{y}$ is:

$$\begin{aligned} \deg^{(1,k-1)}(\tilde{y}) &= \deg^{(1,k-1)}(y) - \deg^{(1,k-1)}(v(x)) \\ &= (k-1) - k \\ &= -1. \end{aligned} \tag{34}$$

The y-coordinates of the interpolation points need to be rescaled:

$$\tilde{y}_i = \frac{y_i'}{v(x_i)}, \tag{35}$$

where the y_i' are the y-coordinates of points after the translation in the reencoding step. Starting from $\tilde{G} = \{1, \tilde{y}, \ldots, \tilde{y}^{d_y}\}$, apply Algorithm 2 to the $O(n-k)$ rescaled points where the min function is with respect to the $(1, -1)$-weighted degree of the polynomials in x and $\tilde{y}$. A formal proof that simplified interpolation produces a correct result is given in [18]. $P(x, y)$ can be reconstructed according to (27) and the Roth-Ruckenstein algorithm can be applied to find the linear y-roots. However, this means that the memory savings from using the reduced polynomials are lost in the root-finding step and there is extra work in multiplying out the polynomials. If this approach is taken then the candidate message polynomial $\hat{f}'(x)$ is reencoded to find a candidate codeword, $\hat{c}'$. Then subtract the candidate codeword from the transformed received word r' to get an estimate of the error vector $\hat{e}$ which can be added to the original received word r to get an estimate $\hat{c}$ of c. If a systematic encoder was used in the transmitter, the message symbols can be read off directly from $\hat{c}$. A block diagram of this scheme is given in Fig. 2.

A better strategy is to directly find $f'(x)/v(x)$ from the reduced polynomial $\tilde{P}(x, \tilde{y})$ as described in Section 3.2. Simplified interpolation also decreases the interpolation time since fewer terms need to be updated at each iteration.

3.1.4. Application to Soft-Decision Decoding. We have to be careful in the application of the simplified interpolation technique to soft-decision decoding. We would like to choose the k reencoding positions as the ones with the largest multiplicities to realize the greatest possible complexity reduction. In the best case there would be k reencoding positions, all with the maximum multiplicity m. Then we could apply (27). Even though there is no guarantee, it is quite a reasonable assumption that most of the time we will find k or more positions with maximum multiplicity, especially for higher SNRs (low error rates). For lower SNRs, we might come short of k. In this case we bump up the multiplicities of the next most reliable positions to the maximum multiplicities. Simulations show that the loss incurred from applying this heuristic is quite small. As an example, for a RS(63,55) code with $m = 4$ as shown in Fig. 3, the loss is about 0.1 dB for high frame-error-rates (FER $> 10^{-2}$) but is negligible for frame-error-rates lower than 10^{-2} which is the region of interest.

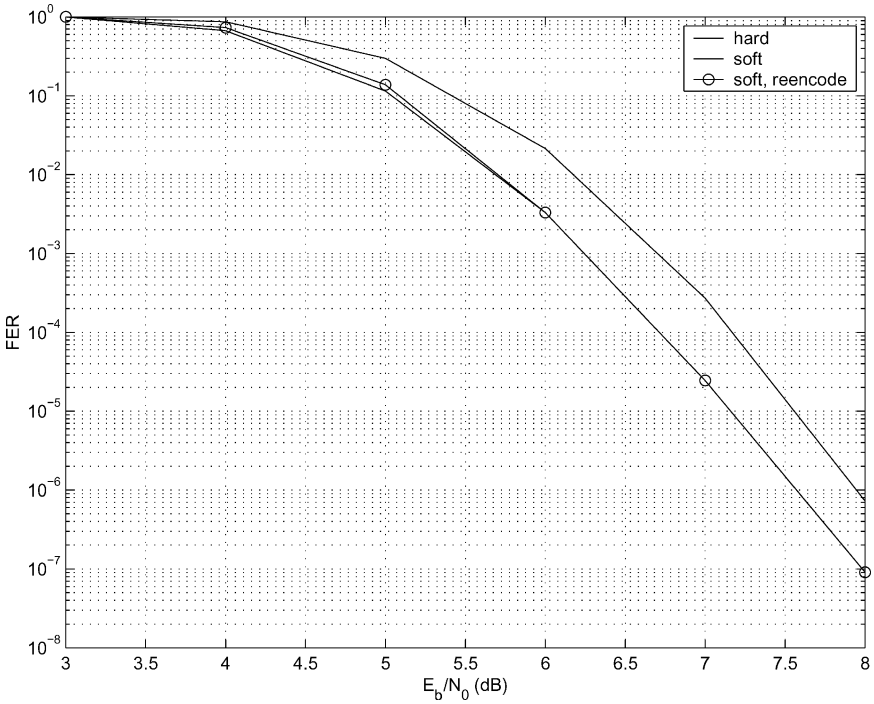

Figure 3. The loss due to forcing there to be k positions in the received hard decision with the maximum multiplicity, $m = 4$, for RS(63,55).

The following lemma gives the condition under which the simplified interpolation technique can be applied.

Lemma 2. *The maximum y-degree, d_y, of the bivariate polynomials in the simplified interpolation applied to a multiplicity matrix where there are at least k points with fixed multiplicity $m > 0$, is equal to m.*

Proof: Assume that $d_y < m$. Then from (11),

$$\left\lfloor \frac{1 + \sqrt{1 + \frac{8C_l}{k-1}}}{2} \right\rfloor < m + 1,$$

where $C_l = (l/2)(m)(m+1) \leq C_M$ is the cost of interpolation with l points of fixed multiplicity $m > 0$. For a real number a and integer b, if $\lfloor a \rfloor < b$ then $a < b$. Therefore

$$\frac{1 + \sqrt{1 + \frac{8C_l}{k-1}}}{2} < m + 1,$$

which implies that $l < k - 1$. But, $l \geq k$. This is a contradiction and the assumption that $d_y < m$ must be incorrect and therefore $d_y \geq m$. But, for the exponent in (27) to be nonnegative, we have $d_y \leq m$. Therefore $d_y = m$. □

The restriction that $d_y = m$ imposes limits on the rate and maximum multiplicity, however, the range of admitted values is quite large. For example, for the popular RS(255,239) code, $m \leq 27$, for RS(255,223), $m \leq 13$, and for RS(255,191), $m \leq 5$. The admissible values of k and m for $n = 255$ are plotted in Fig. 4.

With this assumption, reencoding is now straightforward for soft decoding. It is shown in [8] that the k columns of M that correspond to the reliable positions with maximum multiplicity only contain one entry with multiplicity m and all other entries in that column are zero. Intuitively this is because if a position is very reliable then there should only be one nonzero entry in the corresponding column of the multiplicity matrix. As the reliability decreases, other competing symbols with lower multiplicities arise. The $(n - k)$ unreliable positions may have several nonzero entries in their corresponding column in M. Therefore, the nonzero values from r' should be subtracted from the y-coordinates of all the corresponding points. As well, the division needed for the change of variables should be done for all the nonzero points in the column.

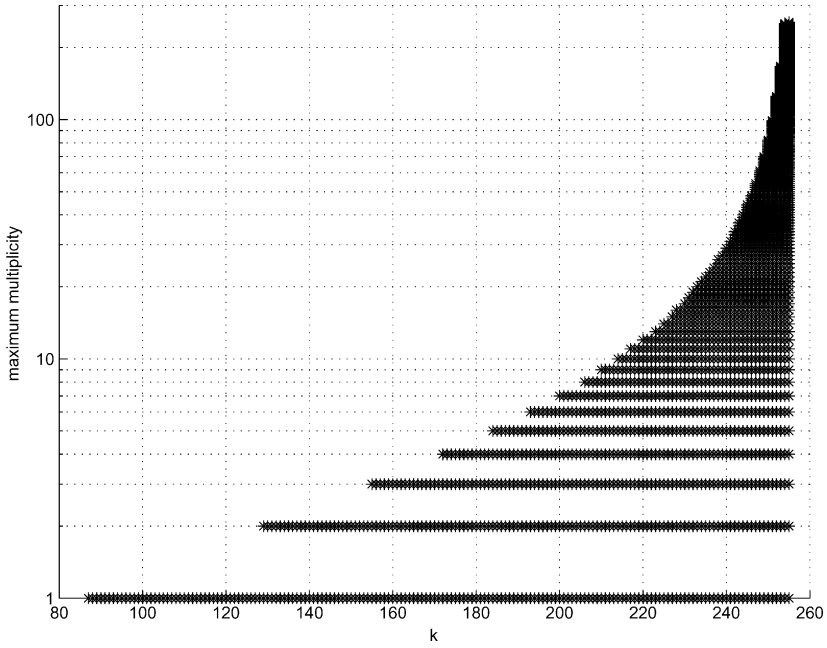

Figure 4. The admissible values of k and maximum multiplicity m for a code of length 255. A star is plotted if the point (k, m) results in $d_y = m$.

3.1.5. Example of Soft-Decision Decoding with Simplified Interpolation.

Example 4. We will use the same code as in Example 1, a (7, 5) Reed-Solomon code over GF(8) and transmit the same message:

$$f = (\alpha, \alpha^4, \alpha^4, \alpha^3, 1).$$

This time however we use a systematic encoder to obtain the codeword:

$$c = (\alpha^6, 1, \alpha, \alpha^4, \alpha^4, \alpha^3, 1).$$

Note that the message appears as the last five symbols in the codeword and the first two symbols are the parity symbols. After transmission through an AWGN channel with BPSK modulation, let the input to the decoder be the reliability matrix:

$$\Pi = \begin{bmatrix} 0.000000 & 0.005232 & 0.002588 & 0.009617 & 0.006731 & 0.003812 & 0.187596 \\ 0.000305 & 0.966177 & 0.000769 & 0.099860 & 0.005083 & 0.000007 & 0.758324 \\ 0.000004 & 0.000001 & 0.729645 & 0.011815 & 0.001246 & 0.000031 & 0.001307 \\ 0.003102 & 0.000272 & 0.216797 & 0.122692 & 0.000941 & 0.000000 & 0.005284 \\ 0.000102 & 0.000152 & 0.000137 & 0.029798 & 0.474019 & 0.986393 & 0.009353 \\ 0.089021 & 0.028157 & 0.000041 & 0.309429 & 0.357984 & 0.001834 & 0.037808 \\ 0.001039 & 0.000000 & 0.038565 & 0.036611 & 0.087737 & 0.007908 & 0.000065 \\ 0.906428 & 0.000008 & 0.011459 & 0.380177 & 0.066260 & 0.000015 & 0.000263 \end{bmatrix}.$$

Applying Algorithm 1 with $s = 12$ gives the multiplicity matrix:

$$M = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 & 2 \\ 0 & 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix},$$

which can be condensed by removing the nonzero entries into a list of interpolation points and their

multiplicities:

x	y	Multiplicity
1	α^6	2
α	1	2
α^2	α	2
α^3	α^6	1
α^4	α^3	1
α^5	α^3	2
α^6	1	2

Examining the list of points we see that the hard decision contains two errors. Now we will apply the reencoding technique to reduce the list of interpolation points. Checking the interpolation list we do find $k = 5$ distinct x-values that have the maximum multiplicity $m = 2$. These "reliable" points will be eliminated from the interpolation list after reencoding and we will only have to do a bivariate interpolation through the remaining $(n - k) = 2$ "unreliable" points. We choose the reencoding points: $\{(1, \alpha^6), (\alpha, 1), (\alpha^2, \alpha), (\alpha^5, \alpha^3), (\alpha^6, 1)\}$. The systematically reencoded codeword is then:

$$\psi = (\alpha^6, 1, \alpha, \alpha^4, \alpha^4, \alpha^3, 1).$$

Subtracting, we find the reencoded interpolation list:

x	y	Multiplicity
α^3	α^3	1
α^4	α^6	1

When implementing a decoder, clearly we only have to explicitly do the addition for the $(n - k)$ unreliable positions since the k systematic "reliable" positions of r' will always be 0. To calculate $v(x)$, perform the univariate interpolation:

$$\begin{aligned} v(x) &= (x - 1)(x - \alpha)(x - \alpha^2)(x - \alpha^5)(x - \alpha^6) \\ &= x^5 + \alpha^6 x^4 + \alpha^4 x^3 + \alpha^4 x^2 + \alpha^6 x + 1. \end{aligned}$$

The change of variables is performed by scaling the points to get:

x	y	Multiplicity
α^3	$\frac{\alpha^3}{v(\alpha^3)} = \alpha^5$	1
α^4	$\frac{\alpha^6}{v(\alpha^4)} = \alpha^2$	1

The interpolation is performed as before with $d_y = 2$ but only $C' = 2$ iterations are required. The interpolation polynomial in $\tilde{y}$ is:

$$\tilde{P}(x, \tilde{y}) = \alpha^3 x \tilde{y}^2 + \alpha^6 \tilde{y} + \alpha^5 \tilde{y}^2.$$

After the factorization stage, the output list contains the two candidates:

$$\begin{aligned} \hat{c}_1 &= (\alpha^6, 1, \alpha, \alpha^4, \alpha^4, \alpha^3, 1), \\ \hat{c}_2 &= (\alpha^5, 1, \alpha^4, \alpha^6, \alpha^3, \alpha^3, 1). \end{aligned}$$

We choose $\hat{c}_1$ after calculating its probability using the reliability matrix. The message is read off directly from $\hat{c}_1$ and we have:

$$\hat{f} = (\alpha, \alpha^4, \alpha^4, \alpha^3, 1).$$

3.1.6. Memory Savings and Speedup. To calculate the maximum number of terms in the reduced polynomials in the simplified interpolation consider the interpolation polynomial as expressed in (27). The degree of $v(x)$ is k and therefore a monomial in $P(x, y)$ has the form $x^{i+(m-j)k} y^j$. Since the $(1, k-1)$-weighted degree of $P(x, y)$ is at most d_x, we have $i \leq d_x - mk + j$. Therefore, there are $d_x - mk + j + 1$ terms with a y-degree of j. Summing up over all values of j, the maximum number of terms of the reduced polynomial is:

$$\begin{aligned} \tilde{N}_{\text{mon}} &= \sum_{j=0}^{d_y} (d_x - mk + 1 + j) \\ &= (d_y + 1)\left(d_x - mk + 1 + \frac{d_y}{2}\right). \end{aligned} \quad (36)$$

From [3], the maximum length of the polynomials in the original interpolation algorithm is:

$$N_{\text{mon}} = \left(d_y + 1\right)\left(d_x - \frac{d_y(k-1)}{2} + 1\right). \quad (37)$$

The memory compression ratio is therefore $\Lambda = N_{\text{mon}}/\tilde{N}_{\text{mon}}$. For simplified interpolation, $d_y = m$, and

from (12) the maximum value of the $(1, k-1)$-weighted degree is,

$$d_x = \left\lfloor \frac{C}{d_y + 1} + \frac{d_y(k-1)}{2} \right\rfloor = \left\lfloor \frac{m(n+k-1)}{2} \right\rfloor. \tag{38}$$

Then the maximum length of the unsimplified polynomials is:

$$N_{\text{mon}} \approx C + m + 1, \tag{39}$$

where the approximation is exact if m is even or $(n-k)$ is odd. Similarly, the maximum length of the reduced polynomial is

$$\tilde{N}_{\text{mon}} \approx C' + m + 1, \tag{40}$$

where the approximation is exact if m is even or $(n-k)$ is odd. Therefore the compression ratio is

$$\Lambda \approx \frac{C + m + 1}{C' + m + 1} = O\left(\frac{n}{n-k}\right). \tag{41}$$

The complexity of the algorithm in terms of the number of arithmetic operations is then reduced by the combined effects of fewer iterations and shorter polynomials. The speedup is:

$$\Gamma = O\left(\left(\frac{n}{n-k}\right)^2\right). \tag{42}$$

There will be some overhead of calculating $v(x)$ and checking for and ensuring the k reliable points but this overhead will be worth it to achieve the overall savings.

3.2. Reduced-Complexity Factorization

The savings realized by simplified interpolation can be carried over into the factorization procedure by applying the Roth-Ruckenstein algorithm directly to the reduced polynomial $\tilde{P}(x, \tilde{y})$ as proposed in [18, 19, 27]. If the message polynomial corresponding to $c' = (c - \psi)$ is a linear y-root of $P(x, y)$ then $f'(x)/v(x)$ is a linear $\tilde{y}$-root of $\tilde{P}(x, \tilde{y})$, or

$$\tilde{P}(x, \tilde{y}) = \left(\tilde{y} - \frac{f'(x)}{v(x)}\right) B(x, \tilde{y}). \tag{43}$$

Applying the Roth-Ruckenstein algorithm directly to the reduced polynomial we obtain a sequence $\{s_0, s_1, \ldots, s_{l-1}\}$ which are the coefficients of

$$s(x) = \frac{f'(x)}{v(x)}, \tag{44}$$

which is a polynomial approximation to the rational function $f'(x)/v(x)$. As we will see below, the number of coefficients, l, does not have to be very large and is much smaller than the k coefficients required in the unsimplified Roth-Ruckenstein procedure.

The transformed received hard-decision word is $r' = c' + e$, which has zeroes in the k reencoded positions, R_k. Therefore, in the k reencoded positions, $c'_i = -e_i$, $i \in R_k$, or,

$$f'(\alpha^i) = -e_i, \quad i \in R_k. \tag{45}$$

If there is no error in position $i \in R_k$ then $e_i = 0$ and $f'(\alpha^i) = 0$. Therefore $(x - \alpha^i)$ is a root of $f'(x)$, or $f'(x) = (x - \alpha^i)D(x)$. Considering all the error-free positions in R_k,

$$f'(x) = \prod_{i \in R_k \text{ s.t. } e_i = 0} (x - \alpha^i)\Omega(x). \tag{46}$$

Therefore,

$$s(x) = \frac{f'(x)}{v(x)} = \frac{\prod_{i \in R_k \text{ s.t } e_i = 0}(x - \alpha^i)\Omega(x)}{\prod_{i \in R_k}(x - \alpha^i)} = \frac{\Omega(x)}{\prod_{i \in R_k \text{ s.t. } e_i \neq 0}(x - \alpha^i)}. \tag{47}$$

The denominator is an error-locator polynomial for the k positions in R_k. Given the "syndrome" polynomial $s(x)$, we can use the Berlekamp-Massey algorithm (BMA) to reconstruct the rational function $\Omega(x)/\Lambda(x)$ where $\Lambda(x)$ is an error-locating polynomial for the k positions in R_k and $\Omega(x)$ is an error-evaluator polynomial for the k positions in R_k. The roots of $\Lambda(x)$ give the error locations in R_k. This technique only finds errors in the set of k reliable positions and not in the $(n-k)$ unreliable positions. To correct any errors in the $(n-k)$ unreliable positions, we can do a systematic reencoding in k arbitrary positions using the erasures-only decoder that is already implemented for the reencoding step. In fact, in a VLSI implementation, a single errors and erasures BMA can be implemented on the chip

and used for three purposes: systematic reencoding, reconstructing the rational function and as a first pass hard-decision decoder in the redecoding architectures proposed in [8, 28].

We are only directly correcting errors in the k reliable positions. Fortunately, most errors are likely in the $(n-k)$ unreliable positions so we only need to correct a small number of errors and hence only need a few coefficients in the syndrome sequence. This greatly speeds up the Roth-Ruckenstein algorithm. As a rule of thumb we use $l = 2\lceil (k/n)t \rceil$ coefficients where $t = \lfloor (n-k)/2 \rfloor$ is the classical error-correcting capability. To get the error values, recall from (45) that $e_i = -f'(\alpha^i), i \in R_k$. We have,

$$\begin{aligned} \frac{f'(x)}{v(x)} &= \frac{\Omega(x)}{\Lambda(x)} \\ f'(x) &= \frac{\Omega(x)v(x)}{\Lambda(x)}. \end{aligned} \tag{48}$$

When evaluating $f'(x)$ at x-values corresponding to the error locations, $v(\alpha^i) = 0$ and $\lambda(\alpha^i) = 0$ when i is an error position in R_k. Using L'Hôpital's rule the error-evaluation formula becomes:

$$f'(\alpha^i) = \frac{\Omega(\alpha^i)v^{(1)}(\alpha^i)}{\Lambda^{(1)}(\alpha^i)}, \tag{49}$$

for the x-values α^i corresponding to error positions in R_k, where $v^{(1)}(x)$ and $\Lambda^{(1)}(x)$ are the formal derivatives of $v(x)$ and $\Lambda(x)$. Notice that we have directly found an estimate of the error vector, $\hat{e}$, without having to subtract off r'. The estimated codeword can be found by adding $\hat{e}$ to the received word r and the message can be read off directly if a systematic encoder was used. A block diagram describing the fully simplified algorithm is given in Fig. 5.

The polynomials in the Roth-Ruckenstein algorithm can grow by d_y terms every time the routine is called recursively. Since we find l coefficients, the maximum length polynomial required is

$$\begin{aligned} \tilde{N}_{\text{mon,RR}} &= \tilde{N}_{\text{mon}} + l(d_y)(d_y+1) \\ &= \tilde{N}_{\text{mon}} + l(m^2+m). \end{aligned} \tag{50}$$

Table 2. The decoding times of the component algorithms for a test case of decoding a RS(255,239) code with $m = 4$.

Algorithm	Time
Soft front end	18 μs
Reencoding	94 μs
Interpolation	788 μs
Root finding	390 μs
Output	60 μs
Total	1.37 ms

3.3. A Fast Software Implementation

We implemented the simplified interpolation and factorization algorithms into our software program. For the same test case as in Example 2, the total decoding time with the simplified algorithms is 1.37 ms. The contributions of the various components of the algorithm are summarized in Table 2. Note that "reencoding" in the table refers to all the steps required for the reencoding including the partition by reliability, systematic encoding, calculation of $v(x)$ and scaling the input. The "output" entry in the table refers to the output Berlekamp-Massey algorithm as well as the systematic encoder. There is some increased overhead required, but it is worth it to realize the overall savings brought by the simplified algorithms. From the table, the speedup of the bivariate interpolation algorithm is 573 times which is on the order of that predicted by (42). The number of Galois field additions in the entire

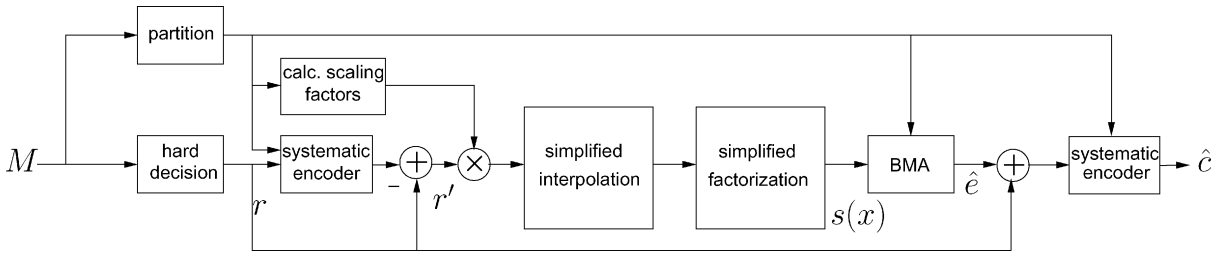

Figure 5. Simplified interpolation and factoring.

Table 3. The decoding times, throughput, and speedup for decoding a RS(255,k) code using simplified interpolation and factoring with various choices of k and m. All test cases were uncorrectable by hard-decision decoders.

k	m	Time (ms)	Throughput (kbps)	Speedup
191	4	15.2	100	33
223	4	6.72	265	86
223	13	413	4.3	251
239	4	1.37	1400	396
239	16	250.7	7.6	953

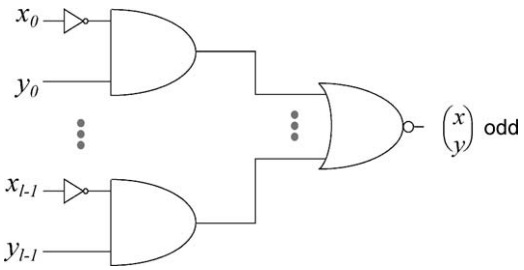

Figure 6. Circuit for implementing the parity of the binomial coefficient $C(x, y)$.

decoder was measured to be 1.7×10^5 and the number of Galois field multiplications was 2.5×10^5, the total of which is 0.17% of the Galois field operations in the unsimplified decoder.

The throughput has been increased to 1.4 Mbps, which is approaching real-time decoding rates. In particular, this decoder implementation is a good candidate to be used as part of the redecoding architecture described in [8, 28]. Table 3 gives decoding times for a range of code rates and values of m.

4. VLSI Architecture

Calculating the Hasse derivative (HD) of a bivariate polynomial is a computationally demanding task that has the flavor of polynomial evaluation. We make the observation that in fields of characteristic two, the product

$$H = \binom{a}{\alpha}\binom{b}{\beta} p_{a,b} x_j^{a-\alpha} y_i^{b-\beta}$$

is nonzero only if $a \geq \alpha, b \geq \beta$, and both $\mathrm{C}(a, \alpha) = \binom{a}{\alpha}$ and $\mathrm{C}(b, \beta) = \binom{b}{\beta}$ are odd. The parity of the binomial coefficients can be calculated easily by Lucas' theorem [17, 29]. If we write the integers x and y in their binary representations, then $\mathrm{C}(x, y)$ is odd and $x \geq y$ if x AND y is equal to y. A simple circuit for calculating the parity of the binomial coefficients is shown in Fig. 6.

Horner's rule [30] for the fast evaluation of polynomials can be used to calculate the HD if the terms of the polynomial are read out in order of decreasing weight. If a sparse polynomial representation is used, exponentiation can be implemented efficiently using the right-to-left binary exponentiation algorithm (RLBE) [30]. The exponent N can be written in its binary representation $N = (N_{L-1}, N_{L-2}, \ldots, N_1, N_0)$ where $N_{L-1} = 1$. In Fig. 7(a) we present an implementation of the RLBE algorithm that executes in N_L clock cycles. Using Horner's rule, the input to the RLBE circuit is the exponent *difference*, which reduces the number of clock cycles. A circuit for implementing the HD is shown if Fig. 7(b). If $\mathrm{C}(a, \alpha)$ or $\mathrm{C}(b, \beta)$ for a partial product are odd then that partial product is zero and the exponentiation can be skipped.

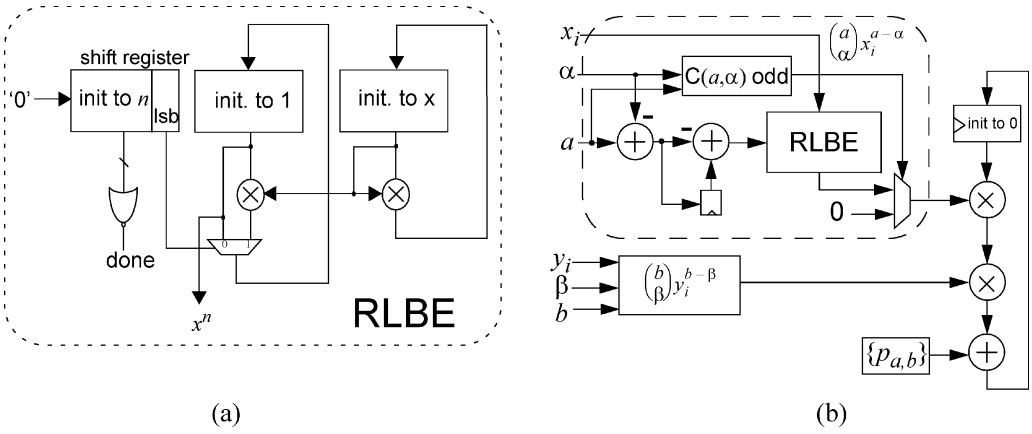

Figure 7. (a) Circuit for right-to-left binary exponentiation. (b) Circuit for calculating the Hasse derivative using Horner's rule.

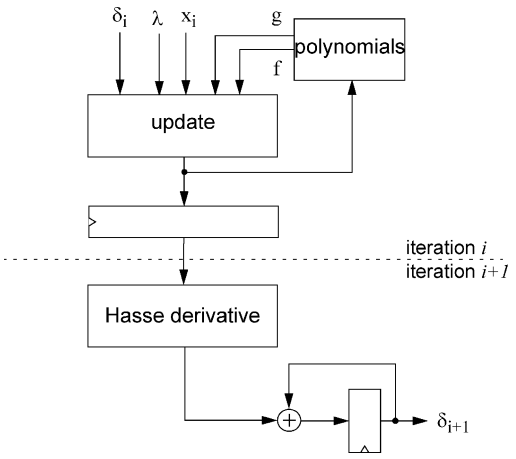

Figure 8. Pipelined architecture for concurrently updating the polynomials of iteration i and calculating the Hasse derivative for iteration $i+1$.

A pipelined architecture for the interpolation algorithm is shown if Fig. 8. There are two possible updates for a bivariate polynomial g:

1. $g \leftarrow \lambda g + \delta f$, where $\lambda, \delta \in \mathrm{GF}(2^{\omega})$ and f is a bivariate polynomial, or
2. $g \leftarrow x_i g + x g$, where $x_i \in \mathrm{GF}(2^{\omega})$.

The polynomial updates are calculated term-by-term. Once a term is updated, it is no longer needed in that iteration. Therefore, the part of the next iteration's Hasse derivative calculation involving that term can be performed. This architecture calculates the polynomial update for iteration i concurrently with the HD calculation for iteration $i+1$.

5. Conclusions

The Koetter-Vardy (KV) soft-decision decoding algorithm is an extension of the Guruswami-Sudan (GS) list-decoding algorithm for Reed-Solomon codes. The KV algorithm seems fairly complex for VLSI implementation because of the large number of iterations in the interpolation step. We explored algorithmic techniques for the efficient implementation of interpolation-based decoders. The basic idea is to reencode the received hard-decision to produce a transformed problem that is easier to solve than the original problem. The result is that the soft information only influences the interpolation procedure through the $(n-k)$ least reliable symbol positions, which for a high-rate code is much smaller than n. We also showed that the length of the polynomials can be significantly reduced, enabling efficient implementations. These ideas were demonstrated in a software implementation that for a moderate complexity level decodes at over 1 Mbps. We have also presented an efficient way to implement the Hasse derivative in a VLSI implementation. By putting these techniques together, we envision efficient VLSI implementations of the Koetter-Vardy algorithm.

Acknowledgment

The authors would like to thank the anonymous referees for comments that helped improve the clarity of the paper. The authors would also like to thank Rasmus Refslund Nielsen, Xinmiao Zhang, and Jun Ma for helpful discussions.

References

1. S.B. Wicker and V.K. Bhargava, "An Introduction to Reed-Solomon Codes," in *Reed-Solomon Codes and Their Applications* S.B. Wicker and V.K. Bhargava, (Eds.), New York, New York: IEEE Press, 1994, ch. 1, pp. 1–16.
2. M. Sudan, "Decoding of Reed-Solomon Codes Beyond the Error Correction Bound," *Journal of Complexity*, vol. 13, no. 1, 1997, pp. 180–193.
3. V. Guruswami and M. Sudan, "Improved Decoding of Reed-Solomon and Algebraic-Geometry Codes," *IEEE Transactions on Information Theory*, vol. 45, 1999, pp. 1757–1767.
4. A. Brinton Cooper III, "Soft Decision Decoding of Reed-Solomon Codes," *Reed-Solomon Codes and Their Applications*, New York, New York: IEEE Press, 1994, ch. 6, pp. 108–124.
5. R. Kotter and A. Vardy, "Algebraic Soft-Decision Decoding of Reed-Solomon Codes," in *Proceedings of the IEEE International Symposium on Information Theory*, 2000, p. 61.
6. R. Koetter and A. Vardy, "Algebraic Soft-Decision Decoding of Reed-Solomon Codes," *IEEE Transactions on Information Theory*, vol. 49, November 2003, pp. 2809–2825.
7. W.J. Gross, F.R. Kschischang, R. Koetter, and P. Gulak, "Simulation Results for Algebraic Soft-Decision Decoding of Reed-Solomon Codes," in *Proceedings of the 21'st Biennial Symposium on Communications*, Queen's University, Kingston, Ontario, Canada, June 2–5 2002, pp. 356–360.
8. W.J. Gross, F.R. Kschischang, R. Koetter, and P. Gulak, "Applications of Algebraic Soft-Decision Decoding of Reed-Solomon Codes," Submitted to IEEE Transactions on Communications, 2003.
9. I.S. Reed and G. Solomon, "Polynomial Codes Over Certain Finite Fields," *SIAM Journal of Applied Mathematics*, vol. 8, 1960, pp. 300–304.
10. H. Hasse, "Theorie der höheren differentiale in einem algebraischen funktionenkörper mit vollkommenem konstantenkörper

bei beliebiger charakteristik," *J. Reine. Ang. Math.*, vol. 175, 1936, pp. 50–54.
11. R.M. Roth and G. Ruckenstein, "Efficient Decoding of Reed-Solomon codes Beyond Half the Minimum Distance," *IEEE Transactions on Information Theory*, vol. 46, Jan. 2000, pp. 246–257.
12. H.M. Möller and B. Buchberger, "The Construction of Multivariate Polynomials with Preassigned Zeros," in *EUROCAM '82, European Computer Algebra Conference*, J. Calmet, (Ed.), vol. 144 of *Lecture Notes In Computer Science*, Marseille, France, April 1982, pp. 24–31.
13. J. Abbott, A. Bigatti, M. Kreuzer, and L. Robbiano, "Computing Ideals of Points," *Journal of Symbolic Computation*, vol. 30, no. 4, 2000, pp. 341–356.
14. G. Feng and K. Tzeng, "A Generalization of the Berlekamp-Massey Algorithm for Multisequence Shift-Register Synthesis with Applications to Decoding Cyclic Codes," *IEEE Transactions on Information Theory*, vol. 37, Sept. 1991, pp. 1274–1287.
15. R. Kötter, "On Algebraic Decoding of Algebraic-Geometric and Cyclic Codes," PhD Thesis, Lynköping University, 1996.
16. R.R. Nielsen, "Decoding AG-Codes Beyond Half the Minimum Distance," Master's Thesis, Technical University of Denmark, Aug. 31 1998.
17. W.J. Gross, F.R. Kschischang, R. Koetter, and P. Gulak, "A VLSI Architecture for Interpolation in Soft-Decision List Decoding of Reed-Solomon Codes," in *Proceedings of the 2002 IEEE Workshop on Signal Processing Systems (SIPS'02)*, San Diego, CA, Oct. 16–18 2002, pp. 39–44.
18. R. Koetter, J. Ma, A. Vardy, and A. Ahmed, "Efficient Interpolation and Factorization in Algebraic Soft-Decision Decoding of Reed-Solomon Codes," in *Proceedings of the IEEE International Symposium on Information Theory*, June 29–July 4, 2003, p. 365.
19. R. Koetter and A. Vardy, "A Complexity Reducing Transformation in Algebraic List Decoding of Reed-Solomon Codes," in *Proceedings of ITW2003*, Paris, France, March 31–April 4, 2003.
20. L.R. Welch and E.R. Berlekamp, "Error Correction for Algebraic Block Codes," US Patent 4,633,470, 1986.
21. R.E. Peile, "On the Performance and Complexity of a Generalized Mimimum Distance Reed-Solomon Decoding Algorithm," *International Journal of Satellite Communications*, vol. 12, 1994, pp. 333–359.
22. D. Dabiri and I.F. Blake, "Fast Parallel Algorithms for Decoding Reed-Solomon Codes," in *Proceedings of the 1994 IEEE International Symposium on Information Theory*, June 27, 1994, p. 97.
23. D. Dabiri and I.F. Blake, "Fast Parallel Algorithms for Decoding Reed-Solomon Codes Based on Remainder Polynomials," *IEEE Transactions on Information Theory*, vol. 41, July 1995, pp. 873–885.
24. E. Berlekamp, "Bounded Distance +1 Soft-Decision Reed-Solomon Decoding," *IEEE Transactions on Information Theory*, vol. 42, 1996, pp. 704–720.
25. S.B. Wicker, *Error Control Systems for Digital Communication and Storage*, Upper Saddle River, New Jersey: Prentice Hall, 1995.
26. B. Kamali and P. Morris, "Application of Erasure-Only Decoded Reed-Solomon Codes in Cell Recovery for Congested ATM Networks," in *IEEE Vehicular Technology Conference*, vol. 2, 24–28 Sept. 2000, pp. 983–986.
27. A. Ahmed, R. Koetter, and N.R. Shanbhag, "VLSI Architectures for Soft-Decision Decoding of Reed-Solomon Codes," Submitted to *IEEE Transactions on VLSI Systems*, Feb. 2003.
28. H. Xia, H. Song, and J.R. Cruz, "Retry Mode Soft Reed-Solomon Decoding," *IEEE Transactions on Magnetics*, vol. 38, 2002, pp. 2325–2327.
29. V.C. da Rocha Jr., "Digital Sequences and the Hasse Derivative," in *Communications Coding and Signal Processing*, B. Honary, M. Darnell, and P. Farrell (Eds.), Communication Theory and Applications, John Wiley and Sons Inc., 1997, vol. 3, pp. 256–268.
30. D.E. Knuth, *The Art of Computer Programming*, Seminumerical Algorithms, Addison-Wesley, 1969, vol. 2.

Warren J. Gross received the B.A.Sc. degree from the University of Waterloo, Waterloo, Ontario, Canada, in 1996, and the M.A.Sc. and Ph.D. degrees from the University of Toronto, Toronto, Ontario, Canada, in 1999 and 2003, respectively, all in electrical engineering.

From 1993 to 1996, he worked in the area of space-based machine vision at Neptec Design Group, Ottawa, Ontario, Canada. He is an Assistant Professor with the Department of Electrical and Computer Engineering, McGill University, Montreal Quebec, Canada, where he has been a faculty member since 2003. During the summer of 2004 he was a Visiting Professor at the University of South Brittany, Lorient, France. His research interests are in the design and applications of signal processing microsystems, VLSI design, and computer architecture.

Dr. Gross is a recipient of the Natural Sciences and Engineering Research Council of Canada postgraduate scholarship, the Walter Summer fellowship and the Government of Ontario/Ricoh Canada Graduate Scholarship in Science and Technology.
wjgross@ece.mcgill.ca

Frank R. Kschischang is a Professor and Canada Research Chair in the Department of Electrical and Computer Engineering at the University of Toronto, where he has been a faculty member since 1991.

He received the B.A.Sc. degree with honours from the University of British Columbia in 1985, and M.A.Sc. and Ph.D. degrees from the University of Toronto in 1988 and 1991, respectively, all in electrical engineering. During 1997–98, he spent a sabbatical year as a visiting scientist at the Massachusetts Institute of Technology.

Dr. Kschischang's research interests are focused on the area of coding techniques primarily on soft-decision decoding algorithms, trellis structure of codes, codes defined on graphs, and iterative decoders. He is a recipient of the Ontario Premier's Research Excellence Award.

From Oct. 1997 to Oct. 2000. Dr. Kschischang served as IEEE Transactions on Information Theory Associate Editor for Coding Theory. He was general co-chair and organizer of the 1997 Canadian Workshop on Information Theory, and will serve as technical program co-chair for the 2004 IEEE International Symposium on Information Theory.

Ralf Koetter received a Diploma in Electronical Engineering from the Technical University Darmstadt, Germany in 1990 and a Ph.D. degree from the Department of Electrical Engineering at Linköping University, Sweden. From 1996 to 1998, he was a visiting scientist at the IBM Almaden Research Lab., San Jose, California. Dr. Koetter was a Visiting Assistant Professor at the University of Illinois at Urbana/Champaign and Visiting Scientist at CNRS in Sophia Antipolis, France. He joined the faculty of the University of Illinois at Urbana/Champaign in 1999 and is currently an Associate Professor at the Coordinated Science Laboratory at the University.

Dr Koetter's research interests include coding and information theory and their application to communication systems.

In the years 1999–2001, he served as associate editor for coding theory & techniques for the IEEE Transactions on Communications. In 2000, he started a term as associate editor for coding theory of the IEEE Transactions on Information Theory. He received an IBM Invention Achievement Award in 1997, an NSF CAREER Award in 2000, and an IBM Partnership Award in 2001. He is a member of the Board of Governors of the IEEE Information Theory Society.

P. Glenn Gulak received his Ph.D. from the University of Manitoba, Winnipeg, Manitoba, Canada. From 1985 to 1988 he was a Research Associate in the Information Systems Laboratory and the Computer Systems Laboratory at Stanford University. Currently, Dr. Gulak is a Professor in the Department of Electrical and Computer Engineering at the University of Toronto, Toronto, Ontario, Canada, and holds the L. Lau Chair in Electrical and Computer Engineering. His research interests are in the areas of memory design, circuits, algorithms and VLSI architectures for digital communications. He is a senior member of the IEEE and a registered professional engineer in the province of Ontario. He has received several teaching awards for undergraduate courses taught in both the Department of Computer Science and the Department of Electrical and Computer Engineering at the University of Toronto. He has served on the ISSCC Signal Processing Technical Subcommittee since 1990 and served as the Technical Program Chair for ISSCC 2001.