VLSI ARCHITECTURES FOR SOFT-DECISION DECODING OF REED-SOLOMON CODES

Arshad Ahmed, Ralf Koetter, and Naresh R. Shanbhag

Coordinated Science Laboratory, ECE Dept.
University of Illinois at Urbana-Champaign
1308, West Main Street,Urbana, IL 61801.
Email: [ahmed4, koetter, shanbhag]@uiuc.edu

ABSTRACT

In this paper, we present architectures for bivariate polynomial interpolation and factorization; the two main steps in algebraic soft-decision decoding of Reed-Solomon codes. We present an efficient formulation of the interpolation algorithm in which dependencies among the discrepancy coefficient computations are utilized to reduce interpolation complexity. Interpolation and factorization complexity is also reduced by using an FFT-like formulation for univariate polynomial translation. The modifications required to incorporate recently proposed algorithm level modifications for efficient interpolation and factorization are also presented. We determine the latency and hardware requirements for soft-decoding a $[255, 239]$ Reed-Solomon code using the proposed architectures.

1. INTRODUCTION

Reed-Solomon codes are error-correcting codes of choice in diverse applications such as magnetic and optical data storage, wireline and wireless communications and in satellite communications. Their widespread use is primarily due to a large burst error-correcting capability and due to the existence of efficient decoding algorithms. Traditional Reed-Solomon decoders are *bounded minimum distance* (BMD) decoders; they decode errors upto half the minimum distance of the code. Efficient algorithms such as the Berlekamp-Massey algorithm (BMA) and the Euclid algorithm [1] exist for BMD decoding. As opposed to BMD decoders, *list decoders* determine all codewords within a specified distance from a received vector. The decoder output is a *list* of possible transmitted codewords since the specified distance, which represents the error correction capability, is usually chosen to be larger than half the minimum distance. Guruswami and Sudan [2] discovered a polynomial time list decoding algorithm which uses interpolation and factorization of bivariate polynomials in order to decode errors upto $\lfloor n - \sqrt{nk} \rfloor$ symbols for a $[n, k]$ code, where n is the codeword length and k is the number of data symbols per codeword. An extension of this algorithm by Koetter and Vardy [3], incorporates soft-information from the channel into the interpolation process. This results in a soft-input, hard-output, decoding algorithm with improved coding gains. However, interpolation and factorization, which form the core of list-decoding algorithms, are computationally intensive. In this paper, we propose efficient architectures for implementing both these steps. These architectures incorporate recently proposed algorithm level complexity reduction techniques [7].

In Section 2, we briefly overview the soft-decision decoding algorithm [3]. A computationally efficient interpolation algorithm and its corresponding architecture is presented in Section 3. Section 4 presents factorization architectures. In Section 5 we determine the hardware complexity and latency associated with soft-decoding a $[255, 239]$ RS code using the proposed architectures. Section 6 concludes the paper.

2. SOFT-DECISION DECODING OF RS CODES

In this section, we present an overview of the RS soft - decoding algorithm. List decoding algorithms and the algebraic soft-decoding algorithm are based on operations on bivariate polynomials. Basic definitions related to bivariate polynomials and list-decoding can be found in [3]. We outline the three main steps involved in the algebraic soft-decoding algorithm:

- **Multiplicity generation step**: For an $[n, k]$ RS-code, given the data received from the channel, determine non-trivial interpolation points $(x_i, y_{i,j})$ and their corresponding multiplicities $m_{i,j}$, where $0 \leq i < n$, $0 \leq j < s_i$, and s_i is the number of interpolation points associated with the i^{th} code symbol position.
- **Interpolation step**: Given the set of non-trivial interpolation points and their corresponding multiplicities, compute the non-trivial polynomial $Q_{min}(X, Y)$ of minimum $(1, k-1)$ -weighted degree that passes through each of the interpolation points $(x_i, y_{i,j})$ with corresponding multiplicity $m_{i,j}$. $Q_{min}(X, Y)$ is denoted the *minimal polynomial*.

This work was funded by NSF grant: CCR 00-73490

- **Factorization step:** Determine all factors of the minimal polynomial of the form $Y - f(X)$ with $deg[f(X)] < k$. Each $f(X)$ corresponds to a valid data polynomial.

The block diagram corresponding to the soft-decoder is shown in Fig. 1. In the Koetter-Vardy front-end, for each of the n code symbol positions, the probability of having received each of the possible 2^q symbols is computed as an $n \times 2^q$ *reliability matrix*; each row represents a code symbol position i and hence a code evaluation point x_i. The 2^q columns hold probabilities corresponding to each of the possible 2^q symbols.

The *multiplicity matrix* is determined from the reliability matrix using a greedy algorithm [3] constrained by the decoder computational complexity. The number of non-trivial (non-zero multiplicity) interpolation points associated with x_i is denoted as s_i. Further, we assume that the non-trivial interpolation points $(x_i, y_{i,j})$ for each index i are ordered with decreasing multiplicity. Details of the interpolation block and the factorization block are given in the next two sections.

3. INTERPOLATION ALGORITHM AND ARCHITECTURES

In this section, we present an efficient formulation of the interpolation algorithm and present architectures for implementing this version of interpolation. Section 3.1 contains details of the computationally efficient interpolation algorithm. In Section 3.2, we present architectures for discrepancy computation, which is a key step in interpolation. Modified architectures incorporating algorithm level re-encoding transformations [7], are presented in Section 3.3. In Section 3.4, we present architectures for polynomial update, which is the other key step in the interpolation algorithm.

3.1. The Interpolation Algorithm

As stated previously, interpolation consists of generating a bivariate polynomial with minimal $(1, k-1)$ weighted degree satisfying the required constraints. For codes with large block-length, we have $k \gg 1$. Hence, interpolation algorithms typically begin with a set of *candidate polynomials* $Q_l(X, Y)$ initialized to all the Y degrees that may be required in the minimal polynomial. Given the number of constraints to be solved, the highest Y degree required in the minimal polynomial is determined from the $(1, k-1)$ weighted lexicographical ordering of bivariate monomials. The candidate polynomials are then iteratively updated to satisfy the constraints while ensuring that the increase in their weighted degree is minimal at each step. The two key steps in the interpolation process are:

- **Discrepancy Computation step**: In this step, the discrepancy coefficients corresponding to a particular constraint are computed for each candidate polynomial.

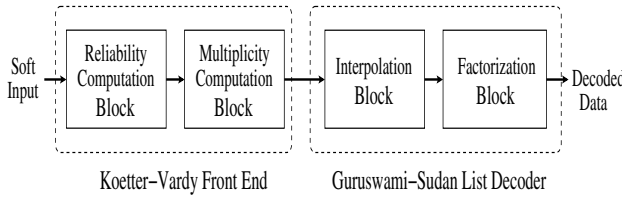

Fig. 1. Block diagram of soft-decision decoder.

- **Polynomial Update step**: Based on the computed discrepancy coefficients, the candidate polynomials are updated so that the discrepancy coefficients are reduced to zero.

Since the allowed Y degrees in the candidate polynomials are determined at initialization, every candidate polynomials can be represented as:

$$Q_l(X, Y) = \sum_{b=0}^{t} \sigma^{(l,b)}(X) Y^b \qquad 0 \le l \le t \qquad (1)$$

where $t+1$ is the total number of candidate polynomials and $\sigma^{(l,b)}(X)$ is a polynomial in X defined as:

$$\sigma^{(l,b)}(X) = \sigma_0^{(l,b)} + \sigma_1^{(l,b)} X + \cdots + \sigma_{w_{l,b}}^{(l,b)} X^{w_{l,b}} \qquad (2)$$

where $w_{l,b}$ is the degree of $\sigma^{(l,b)}(X)$.

An interpolation algorithm is presented in [4] in which the two steps of discrepancy coefficient computation and polynomial update are performed sequentially for each constraint corresponding to an interpolation point. The formula for computing discrepancy coefficients [4] is given by:

$$d_{\alpha\beta}^{(l)} = \sum_{b \ge \beta}^{t} \sum_{a \ge \alpha}^{w_{l,b}} \binom{a}{\alpha} \binom{b}{\beta} \sigma_a^{(l,b)} x_i^{a-\alpha} y_{i,j}^{b-\beta} \quad 0 \le l \le t \quad (3)$$

From a implementation perspective, (3) can be evaluated using Horner's formula to compute the exponentiation terms; this requires that the circuit cycle through the powers sequentially even if the binomial term is zero. Another approach is to compute only those powers with corresponding non-zero binomial terms [8]. This results in lower latency, but requires exponentiation circuits. Hence computing the discrepancy coefficients is either slow or expensive. Moreover, implementing interpolation as presented in [4] requires an iteration of discrepancy computation and polynomial update for each constraint.

In order to speed up the interpolation process, we introduce a modified form of the interpolation algorithm termed the **Point Serial Interpolation Algorithm (PSIA)**. In the **PSIA**, all the discrepancy coefficients corresponding to a particular interpolation point are computed in parallel. This step is followed by the polynomial update step for each of the constraints corresponding to that interpolation point. We will show that dependencies among the discrepancy coefficients of a interpolation point can be exploited to reduce their computation complexity and hence obtain a high-speed architecture.

Initialization:
$Q_0(X,Y) = 1; Q_1(X,Y) = Y; \cdots; Q_t(X,Y) = Y^t$
$Wdeg_0 = 0; Wdeg_1 = k-1; \ldots; Wdeg_t = t(k-1);$
Input: $(x_i, y_{i,j}, m_{i,j});\ 0 \le i < n; 0 \le j < s_i$
for $i = 0$ **to** $n-1$ and $j = 0$ **to** $s_i - 1$ **do**
Discrepancy Polynomial Computation (DPC) :
$D_{i,j}^{(l)}(X,Y) = \sum_{\alpha=0}^{m_{i,j}-1} \sum_{\beta=o}^{m_{i,j}-1-\alpha} d_{\alpha\beta} X^\alpha Y^\beta$
$(l = 0, 1, \cdots, t)$
Polynomial Update (PU) :
for $\alpha = 0$ **to** $m_{i,j} - 1$ and $\beta = 0$ **to** $m_{i,j} - 1 - \alpha$ **do**
PU0: $minl = \text{argmin}(Wdeg_k | d_{\alpha\beta}^{(k)} \neq 0, 0 \le k \le t)$
PU1: (l=0 **to** t and $l \neq minl$)
$Q_l(X,Y) \Leftarrow d_{\alpha\beta}^{(minl)} Q_l(X,Y) + d_{\alpha\beta}^{(l)} Q_{minl}(X,Y)$
$D_{i,j}^{(l)}(X,Y) \Leftarrow d_{\alpha\beta}^{(minl)} D_{i,j}^{(l)}(X,Y) + d_{\alpha\beta}^{(l)} D_{i,j}^{(minl)}(X,Y)$
PU2: ($l = minl$)
$Q_{minl}(X,Y) \Leftarrow Q_{minl}(X,Y) \times (X + x_i)$
$D_{i,j}^{(minl)}(X,Y) \Leftarrow D_{i,j}^{(minl)}(X,Y) \times X$
$Wdeg_{minl} \Leftarrow Wdeg_{minl} + 1$
endfor
endfor
Output: $Q_{minl}(X,Y); minl = \text{argmin}(Wdeg_k; 0 \le k \le t)$

Fig. 2. **The Point Serial Interpolation Algorithm (PSIA).**

In order to aid notation, the discrepancy coefficients are represented as a polynomial termed the *discrepancy polynomial*. The discrepancy polynomial $D_{i,j}^{(l)}(X,Y)$, associated with the l^{th} candidate polynomial for the interpolation point $(x_i, y_{i,j})$ can be written as:

$$\begin{aligned} D_{i,j}^{(l)}(X,Y) &= \sum_{\alpha < m_{i,j}} \sum_{\beta < m_{i,j}-\alpha} d_{\alpha\beta}^{(l)} X^\alpha Y^\beta \\ &= \sum_{\alpha < m_{i,j}} d_\alpha^{(l)}(Y) X^\alpha \end{aligned}$$

where $d_\alpha^{(l)}(Y) = \sum_{\beta=0}^{m_{i,j}-1-\alpha} d_{\alpha\beta}^{(l)} Y^\beta$.

The discrepancy polynomial consists of terms in the translated candidate polynomial $Q_l(X + x_i, Y + y_i)$, which have a degree less than $m_{i,j}$. We denote the process of translation followed by selection by :

$$D_{i,j}^{(l)}(X,Y) = [Q_l(X + x_i, Y + y_{i,j})]_{m_{i,j}} \qquad (4)$$

We will show that the complexity of computing the discrepancy polynomial is similar to that of computing a single discrepancy coefficient. Hence, the latency and complexity of discrepancy computation is reduced by a factor close to $\binom{m_{i,j}+1}{2}$. A complete pseudo-code corresponding to the **PSIA** is presented in Fig. 2 and the corresponding hardware block diagram is shown in Fig. 4. In Fig. 2, we see that there are two types of polynomial manipulations involved in the **PU** step. Step **PU1**, which involves the addition of two polynomials after proper scaling, is a *degree preserving step*, i.e, the degree of the candidate polynomial does not increase in this step. Step **PU2**, in which a candidate polynomial is multiplied with $(X + x_i)$, is a *degree enhancing step*; the degree of the candidate polynomial increases by unity. In steps **PU1** and **PU2**, the discrepancy polynomials $D_{i,j}^{(l)}(X,Y)$ are updated along with the candidate

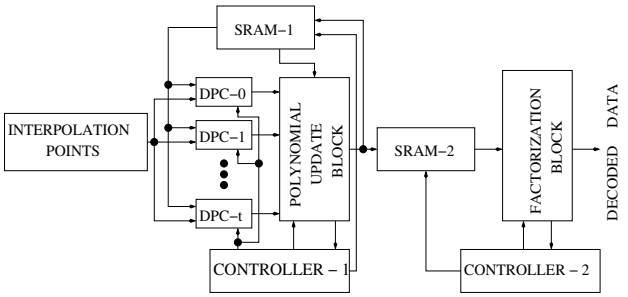

Fig. 4. Block diagram of the interpolation and factorization architecture.

polynomials. While this adds to the update complexity, the update latency is unaffected as candidate polynomial update and discrepancy polynomial update proceed in parallel.

The details of the architectures for the discrepancy polynomial computation (**DPC**) block and the polynomial update (**PU**) block are presented in the next three subsections. The details of the factorization architecture (**FA**) are presented in the next section. The hardware requirements and latency of the architectures proposed in the succeeding sections is summarized in Table 1.

3.2. Architecture for Discrepancy Computation

In this subsection, we present architectures for discrepancy computation. We compute the discrepancy polynomial using a two stage implementation. We first compute $Q_l(X + x_i, Y)$; the translation of $Q_l(X,Y)$ along the X variable. This is followed by the translation step along the Y variable. From (1), we see that translation along the X variable is an operation on the $\sigma^{(l,b)}(X)$ polynomials. Specifically, we need to compute the first $m_{i,j}$ coefficients in $\sigma^{(l,b)}(X + x_i)$. We denote these coefficients as a polynomial $\mu^{(l,b)}(X) = \sum_{\alpha=0}^{m_{i,j}} \mu_\alpha^{(l,b)} X^\alpha$. If $w_{l,b} > m_{i,j}$, the coefficients of $\mu^{(l,b)}(X)$ can be determined efficiently via a two step process, both of which are derived using the property: $(X + x_i)^\tau = X^\tau + x_i^\tau$ in $GF(2^q)$ for τ a power of 2. In the first step, we reduce the $w_{l,b}$ degree polynomial to an equivalent polynomial $\tilde{\sigma}^{(l,b)}(X) = \sum_{a=0}^{\tau-1} \tilde{\sigma}_a^{(l,b)} X^a$ of degree $\tau - 1$, where $\tau \ge m_{i,j} > \frac{\tau}{2}$ and τ is a integer power of 2. The coefficients of $\tilde{\sigma}^{(l,b)}(X)$ are obtained using:

$$\tilde{\sigma}_a^{(l,b)} = \sum_{s=0}^{\lceil w_{l,b}/\tau \rceil} \sigma_{\tau s+a}^{(l,b)} x_i^{\tau s} \qquad 0 \le a < \tau \qquad (5)$$

In the next step, we translate this equivalent polynomial:

$$\mu^{(l,b)}(X) = [\tilde{\sigma}^{(l,b)}(X + x_i)]_{m_{i,j}} \qquad (6)$$

Since (5) has no binomial factors, all the terms contribute to the summation and hence it can be computed efficiently using the Horner's formula. In (6), a degree $\tau - 1$ polynomial is translated to another polynomial of degree $\le \tau$. This step is implementated using an FFT like formulation shown in Fig. 3(c). This implementation uses the relation: $(X + x_i)^\tau = X^\tau + x_i^\tau$ when τ is a power of 2.

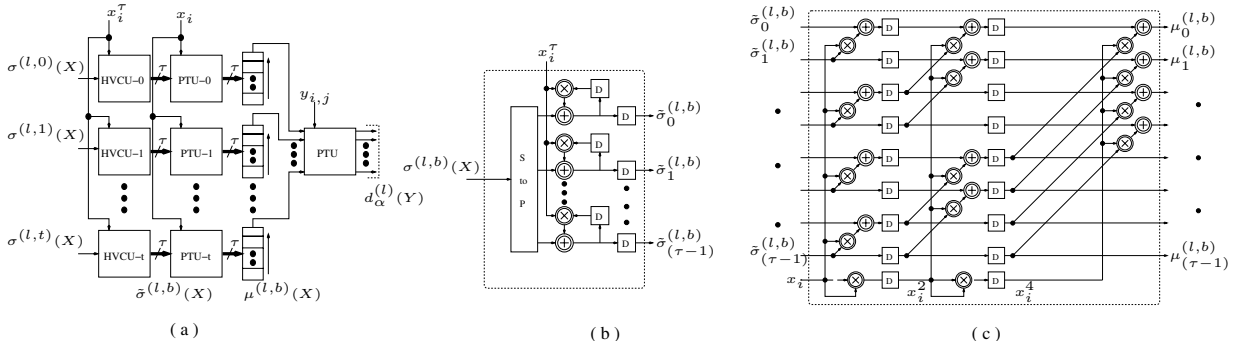

Fig. 3. Block diagram of the **DPC** unit (a) and its building blocks:(b) Horner's vector computation unit (**HVCU**) and (c) polynomial translation unit (**PTU**).

The architecture for (5) based on the Horner's formula is shown in Fig. 3(b) and the architecture for computing (6) is shown in Fig. 3(c). Finally, the discrepancy polynomial is computed using:

$$d_{\alpha}^{(l)}(Y) = [\tilde{\mu}^{(l,\alpha)}(Y + y_{i,j})]_{m_{i,j}-\alpha} \quad 0 \leq \alpha < m_{i,j} \quad (7)$$

where $\tilde{\mu}^{(l,\alpha)}(Y) = \sum_{b=0}^{t} \mu_{\alpha}^{(l,b)} Y^b$. The polynomial translation architecture shown in Fig. 3(c) is also used to implement this step. A block diagram of the discrepancy polynomial computation (**DPC**) block is shown in Fig. 3(a). As the degree of $\sigma^{(l,b)}(X)$ increases, the complexity of (5) dominates the discrepancy computation complexity and for τ coefficients, approaches the complexity of computing (3). Hence, the complexity of computing the discrepancy polynomial is equivalent to that of computing a single discrepancy coefficient which leads to a factor $\binom{m_{i,j}+1}{2}$ speedup and complexity reduction in the discrepancy computation step.

3.3. Architectures for re-encoded interpolation

Re-encoding of interpolation points [8], is an algorithm level technique in which the initialization of the candidate polynomials is modified in order to force $Q_l(X, Y)$ to pass through k specific interpolation points with required multiplicity. In this subsection, we modify the architectures presented in Section III-(B) in order to incorporate re-encoded interpolation. With the re-encoding transformation, the candidate polynomials at any stage can be represented as:

$$Q_l(X, Y) = \sum_{b=0}^{t} \sigma^{(l,b)}(X) g^{(b)}(X) Y^b \quad (8)$$

$$\text{where } g^{(b)}(X) = \prod_{i=0}^{k-1} (X + x_i)^{[m_{i,0}-b]} \quad (9)$$

where $[s] \equiv max(s, 0)$. Further reduction in interpolation complexity is achieved by using a co-ordinate transformation $Z = \frac{Y}{\prod_{i=0}^{k-1}(X+x_i)}$ [7], where for simplicity we have assumed that the first k intepolation points have distinct code-evaluation points and are chosen for co-ordinate transformation. It should be noted that this transformation is interpolation point dependent; the denominator of this transformation should not contain the code-evaluation point of the current interpolation point. For the interpolation points, whose code evaluation point was not among the k chosen indices, the candidate polynomials in (8) can be re-written as:

$$Q_l(X, Z) = g^{(0)}(X) \sum_{b=0}^{t} \sigma^{(l,b)}(X) h^{(b)}(X) Z^b \quad (10)$$

where $h^{(b)}(X) = \prod_{i=0}^{k-1} (X + x_i)^{[b-m_{i,0}]}$.

Now, the polynomial $h^{(b)}(X)$ termed an *auxiliary polynomial*, has a smaller degree compared to $g^{(b)}(X)$. The auxiliary polynomials are independent of the candidate polynomial and fixed for a given received vector. In (10), the polynomial $g^{(0)}(X)$ of degree $\sum_{i=0}^{k-1} m_{i,0}$, is common to all the interpolation polynomials and is excluded from the interpolation process. The interpolation process is carried on the set of polynomials $\tilde{Q}_l(X, Z)$ where:

$$\tilde{Q}_l(X, Z) = \sum_{b=0}^{t} \sigma^{(l,b)}(X) h^{(b)}(X) Z^b \quad (11)$$

As noted earlier, the co-ordinate transformation is interpolation point dependent. Hence the auxiliary polynomial needs to be modified for performing interpolation on those interpolation points whose code-evaluation point was among the k chosen indices [7]. In any case, the interpolation is performed on candidate polynomials having the form given in (11).

We now discuss the modifications in the interpolation architectures to incorporate re-encoded interpolation. Retaining the polynomials in X as two separate polynomials as in (11), reduces the complexity of both discrepancy polynomial computation and of polynomial update. The process of interpolation as described previously, is modified to account for the auxiliary polynomial. We now need to compute two translated polynomials, $[\sigma^{(l,b)}(X + x_i)]_{m_{i,j}}$ and $[h^{(b)}(X + x_i)]_{m_{i,j}}$, using the two step procedure as described in Section III-A.

And $\mu^{(l,b)}(X)$, is now given by:

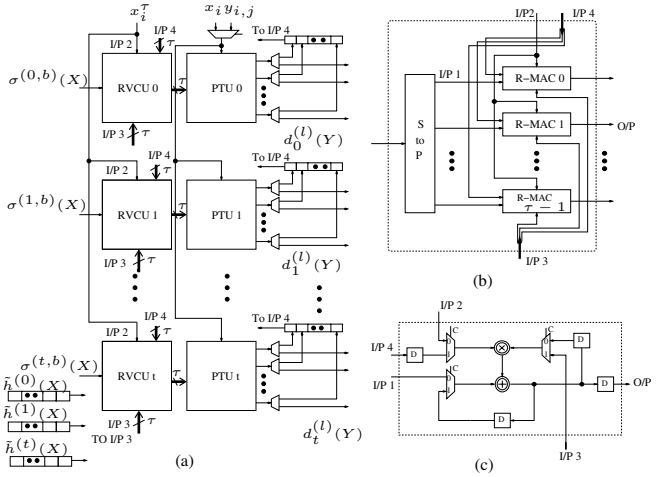

Fig. 5. Re-encoded **DPC** and its building blocks: (a) Re-encoded discrepancy polynomial computation (**R-DPC**) unit, (b) **R-MAC** vector computation unit (**RVCU**), and (c) reconfigurable MAC (**R-MAC**) unit.

$$\mu^{(l,b)}(X) = [[\sigma^{(l,b)}(X+x_i)]_{m_{i,j}} \times [h^{(b)}(X+x_i)]_{m_{i,j}}]_{m_{i,j}} \quad (12)$$

We observe that the operation being performed in (12) is polynomial multiplication. Hence, computing a single coefficient $\mu_\alpha^{(l,b)}$ in $\mu^{(l,b)}(X)$ requires the first α coefficients of the translated polynomials $\sigma^{(l,b)}(X+x_i)$ and $h^{(b)}(X+x_i)$. This is another motivation for joint computation of the discrepancy coefficients.

The sequence of steps leading to the discrepancy polynomial is: (5), (6), (12), and (7). We note that (6) and (7) involve the same operations and can be assigned to the **PTU** unit of Fig. 3(b). The **HVCU** architecture in Fig. 3(a) for performing the Horner's summation, is now modified so that it can also perform the convolution operation required in (12).

The reconfigurable MAC (**R-MAC**) unit, shown in Fig. 5(c) can perform both (5) and (12) with appropriate control to the multiplexers. The re-encoded discrepancy polynomial computation (**RDPC**) unit is shown in Fig. 5(a). The **R-MAC** vector computation unit (**RVCU**) shown in Fig. 5(b) comprises of an array of **R-MAC** units. The translated auxiliary polynomial $[h^{(b)}(X+x_i)]_{m_{i,j}}$ computation is implemented using a cascade of a **HVCU** unit and a **PTU** unit.

3.4. Polynomial Update Architectures

In this subsection, we present architectures for performing polynomial update. The computations involved in candidate polynomial update and discrepancy polynomial update in the **PSIA** are shown in Fig. 2. The steps **PU1** and **PU2** can be reformulated by dividing the update equations by $d_{\alpha\beta}^{(minl)}$ as shown below:

$$\begin{aligned} Q_l(X,Y) &= \lambda_{\alpha\beta}^{(l)} Q_{minl}(X,Y) + Q_l(X,Y) \\ D_{i,j}^{(l)}(X,Y) &= \lambda_{\alpha\beta}^{(l)} D_{i,j}^{(minl)}(X,Y) + D_l(X,Y) \end{aligned} \quad (13)$$

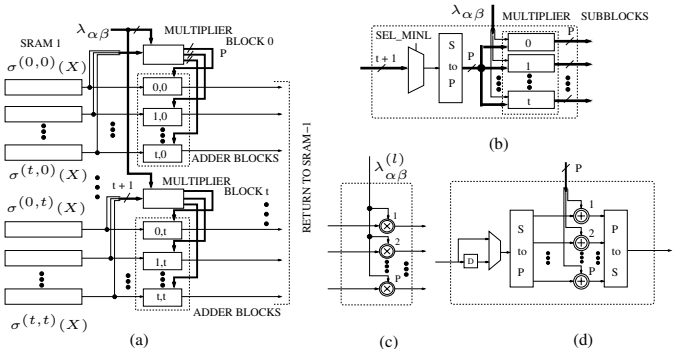

Fig. 6. Polynomial update architecture and building blocks: (a) P-parallel candidate polynomial update (**PU**) architecture, (b) multiplier block, (c) multiplier sub-block, and (d) adder block

$$\begin{aligned} Q_{minl}(X,Y) &= \lambda_{\alpha\beta}^{(minl)} Q_{minl}(X,Y) + XQ_{minl}(X,Y) \\ D_{i,j}^{(minl)}(X,Y) &= D_{i,j}^{(minl)}(X,Y) \times X \\ Wdeg_{minl} &= Wdeg_{minl} + 1 \end{aligned}$$

$$\text{where } \lambda_{\alpha\beta}^{(l)} = \begin{cases} x_i & \text{if } l = minl \\ \frac{1}{d_{\alpha\beta}^{(minl)}} \times d_{\alpha\beta}^{(l)} & \text{otherwise} \end{cases} \quad (14)$$

This reformulation reduces the total number of multiplications by $1/2$ at the cost of one inversion and t multiplications required to compute the $\lambda_{\alpha\beta}^{(l)}$ coefficients in (14). Step **PU0** is implemented using a table look-up of the sorted polynomial orders. Parallel processing can be used within an interpolation constraint to reduce the latency of polynomial update. An architecture for polynomial update with P degrees of parallel processing is shown in Fig. 6(a). The sub-blocks are shown in Fig. 6(b)-(d). In order to avoid the delay associated with moving polynomials through fixed length shift registers, the candidate polynomials are stored in dual port SRAM buffers. From (13), we see that the number of coefficients that need to be updated depend only on the degree of the minimal polynomial. Hence the addressing of the SRAM buffers can utilize read/write incremental pointers which roll over when the pointer values equal the length of the corresponding $\sigma^{(minl,b)}(X)$ in the minimal polynomial. From (13) and (14), we infer that the architectures for discrepancy polynomial update would be similar to that of candidate polynomial update, but the extent of parallel processing required would be smaller since the degree of the discrepancy polynomial is bounded by $m_{i,j}$.

4. ARCHITECTURES FOR FACTORIZATION

In this section, we describe architectures for factorization. Given the minimal polynomial $Q_{min}(X,Y)$ that satisfies all the interpolation constraints, the factorization procedure determines all the factors of $Q_{min}(X,Y)$ of the form $Y - f(X)$ with $deg[f(x)] < k$. The factorization architecture

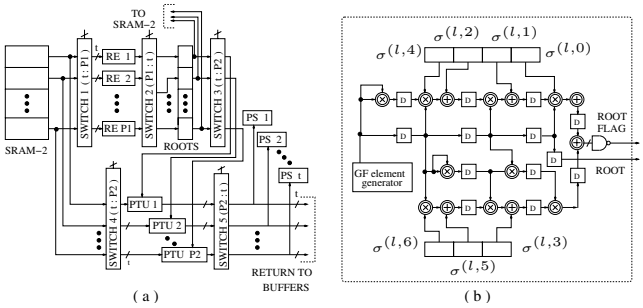

Fig. 7. Factorization architecture and building blocks: (a) **FA** and (b) root evaluator (**RE**) architecture.

is based on the algorithm presented in [5]. From an implementation perspective, it consists of iterative application of the following three steps:

- Root Evaluation (**RE**) step.
- Polynomial Translation (**PT**) step.
- Polynomial Shift (**PS**) step.

In the **RE** step, we determine the roots of:

$$Q_{min}(0, Y) = \sigma_0^{(l,0)} + \sigma_0^{(l,1)} Y + \cdots + \sigma_0^{(l,t)} Y^t \quad (15)$$

The **PT** and **PS** steps compute $Q_{min}(X, XY + f_0)$ from $Q_{min}(X, Y)$, where f_0 is a root found from the **RE** step. In the **PT** step we determine $\tilde{Q}_{min}(X, Y) = Q_{min}(X, Y + f_0)$ and in the **PS** step we determine $\tilde{Q}_{min}(X, XY)$. When the re-encoding and the co-ordinate transformation are used, it can be shown that the number of iterations of these three steps can be reduced from k, the datalength of the code, to 2θ, where θ is the maximum number of symbol errors that we plan to correct. We also reduce complexity by noting that the **PT** step need be applied to the minimal polynomial only on those coefficients that will be involved in the **RE** step. Usually, the highest X degree ζ of the minimal polynomial required in the **RE** step is only slightly greater than 2θ. An architecture for the overall factorization process is shown in Fig. 7(a). In the factorization procedure, checking different GF elements for roots in (15) can be done in parallel. Also, the **PT** step for different X degrees can be computed in parallel. This is shown in Fig. 7(a) where parallel processing by a factor P_1 and P_2 are utilized in the **RE** and **PT** steps respectively. **PS** involves no explicit computations on $\tilde{Q}_{min}(X, Y)$. It involves shifting the $\tilde{\sigma}^{(l,b)}(X)$ polynomials relative to each other. Rather than move the polynomial, we use registers to keep track of the effective displacement of each of the polynomials. These shift values are added to the address generator of the SRAM buffers, in order to produce the effective address. We note that the maximum number of roots that may be obtained from the factorization procedure is bounded by the highest Y degree in the minimal polynomial $Q_{min}(X, Y)$.

The architecture for the **RE** step is shown in Fig. 7(b). The architecture of the **PTU** block is the one shown in Fig. 3(a). The architecture in Fig. 7(b) is based on the technique presented in [6] for finding the roots of a degree 6 polynomial. The polynomial in (15) with $t = 6$, is represented as two polynomials $A(X)$ and $B(X)$ where :

$$\begin{aligned} A(X) &= \sigma_0^{(l,4)} X^4 + \sigma_0^{(l,2)} X^2 + \sigma_0^{(l,1)} X + \sigma_0^{(l,0)} \\ &= \sigma_0^{(l,0)} + X(\sigma_0^{(l,1)} + X[\sigma_0^{(l,2)} + X^2 \sigma_0^{(l,4)}]) \\ B(X) &= \sigma_0^{(l,6)} X^6 + \sigma_0^{(l,5)} X^5 + \sigma_0^{(l,3)} X^3 \\ &= X^3(\sigma_0^{(l,3)} + X^2[\sigma_0^{(l,5)} + X\sigma_0^{(l,6)}]) \end{aligned}$$

Since the characteristic of $GF(2^q)$ is 2, for any element $\nu \in GF(2^q)$, $A(\nu) = B(\nu)$ implies that ν is a root of (15). In the architecture of Fig. 7(b), we evaluate $A(\nu)$ and $B(\nu)$ and check for equality.

5. A RS $[255, 239]$ SOFT-DECODER ARCHITECTURE

In this section, we present a soft-decoder architecture for a $[255, 239]$ RS code using the interpolation and factorization architectures described in Sections III and IV respectively. Since the implementation of the Koetter-Vardy front end is application dependent, we assume a hypothetical front end which supplies the interpolation points and the corresponding multiplicities. Table 1 summarizes the latency and hardware overhead of each of the soft-decoding sub-blocks. The critical path in the proposed architecture consists of the path through a multiplier, adder, and a 2:1 multiplexer which occurs in the **R-MAC** unit shown in Fig. 5(c). Table 2 lists the system level parameters associated with a $[255, 239]$ soft-decoding algorithm. In Table 2, P_{CP} and P_{DP} refer to the polynomial update parallelization factor for candidate polynomials and discrepancy polynomials respectively. The latency and hardware requirements of the architecture are summarized in Table 3. The last row of Table 3 lists the hardware requirements of a systolic architecture for implementing the Berlekamp-Massey algorithm (BMA) [9], which solves the key equation in BMD decoding. The numbers within brackets denote the factor by which the soft-decoding terms are greater than the BMA architecture. This gives an indication of the area and delay cost associated with obtaining the increased coding gains of soft-decoding. From these numbers, we infer that in terms of complexity, further research both at the algorithm and architecture level is required to make soft-decoder implementations competitive with BMD decoders.

6. CONCLUSIONS

In this paper, we have presented an efficient formulation for interpolation in soft-decision decoding. We have proposed efficient architectures for implementing the interpolation and factorization steps. We have enumerated the hardware requirements and latency associated with soft-decoding a $[255, 239]$ RS code using the proposed architectures. We have compared the latency and hardware requirements of

Table 1. Latency and hardware requirements of proposed architectures

	Latency (clock cycles)	Hardware		
		MACs	2:1 Mux	Storage
HVCU	$\lceil \frac{w_{l,b}}{\tau} \rceil$	τ	-	4τ
PTU	$log_2\tau$	$\frac{\tau}{2}log_2\tau + (log_2\tau - 1)$	-	$\tau\,(log_2\tau - 1)$
DPC	**HVCU**$_{lat}$+ 2**PTU**$_{lat}$	$(t+1)$**HVCU**$_{MAC}$+ $(t+2)$**PTU**$_{MAC}$	-	$(t+1)$**HVCU**$_{Sto}$+ $(t+2)$**PTU**$_{Sto}$+ $(t+1)\tau$
R-MAC	1	1	3	4
RVCU	**HVCU**$_{lat}$ (5) $m_{i,j}$ (12)	τ	τ**R-MAC**$_{Mux}$	τ**R-MAC**$_{Sto}$ $+2\tau$
RDPC	**HVCU**$_{lat}$+$m_{i,j}$ $+2$**PTU**$_{lat}$	$(t+1)$**RVCU**$_{MAC}$+ $(t+1)$**PTU**$_{MAC}$	$(t+1)$**RVCU**$_{Mux}$ $+\tau(t+1)$	$(t+1)$**RVCU**$_{Sto}$+ $(t+1)$**PTU**$_{Sto}$
Mult. Blk.	2	$P(t+1)$	t	$2P$
Add Blk.	2	-	1	$4P$
PU	$\lceil \frac{max(w_{l,b})}{P} \rceil$ $+2$	$(t+1)$Mult$_{MAC}$	$(t+1)^2$Add$_{Mux}$+ $(t+1)$Mult$_{Mux}$	$(t+1)^2$ Add$_{Sto}$+ $(t+1)$Mult$_{Sto}$
RE	5	9	-	20
FA	$\frac{t\zeta 2^q}{P_2} + \frac{t\zeta^2}{2P_1}$	P$_1$**RE**$_{MAC}$ +P$_2$**PTU**$_{MAC}$	$5(t-1)$ $+2(P_1+P_2-2)$	P$_1$ **RE**$_{Sto}$+ P$_2$**PTU**$_{Sto}$

Table 2. Soft-decoder implementation details for $[255, 239]$ RS code

Quantity	Value	Quantity	Value
Total constraints	5355	τ	8
Constraints after reencoding	≈ 500	Max. mult	8
Interpolation points	≈ 100	P$_{CP}$	20
Average multiplicity	≈ 3	P$_{DP}$	4
Maximum $w_{l,b}$	80	P$_1$	40
Max. correctable symbol errors	13	P$_2$	5
Candidate polynomials $t+1$	7	ζ	30

Table 3. Latency and hardware requirements of $[255, 239]$ implementation

	Latency (cycles)	Hardware		
		MACs	2:1 Mux	Storage
RDPC	1400	1232	1568	3472
PU	1300	1174	91	3024
FA	1980	430	130	680
Total	2700	2836	1789	7176
BMA [9]	16	50	25	50

the proposed architectures with that of a systolic implementation of BMA for the same code parameters.

7. REFERENCES

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

[2] V. Guruswami and M. Sudan, "Improved decoding of Reed-Solomon and algebraic-geometric codes," *IEEE Trans. Inform. Theory* , vol. 45, pp. 1757-1767, Sept. 1999.

[3] R. Koetter and A. Vardy, "Algebraic soft-decision decoding of Reed-Solomon codes," Submitted to *IEEE Trans. Inform. Theory*, May 31, 2000.

[4] R. R. Nielsen and T. Høholdt, "Decoding Reed-Solomon codes beyond half the minimum distance," preprint, 1998.

[5] R. M. Roth and G. Ruckenstein, "Efficient decoding of Reed-Solomon codes beyond half the minimum distance," *IEEE Trans. Inform. Theory*, vol. 46, pp. 246-257, Jan. 2000.

[6] T. K. Troung, J. H. Jeng and I. S. Reed, "Fast algorithm for computing the roots of error locator polynomials upto degree 11 in Reed-Solomon decoders," *IEEE Trans. on Comm.*, vol. 49, pp. 779 -783, May 2001.

[7] R. Koetter, J. Ma, A. Vardy and A. Ahmed, "Efficient interpolation and factorization in algebraic soft-decision decoding of Reed-Solomon codes," *Proc. of IEEE Symp. On Info. Theory*, 2003.

[8] W. J. Gross, F. R. Kschsichang, R. Koetter and P. G. Gulak, "A VLSI architecture for interpolation in soft-decision list decoding of Reed-Solomon codes," *Proc. of IEEE Workshop on Signal Processing Systems*, Oct. 2002.

[9] D. V. Sarwate and N. R. Shanbhag, "High-speed architectures for Reed-Solomon decoders," *IEEE Trans. VLSI Systems*, vol. 9, pp. 641-655, Oct. 2001.