Combined Interpolation Architecture for Soft-decision Decoding of Reed-Solomon Codes

Jiangli Zhu, Xinmiao Zhang
Case Western Reserve University
{jiangli.zhu, xinmiao.zhang}@case.edu

Zhongfeng Wang
Broadcom Corporation
zfwang@broadcom.com

Abstract— **Reed-Solomon (RS) codes are one of the most extensively used error control codes in digital communication and storage systems. Recently, significant advancements have been made on algebraic soft-decision decoding (ASD) of RS codes. These algorithms can achieve substantial coding gain with polynomial complexity. One major step of ASD is the interpolation. Various techniques have been proposed to reduce the complexity of this step. Further speedup of this step is limited by the inherent serial nature of the interpolation algorithm. In this paper, taking the bit-level generalized minimum distance (BGMD) ASD as an example, we propose a novel technique to combine the computations from multiple interpolation iterations. Compared to the single interpolation iteration architecture for a (255, 239) RS code, the combined architecture can achieve 2.7 times throughput with only 2% area overhead in high signal-to-noise ratio scenarios.**

I. Introduction

Reed-Solomon (RS) codes are among the most widely used error-correcting codes in digital communication and data storage systems. Recently, significant advancements have been made on algebraic soft-decision decoding (ASD) [1]–[3] of RS codes. By incorporating the probability information from the channel into the algebraic interpolation process developed by Guruswami and Sudan [4], significant coding gain can be achieved by ASD, while their complexity is polynomial with respect to the codeword length. ASD algorithms consist of two major steps: the interpolation and the factorization. In this paper, we focus on the interpolation step, which is the speed bottleneck of ASD algorithms.

The most popular interpolation algorithm is the Nielson's algorithm [5], [6]. This is an iterative algorithm and each iteration mainly consists of discrepancy coefficient computation and candidate polynomial updating. Various techniques have been proposed to reduce the complexity of the interpolation. The total number of iterations involved in the interpolation can be reduced by employing the re-encoding and coordinate transformation techniques proposed in [7], [8]. Optimizing the computations involved in each iteration is another approach to reduce the interpolation complexity. The latency of the discrepancy coefficient computation can be reduced by the point-serial scheme proposed in [9], [10]. Based on this scheme, a systolic array architecture has been proposed for the interpolation step [9]. In addition, deep pipelining of the interpolation architecture is enabled by using a hybrid representation for finite field elements [11]. In [12], it is found that a significant proportion of the discrepancy coefficients are zero and the updating of corresponding candidate polynomials can be skipped. Accordingly, the number of hardware units implementing candidate polynomial updating can be reduced substantially. Although these efforts led to faster and smaller area implementation of the interpolation, further speedup is limited by the inherent serial nature of the Nielson's algorithm.

In this paper, we present a novel technique to combine the computations from multiple iterations of the interpolation. To illustrate our idea, we use the bit-level generalized minimum distance (BGMD) ASD with maximum multiplicity two as an example. Multiple interpolation iterations need to be carried out for an interpolation point with multiplicity larger than one. For the iterations associated with the same interpolation point, the discrepancy coefficients can be computed in a 'look-ahead' manner. Based on the computed discrepancy coefficients, the candidate polynomial updating in those iterations can be combined efficiently. In the BGMD algorithm, there may also be pairs of interpolation points with multiplicity one that share the same X coordinate. Only one interpolation iteration is required for each of these points. However, the discrepancy coefficients for each pair can be computed simultaneously with small overhead due to the same X coordinate. Accordingly the corresponding polynomial updating can be also combined. The savings can be brought by the proposed scheme is dependent on the multiplicities of the interpolation points, which are decided by channel conditions. In high signal-to-noise ratio (SNR) scenarios, our proposed combined interpolation architecture for the BGMD decoding of a (255, 239) RS code can achieve 2.7 times throughput compared to single interpolation iteration architectures, while the area requirement has only been increased by 2%. In terms of speed/area ratio, our architecture is 165% more efficient. At low SNR, our architecture is 70% more efficient.

The structure of this paper is as follows. Section 2 introduces BGMD ASD algorithm and the Nielson's interpolation algorithm. The proposed combined interpolation scheme is presented in Section 3. Then in Section 4, efficient architectures are proposed for the combined interpolation. Section 5 gives the hardware requirement and latency analyses. Conclusions are provided in Section 6.

II. The BGMD Algorithm and the Nielson's interpolation

In this paper, we consider RS codes constructed over finite field $GF(2^q)$. For a primitive RS codes, $n = 2^q - 1$. For k

message symbols, $(f_0, f_1, \cdots, f_{k-1})$, where $f_i \in GF(2^q)$, the message corresponding polynomial is $f(X) = f_0 + f_1X + \cdots + f_{k-1}X^{k-1}$. The encoding of RS codes can be carried out by evaluating the message polynomial at n distinct nonzero elements of $GF(2^q)$. Assume the fixed-ordered set $\{x_0, x_1, \cdots, x_{n-1}\}$ is chosen as the n distinct evaluation elements, the codeword corresponding to $(f_0, f_1, \cdots, f_{k-1})$ is $(f(x_0),\ f(x_1),\ \cdots,\ f(x_{n-1}))$.

ASD algorithm decoding consists of three steps: the multiplicity assignment, the interpolation, and the factorization. As the first step, the multiplicity assignment receives the reliability information from the channel and then decides on the multiplicities of the interpolation points. To increase the probability that the correct message polynomial can be recovered, usually assign larger multiplicities to those more reliable points. Popular multiplicity assignment schemes include the Koetter-Vardy (KV) [1], low-complexity Chase (LCC) [2] and the BGMD scheme [3]. In this paper, we will use the BGMD scheme with maximum multiplicity two as an example to illustrate the proposed technique. In the BGMD algorithm, the multiplicities are assigned by using bit-level reliabilities. Each received symbol, θ_j, consists of q noise-corrupted bits. Define the log-likelihood ratio (LLR) of a noise-corrupted bit, δ, by $LLR_\delta = \log(Pr(0|\delta)/Pr(1|\delta))$. δ is considered to be erased if $|LLR_\delta|$ is smaller than a threshold. Assume the maximum multiplicity of the interpolation points is m_{max}, the BGMD algorithm assigns multiplicities based on the number of bits erased in each received symbol according to the following scheme.

Algorithm A: BGMD Multiplicity Assignment

1) if no bit is erased in θ_j, assign m_{max} to (x_j, y_j), where y_j is the hard-decision of θ_j;

2) if there is only one bit erased in θ_j, assign $m_{max}/2$ to both (x_j, y_{j1}) and (x_j, y_{j2}), where y_{j1} and y_{j2} are the hard-decision of θ_j and the field element that differs from the hard-decision in only the erased bit, respectively;

3) if there are more than one bit erased in θ_j, do not assign any multiplicity to the interpolation points with x_j.

Although ASD algorithms differ in the multiplicity assignment step, they share the same interpolation and factorization steps. Some definitions necessary for understanding the interpolation-based decoding algorithms are given below.

Definition 1 A bivariate polynomial $Q(X,Y)$ passes a point (x_0, y_0) with multiplicity m if the shifted polynomial $Q(X + x_0, Y + y_0)$ contains a monomial $X^\alpha Y^\beta$ with degree $\alpha + \beta = m$, and does not contain any monomial with degree less than m.

Definition 2 For non-negative integers w_x and w_y, the (w_x, w_y)-weighted degree of a bivariate polynomial $Q(X,Y) = \sum_{i=0}^{\infty}\sum_{j=0}^{\infty} q_{i,j}X^iY^j$ is the maximum of $iw_x + jw_y$ such that $q_{i,j} \neq 0$.

The purpose of the interpolation step is to find a bivariate polynomial, $Q(X,Y)$, with minimum $(1, k-1)$ weighted degree that passes each non-trivial interpolation point with at least its associated multiplicity. Then the factorization finds all factors of $Q(X,Y)$ in the form of $Y - f(X)$ with the degree of $f(X)$ at most k. These derived polynomials contain all the message polynomials in the list as a subset. In this paper, we focus on the interpolation step. Popular interpolation algorithms include the Nielson's algorithm and the Lee-O'Sullivan (LO) algorithm [13]. However, the LO algorithm can not take in more points once the interpolation started. In this paper, we use the Nielson's algorithm in our design. The pseudo codes of this algorithm are listed in Algorithm B.

Algorithm B: Nielson's Interpolation

```
  initialization:
  $Q^{(0)}(X,Y) = 1, Q^{(1)}(X,Y) = Y, \cdots, Q^{(t)}(X,Y) = Y^t$
  $Wdeg_0 = 0, Wdeg_1 = k-1, \cdots, Wdeg_t = t(k-1)$
  interpolation starts:
  for each interpolation point $(x_i, y_{i,j})$ with multiplicity $m_{i,j}$
      for $\alpha = 0$ to $m_{i,j} - 1$ and $\beta = 0$ to $m_{i,j} - \alpha - 1$
I1:       compute $d^{(l)}_{\alpha,\beta}(x_i, y_{i,j}) (0 \le l \le t)$
I2:       $minl = \arg\min_l (Wdeg_l | d^{(l)}_{\alpha,\beta}(x_i, y_{i,j}) \neq 0, 0 \le l \le t)$
          for $l = 0$ to $t$, $l \neq minl$
I3:           $Q^{(l)}(X,Y) \Leftarrow d^{(minl)}_{\alpha,\beta}(x_i, y_{i,j}) Q^{(l)}(X,Y)$
                              $+ d^{(l)}_{\alpha,\beta}(x_i, y_{i,j}) Q^{(minl)}(X,Y)$
I4:       $Q^{(minl)}(X,Y) \Leftarrow Q^{(minl)}(X,Y)(X + x_i)$
          $Wdeg_{minl} \Leftarrow Wdeg_{minl} + 1$
  Output: $Q^{(\varphi)}(X,Y) (\varphi = \arg\min_l (Wdeg_l | 0 \le l \le t))$
```

In Algorithm B, the discrepancy coefficient, $d^{(l)}_{\alpha,\beta}(x_i, y_{i,j})$, is defined as the coefficient of the monomial $X^\alpha Y^\beta$ in $Q^{(l)}(X + x_i, Y + y_{i,j})$. It can be computed as

$$d^{(l)}_{\alpha,\beta}(x_i, y_{i,j}) = \sum_{a \ge \alpha}\sum_{b \ge \beta} \binom{a}{\alpha}\binom{b}{\beta} x_i^{a-\alpha} y_{i,j}^{b-\beta} q^{(l)}_{a,b}, \qquad (1)$$

where $q^{(l)}_{a,b}$ is the coefficient of X^aY^b in $Q^{(l)}(X,Y)$. The basic idea of Algorithm B is to first initialize a set of $t+1$ bivariate candidate polynomials $Q^{(l)}(X,Y)$ $(0 \le l \le t)$ as $1, Y, Y^2, \cdots, Y^t$. Then these polynomials are updated to satisfy one extra interpolation constraint at a time at the cost of minimum increase in $(1, k-1)$-weighted degree. After the constraints for all interpolation points have been satisfied, the candidate polynomial with the least weighted degree is selected as the output. It can be derived that an interpolation point of multiplicity m adds $m(m+1)/2$ constraints. Since one extra constraint is satisfied in each iteration, $m(m+1)/2$ interpolation iterations need to be carried out for a point of multiplicity m. The re-encoding and coordinate transformation [7], [8] can convert the Y coordinate of the k most reliable interpolation points to zero. As a result, the interpolation over these points can be pre-solved by simple univariate interpolation. In addition, the point-serial scheme [9], [10] and a hybrid finite field element representation [11] can be employed to increase the speed of the discrepancy coefficient computation. Further speedup of the interpolation is limited by the serial nature of Algorithm B.

III. The Combined Interpolation

Due to the data dependency, the computations from multiple interpolation iterations can not be carried out simultaneously. In this section, taking the BGMD algorithm as an example, we propose a novel and efficient scheme that combines the candidate polynomial updating of multiple iterations.

In the BGMD decoding with maximum multiplicity two, each coordinate position can have one interpolation point with multiplicity two, two interpolation points with multiplicity one or no interpolation point. In this case, it can be derived that there are three candidate polynomials involved in the interpolation for high rate codes ($t = 2$ in Algorithm B). Now consider the interpolation over a point, (x_0, y_0), of multiplicity two, which requires three iterations for the constraints $(\alpha, \beta) = (0,0), (0,1)$ and $(1,0)$ in Algorithm B. Since the interpolation point is the same in these three iterations, (x_0, y_0) is dropped from the $d^{(l)}_{\alpha,\beta}(x_0, y_0)$ notation in the remaining of this paper. Assume in the first iteration, $Q^{(0)}(X,Y)$ is picked as the minimum polynomial (*i.e. minl* = 0). Then the polynomials are updated according to the I3 and I4 steps of Algorithm B as

$$\begin{cases} Q^{(0)}(X,Y) \Leftarrow Q^{(0)}(X,Y)(X+x_0) \\ Q^{(1)}(X,Y) \Leftarrow d^{(0)}_{0,0}Q^{(1)}(X,Y) + d^{(1)}_{0,0}Q^{(0)}(X,Y) \\ Q^{(2)}(X,Y) \Leftarrow d^{(0)}_{0,0}Q^{(2)}(X,Y) + d^{(2)}_{0,0}Q^{(0)}(X,Y) \end{cases} \quad (2)$$

In the remaining of this paper, we use $d^{(l)}_{i,j(\alpha,\beta)}$ to denote the coefficient of X^iY^j in $Q^{(l)}(X+x_0, Y+y_0)$ before it is updated in the iteration of constraint (α,β). Apparently, the discrepancy coefficient, $d^{(l)}_{\alpha,\beta}$, in Algorithm B is equivalent to $d^{(l)}_{\alpha,\beta(\alpha,\beta)}$. From (2), it can be derived that $d^{(0)}_{0,1} = d^{(0)}_{-1,1(0,0)} = 0$. Hence $Q^{(0)}(X,Y)$ will not be picked as the minimum polynomial and does not need to be updated in the iteration of $(\alpha,\beta) = (0,1)$. On the other hand, $d^{(1)}_{0,1} = d^{(0)}_{0,0}d^{(1)}_{0,1(0,0)} + d^{(1)}_{0,0}d^{(0)}_{0,1(0,0)}$ and $d^{(2)}_{0,1} = d^{(0)}_{0,0}d^{(2)}_{0,1(0,0)} + d^{(2)}_{0,0}d^{(0)}_{0,1(0,0)}$. Therefore, either $Q^{(1)}(X,Y)$ or $Q^{(2)}(X,Y)$ can be the minimum polynomial in the iteration of $(\alpha,\beta) = (0,1)$, depending on their weighted degrees and whether $d^{(1)}_{0,1}$ or $d^{(2)}_{0,1}$ is zero. Assume $Q^{(1)}(X,Y)$ is picked as the minimum polynomial, the candidate polynomials are updated again according to the I3 and I4 steps. Similarly, $Q^{(1)}(X,Y)$ has zero discrepancy coefficient in the last iteration of $(\alpha,\beta) = (1,0)$, and will not be the minimum polynomial and does not need to be updated. In the last iteration, from (2), $d^{(0)}_{1,0}$ is nonzero. Therefore, the minimum polynomial can be either $Q^{(0)}(X,Y)$ or $Q^{(2)}(X,Y)$. Depending on whether $Q^{(0)}(X,Y)$ has been picked as the minimum polynomial twice or once in the three iterations, the updated polynomials from the last iteration can be written in terms of the polynomials at the beginning of the first iteration in one of the two formats below:

$$\begin{cases} Q^{(0)}(X,Y) \Leftarrow Q^{(0)}(X,Y)(X+x_0)^2 \\ Q^{(1)}(X,Y) \Leftarrow (d^{(0)}_{0,0}Q^{(1)}(X,Y) + d^{(1)}_{0,0}Q^{(0)}(X,Y))(X+x_0) \\ Q^{(2)}(X,Y) \Leftarrow d^{(0)}_{0,0}\big(\Delta_0 Q^{(0)}(X,Y) + \Delta_1 Q^{(1)}(X,Y) \\ \qquad + \Delta_2 Q^{(2)}(X,Y)\big) + \Delta_3 Q^{(0)}(X,Y)(X+x_0) \end{cases} \quad (3)$$

$$\begin{cases} Q^{(0)}(X,Y) \Leftarrow d^{(0)}_{0,0}\big(\Delta_0 Q^{(0)}(X,Y) + \Delta_1 Q^{(1)}(X,Y) \\ \qquad + \Delta_2 Q^{(2)}(X,Y)\big) + \Delta_3 Q^{(0)}(X,Y)(X+x_0) \\ Q^{(1)}(X,Y) \Leftarrow (d^{(0)}_{0,0}Q^{(1)}(X,Y) + d^{(1)}_{0,0}Q^{(0)}(X,Y))(X+x_0) \\ Q^{(2)}(X,Y) \Leftarrow \big(\Delta_0 Q^{(0)}(X,Y) + \Delta_1 Q^{(1)}(X,Y) \\ \qquad + \Delta_2 Q^{(2)}(X,Y)\big)(X+x_0) \end{cases} \quad (4)$$

According to (3) and (4), the candidate polynomial updating for the three iterations can be combined. In these equations, each Δ_j $(0 \leq j \leq 3\ )$ is a sum of products of $d^{(i)}_{0,0(0,0)}$, $d^{(i)}_{0,1(0,0)}$ and $d^{(i)}_{1,0(0,0)}$ for $i = 0,1,2$. Following the interpolation for the constrains $(\alpha,\beta) = (0,0)$, $(0,1)$ and $(1,0)$, Δ_j can be computed as

$$\begin{cases} \Delta_0 = d^{(1)}_{0,0(0,0)}d^{(2)}_{0,1(0,0)} + d^{(1)}_{0,1(0,0)}d^{(2)}_{0,0(0,0)} \\ \Delta_1 = d^{(2)}_{0,0(0,0)}d^{(0)}_{0,1(0,0)} + d^{(2)}_{0,1(0,0)}d^{(0)}_{0,0(0,0)} \\ \Delta_2 = d^{(0)}_{0,0(0,0)}d^{(1)}_{0,1(0,0)} + d^{(0)}_{0,1(0,0)}d^{(1)}_{0,0(0,0)} \\ \Delta_3 = \Delta_0 d^{(0)}_{1,0(0,0)} + \Delta_1 d^{(1)}_{1,0(0,0)} + \Delta_2 d^{(2)}_{1,0(0,0)} \end{cases} \quad (5)$$

From (1), it can be observed that $d^{(i)}_{0,1(0,0)}$ and $d^{(i)}_{1,0(0,0)}$ can be computed as byproducts of the $d^{(i)}_{0,0(0,0)}$ computation. Hence, compared to the discrepancy coefficient computation in the iteration of $(\alpha,\beta) = (0,0)$, the computation of Δ_j only requires small overhead to derive the sum of products.

As it can be observed from (3) and (4), $Q^{(1)}(X,Y)$, which is picked as the minimum polynomial in the iteration of $(\alpha,\beta) = (0,1)$, is the same for both cases. In addition, the $Q^{(2)}(X,Y)$ for the first case is the same as the $Q^{(0)}(X,Y)$ for the second case. When different candidate polynomials are picked as the minimum polynomials in the three iterations, the output polynomials at the end of the third iteration are in the same format as either (3) or (4), except that the candidate polynomials are switched. From our extensive simulations, we did not find any case that has all three discrepancy coefficients as zero in a iteration.

In the BGMD algorithm, there may also be pairs of interpolation points with multiplicity one that share the same X coordinate. For each of these points, only one interpolation iteration needs to be carried out. Although the discrepancy coefficient for one point can not be computed as the byproduct of the discrepancy coefficient computation of another, the discrepancy coefficients for the two points with the same X coordinate can be computed simultaneously with small overhead due to the shared coordinate and low Y-degree. Accordingly, the corresponding candidate polynomial updating can be also combined. In addition, the polynomial picked as the minimum polynomial during the interpolation over the point (x_j, y_{j1}) is multiplied by $(X + x_j)$. Hence this polynomial has zero discrepancy coefficient during the interpolation over the point (x_1, y_{j2}), and will not be picked as the minimum polynomial again. Assume $Q^{(0)}(X,Y)$ and $Q^{(1)}(X,Y)$, respectively, are the minimum polynomials in the interpolation iterations for two points of multiplicity one that share the same X coordinate. The candidate polynomial updating in these two iterations can be combined as

$$\begin{cases} Q^{(0)}(X,Y) \Leftarrow Q^{(0)}(X,Y)(X+x_0) \\ Q^{(1)}(X,Y) \Leftarrow Q^{(1)}(X,Y)(X+x_0) \\ Q^{(2)}(X,Y) \Leftarrow \Delta_0 Q^{(0)}(X,Y) + \Delta_1 Q^{(1)}(X,Y) \\ \qquad\qquad +\Delta_2 Q^{(2)}(X,Y) \end{cases} \tag{6}$$

where Δ_i $(i = 0,1,2)$ have the same format as those in (5), except that $d^{(i)}_{0,1,(0,0)}$ should be replaced by $d^{(i)}_{0,0,(0,0)}$ for the second point with the same X coordinate. In the case that different polynomials are picked as the minimum polynomials, the combined polynomials are in the same format as those in (6).

IV. VLSI Architecture for the Combined Interpolation

In this section, efficient architectures for the combined interpolation are presented. Since all the coefficients in the combined candidate polynomial updating in (3), (4) and (6) are computed based on the polynomial coefficients at the beginning of the iteration of $(\alpha,\beta) = (0,0)$, the subscript $(0,0)$ is dropped from the notation $d^{(l)}_{\alpha,\beta(0,0)}$ in this section, *i.e.*, $d^{(l)}_{\alpha,\beta(0,0)}$ is now denoted by $d^{(l)}_{\alpha,\beta}$.

A. Discrepancy Coefficient Computation Architecture

To enable the combining of the candidate polynomial updating, $d^{(l)}_{\alpha,\beta}$ for $(\alpha,\beta) = (0,0),(0,1)$ and $(1,0)$ need to be computed first based on the starting candidate polynomials in the first iteration. Then Δ_i are computed by using (5). The discrepancy coefficient computation equation can be rewritten as

$$\begin{aligned} d^{(l)}_{\alpha,\beta} &= \sum_{b\geq\beta}\left(\sum_{a\geq\alpha}\binom{a}{\alpha} q^{(l)}_{a,b} x_0^{a-\alpha}\right)\binom{b}{\beta} y_0^{b-\beta} \\ &= \sum_{b\geq\beta} c^{(l,b)}_{\alpha}\binom{b}{\beta} y_0^{b-\beta} \end{aligned} \tag{7}$$

where $c^{(l,b)}_{\alpha} = \sum_{a\geq\alpha}\binom{a}{\alpha} q^{(l)}_{a,b} x_0^{a-\alpha}$. Substituting α by 0 and 1, respectively, it can be derived that

$$c^{(l,b)}_0 = \sum_{a\geq 0} q^{(l)}_{a,b} x_0^a,\ c^{(l,b)}_1 = \sum_{a\geq 1}\binom{a}{1} q^{(l)}_{a,b} x_0^{a-1} \tag{8}$$

Fig. 1 (a) illustrates the architecture for computing $c^{(l,b)}_0$ and $c^{(l,b)}_1$. The register on the top left corner is initialized as '1' and the multiplier-register loop generates x_0^a in clock cycle a. The coefficients, $q^{(l)}_{a,b}$, are fed to the multiplier serially with the least significant one first. After $q^{(l)}_{a,b}$ is multiplied by x_0^a, the products are summed up to derive $c^{(l,b)}_0$. The computation of $c^{(l,b)}_1$ involves $\binom{a}{1}$. Since $\binom{a}{1}$ is computed over finite fields of characteristic two, it is either 0 or 1 when a is an even or odd number, respectively. Hence $\binom{a}{1}$

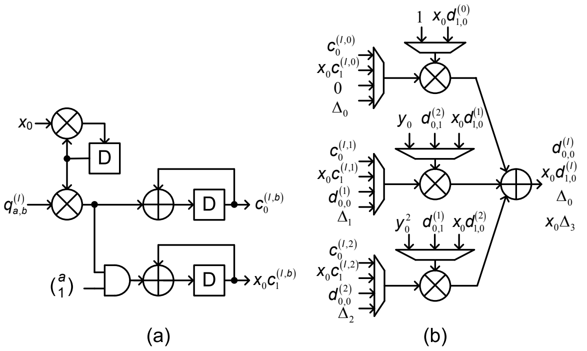

Fig. 1. The architecture for discrepancy coefficient computation (DCC)

equals the least significant bit of a. According to (8), $c^{(l,b)}_1$ can be computed by using another multiplier and adder in a similar way as that for $c^{(l,b)}_0$ computation. Alternatively, scaling a candidate polynomial by a nonzero factor does not change the decoding output. Since $x_0 \neq 0$, we can compute $c^{(l,b)}_1 x_0 = \sum_{a\geq 1}\binom{a}{1} q^{(l)}_{a,b} x_0^a$ instead, as long as other involved discrepancy coefficients are also scaled properly. The computation of $c^{(l,b)}_1 x_0$ does not need multiplier and can be done by adding just one adder loop as shown by the bottom branch in Fig. 1(a). Nine copies of the architecture in Fig. 1(a) are required. Hence computing $c^{(l,b)}_1 x_0$ instead brings a saving of nine multipliers. The number of pipelining stages of this architecture is $\xi_{dcca} = 1$.

Once $c^{(l,b)}_{\alpha}$ are computed, $d^{(l)}_{\alpha,\beta}$ can be derived according to (7). Since the maximum Y-degree of the polynomials is two, $d^{(l)}_{0,1} = c^{(l,1)}_0$ and $d^{(l)}_{\alpha,0} = \sum_{0\leq b\leq 2} c^{(l,b)}_{\alpha} y_0^b$ for $\alpha = 0,1$. Accordingly, $d^{(l)}_{0,0}$ and $x_0 d^{(l)}_{1,0}$ can be computed using the architecture in Fig. 1(b) by routing proper inputs through the multiplexors. In total, three copies of this architecture are required, one for each candidate polynomial. After the discrepancy coefficients are computed, each of Δ_i for $i = 0,1,2$ is computed by one copy of the architecture. Then $x_0\Delta_3$ is computed by using a single architecture. Since all these computations are carried out in the architecture in Fig. 1(b) in a time-multiplexed way, it takes $\xi_{dccb} = 4$ clock cycles to derive the required coefficients in (3) and (4). The same architectures can be used to compute the coefficients in (6).

B. Polynomial Updating Architecture

After Δ_j have been computed, the polynomial updating can be carried out by using the PU architecture shown in Fig. 2. Since there is no data dependency among the coefficients with different Y degree, they are updated in parallel in our design. Fig. 2 shows the architecture for updating the coefficients with one of the three Y-degrees. In this architecture, the multiplications by $(X + x_0)$ are implemented by feeding the polynomial coefficients serially with the least significant one first to the A blocks. It can be observed that the four different polynomials in (3) and (4) have the following common terms: $Q^{(0)}(X,Y)(X+x_0)$, $Q^{(1)}(X,Y)(X+x_0)$, and $\Delta_0 Q^{(0)}(X,Y) + \Delta_1 Q^{(1)}(X,Y) + \Delta_2 Q^{(2)}(X,Y)$. These terms are shared in the PU architecture to reduce the complexity of the combined polynomial updating. (3) and (4) only has one different polynomial. The different polynomial is chosen by the 2-to-1 multiplexor in gray color. By passing through the

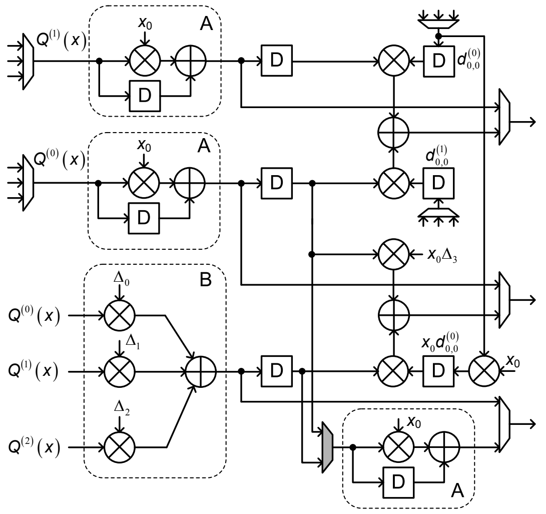

Fig. 2. The architecture for polynomial updating (PU)

lower input to each of the 2-to-1 multiplexor on the right of Fig. 2, the three polynomials in (3) or (4) are available at the output. In addition, the PU architecture also takes care of the combined polynomial updating for two interpolation points of multiplicity one according to (6). In this case, the upper input of each multiplexor needs to be passed to the output. As it was mentioned in Section 3, it is possible that other candidate polynomials instead of $Q^{(0)}(X,Y)$ and $Q^{(1)}(X,Y)$ are picked as the minimum polynomials in the first and second iterations. In this case, the candidate polynomials and their discrepancy coefficients need to be switched. These switching are handled by the 3-to-1 multiplexors in Fig. 2 . It can be noted that an extra multiplier is added to multiply x_0 with $d^{(0)}_{0,0}$. This is because that Δ_3 is scaled by x_0 when it is computed, and hence $d^{(0)}_{0,0}$ also needs to be scaled by the same factor to compute $Q^{(2)}(X,Y)$ in (3) or $Q^{(0)}(X,Y)$ in (4). Although calculating $x_0 d^{(0)}_{0,0}$ here requires one more multiplier, nine multipliers have been saved in the DCC architecture by computing $x_0\Delta_3$ instead of Δ_3.

In previous designs [10], [11], the coefficients of updated polynomials are fed back to their own memory blocks and multiplexors are required for coefficient routing. In our architecture, instead, the three updated polynomials at the output are written back to fixed memory blocks. Although a few extra gates are required to keep track of the weighted degree of the polynomials, the multiplexors for routing are eliminated and hence the critical path as well as total gate number are reduced. To increase the speed, registers are added to break the data path in Fig. 2 into $\xi_{pu} = 2$ pipelining stages.

C. Computation Scheduling

All blocks in the DCC architecture in Fig. 1 and the PU architecture in Fig. 2 take polynomial coefficients with the least significant one first. Once the coefficients have been updated, they can be passed to the DCC architecture to compute the discrepancy coefficients for the next interpolation iteration. A control unit is employed to generate the control signals for the DCC and PU architectures. It is also used to record and update the weighted degrees of the polynomials. During the interpolation, the degrees of polynomials are either increased by one or two according to (3) and (4). The degree change can be decided right after the discrepancy coefficients are computed. Hence the degree updating can be carried out in parallel with polynomial updating.

It can be observed that the block B in Fig. 2 has the same architecture as that in Fig. 1 (b), if the multiplexors are ignored. All coefficients of $q^{(l)}_{a,b}$ for $a \geq \alpha$ are required to compute $c^{(l,b)}_{\alpha}$. Accordingly, during the updating of candidate polynomials, $c^{(l,b)}_{\alpha}$ for the next iteration is not available and the computation of $d^{(l)}_{\alpha,\beta}$ and Δ_i can not start. On the other hand, the polynomial updating can not start before $d^{(l)}_{\alpha,\beta}$ and Δ_i are computed. Hence, the architecture in Fig. 1 (b) and the block B in Fig. 2 will not be activated simultaneously. Therefore, by adding one extra input to each of the multiplexors on top of the multipliers in Fig. 1, a single architecture can be shared for both purposes in a time-multiplexed way.

V. Hardware Requirement and Latency Analyses

Table I lists the gate count and critical path of each building block, except the control unit, in the combined interpolation architecture for a (255, 239) RS code. All the gates in this table are 2-input gates, and the Muxes refer to 1-bit 2:1 multiplexors. A $GF(2^8)$ multiplier can be implemented by 64 XOR gates and 48 AND gates with 6 XOR gates and 1 AND gate in the critical path. The gates required for the architecture in Fig. 1 (b) is not counted in the DCC architecture. They are included in the PU architecture instead. As it can be observed from Table I, the critical path of our combined interpolation architecture consists of 12 gates. From simulations, the maximum X-degree of the candidate polynomial coefficients is 33. Hence a memory of $33 \times 3 \times 3 = 297$ bytes is required to store the polynomial coefficients involved in the interpolation. In our design, the weighted degrees of the polynomials are stored in registers. Taking these registers into account, the total number of registers required in our design is 345.

In the original Nielson's interpolation, one iteration is carried out for each interpolation constraint. Hence only one discrepancy coefficient needs to be computed for each polynomial. This computation can still be carried out by the architecture shown in Fig. 1 by removing unnecessary multiplexors. Compared to prior architectures for discrepancy coefficient computation, our architecture requires less gate in the case that the maximum multiplicity is two. The candidate polynomial updating architecture proposed in [10] has less gate count than other published designs. The critical path in this architecture consists of one multiplier, three Mux and one inverter. A $GF(2^8)$ inverter can be implemented by a 256×8 look-up table (LUT). In addition, its delay usually equals to that of 2-3 serially concatenated XOR gates. Replacing the delay of the inverter with that of 2 XOR gates, the critical path of single iteration interpolation architecture includes 12 gates. In total, the implementation of this architecture needs 2055 XOR gates, 1375 AND gates, 29 OR gates, 184 Muxes, and 273 registers. Besides the memory

TABLE I

GATE COUNTS AND CRITICAL PATHS

	area	critical path
Multiplier	64XOR+48AND	6XOR+1AND
Adder	8XOR	1XOR
DCC	784XOR+552AND	7XOR+ 2AND
PU	2152XOR+1488AND+552Mux	8XOR+1AND+3Mux
Control	61XOR+102AND+106OR+82Mux	1XOR+4AND+4OR+3Mux
Total	2997XOR+2142AND+106OR+634Mux	N/A

for the LUT, the same amount of memory is needed to store the polynomial coefficients as in our design. Accordingly, the memory requirement is $256+297=553$ bytes.

The DCC and PU architectures take in coefficients serially. Hence, the number of clock cycles required in each interpolation iteration is decided by the maximum X-degree of the candidate polynomials. This degree increases uniformly with interpolation iteration. Therefore, the latency of the overall interpolation can be derived by multiplying the average number of clock cycles required by each iteration with the number of interpolation iterations. In addition, in each iteration, extra clock cycles used for pipelining are needed to take into account and the total pipelining latency is $\xi_{dcca}+\xi_{dccb}+\xi_{pu}=1+4+2=7$ in our design. From simulations, it can be derived that the average maximum X-degree of the polynomials, d_x, is 13 when the signal-to-noise ratio (SNR) is high. Accordingly, the average number of clock cycles required for each combined interpolation iteration is $d_x+7=20$. At low SNR, d_x becomes 8. Hence, the average number of clock cycles required for each combined interpolation iteration becomes 15. After applying re-encoding and coordinate transformation, the interpolation only needs to be carried out over at most $n-k=16$ interpolation points of multiplicity two, or 16 pairs of points with multiplicity one, or any combination of them. As a result, at most 16 combined interpolation iterations are required in our proposed scheme. In the original Nielson's interpolation, the average number of clock cycles required for each iteration is d_x+5. However, the number of interpolation iterations depends the number of constraints need to be satisfied. At high SNR, most of the interpolation points have multiplicity two, each of which adds 3 constraints. Hence, the number of interpolation iterations is at most $3\times 16=48$. On the other hand, at lower SNR, most of the interpolation points have multiplicity one, and at least $2\times 16=32$ iterations are required in this case.

Each AND gate or OR gate occupies 3/4 of the area of an XOR, each memory cell and Mux has the same area as an XOR, and each register requires about 3 times of the area of an XOR. Taking this into account, for a (255, 239) RS code, the area requirement of the proposed interpolation architecture is equivalent to that of $2997+(2142+106)\times 0.75+634+297\times 8+345\times 3=8728$ XOR gates. The area required by the single iteration interpolation architecture equals that of $2055+(1375+29)\times 0.75+184+553\times 8+273\times 3=8535$ XOR gates. Hence, our architecture only requires 2% more area. The critical path of our architecture is the same as that in the previous one. In high SNR scenarios, our architecture can achieve a throughput of $(18\times 48)/(20\times 16)=2.7$ times. Hence, our architecture is 165% more efficient in terms of speed/area ratio. When the SNR is low, our architecture is $(13\times 32)/(15\times 16)=1.73$ times faster, and thus is 70% more efficient.

The application of the proposed combined polynomial updating scheme is not limited to the architectures discussed in this paper. Considering the interpolation architectures in [11], the proposed scheme can also be employed to achieve further speedup.

VI. CONCLUSION

A novel scheme is proposed in this paper to combine interpolation iterations. In our scheme, the discrepancy coefficients from multiple iterations are computed using a 'look-ahead' method. Then the polynomial updating of these iterations are combined. Applying the proposed scheme, our combined interpolation architecture can achieve significant speedup at the cost of negligible area overhead. In the case that more candidate polynomials are involved, there are more possibilities with regard to which polynomial is chosen as the minimum polynomial in each iteration. As a result, combining interpolation iterations would lead to larger number of different formats for the updated polynomials. We will study if the scheme proposed in this paper can be extended to the cases where more polynomials are involved.

ACKNOWLEDGEMENT

This work was supported by NSF under grant 0802159 and 0708685.

REFERENCES

[1] R. Koetter and A. Vardy, "Algebraic soft-decision decoding of Reed-Solomon codes," *IEEE Trans. Inf. Theory*, vol. 49, no. 11, pp. 2809-2825, Nov. 2003.

[2] J. Bellorado and A. Kavcic, "A Low-Complexity Method for Chase-Type Decoding of Reed-Solomon Codes," *Proc. ISIT*, pp. 2037-2041, Jul. 2006.

[3] J. Jiang and K. Narayanan, "Algebraic soft decision decoding of Reed-Solomon codes using bit-level soft information" *Proc. Allerton Conference on Communications, Control and Computing*, 2006.

[4] V. Guruswami and M. Sudan, "Improved decoding of Reed-Solomon and Algebraic-Geometric codes," *IEEE Trans. Inform. Theory*, vol. 45, pp. 1755-1764, Sep 1999.

[5] R. R. Nielson, List Decoder of Linear Block Codes, Ph.D dissertation, Technical University of Denmark, 2001.

[6] R. Koetter, On Algebraic Decoding of Algebraic-Geometric and Cyclic Codes, Ph.D. dissertation, Dept. of Elec. Engr., Linkoping University, Linkoping, Sweden, 1996.

[7] W. J. Gross, F. R. Kschischang, R. Koetter and P. Gulak, "A VLSI architecture for interpolation in soft-decision decoding of Reed-Solomon codes," *Proc. of SiPS*, pp. 39-44, San Diego, Oct. 2002.

[8] R. Koetter and A. Vardy, "A complexity reducing transformation in algebraic list decoding of Reed-Solomon codes," *Proc. of ITW*, Paris, France, Mar. 2003.

[9] A. Ahmed, N. Shanbhag and R. Koetter, "Systolic interpolation architectures for soft-decoding Reed-Solomon codes", *Proc. of SiPS*, pp. 81-86, Seoul, Korea, Aug. 2003.

[10] A. Ahmed, R. Koetter and N. Shanbhag, "VLSI architecture for soft-decision decoding of Reed-Solomon codes," *Proc. of ICC*, Paris, France, Jun. 2004.

[11] Z. Wang and J. Ma, "High-speed interpolation architecture for soft-decision decoding of Reed-Solomon codes," *IEEE Trans. on VLSI Systems*, pp. 937-950, Sep. 2006.

[12] X. Zhang, "Reduced complexity interpolation architecture for soft-decision Reed-Solomon decoding," *IEEE Trans. on VLSI Systems*, vol. 14, no. 10, pp. 1156-1161, Oct. 2006.

[13] K. Lee and M. O'Sullivan, "An interpolation algorithm using Grobner bases for soft-decision decoding of Reed-Solomon codes," *Proc. of ISIT*, Seattle, Washington, Jul. 2006.