HIGH-SPEED FACTORIZATION ARCHITECTURE FOR SOFT-DECISION REED-SOLOMON DECODING

Xinmiao Zhang

Department of Electrical Engineering and Computer Science
Case Western Reserve University, 10900 Euclid Ave., Cleveland, OH 44106-7071

ABSTRACT

Reed-Solomon (RS) codes are among the most widely utilized error-correcting codes in modern communication and computer systems. Among the decoding algorithms of RS codes, the recently proposed Koetter-Vardy (KV) soft-decision decoding can achieve substantial coding gain, while has a polynomial complexity. One of the major steps of the KV decoding is the factorization. The root computation involved in each iteration level of the factorization is traditionally implemented by exhaustive search. A fast factorization architecture has been proposed to circumvent the exhaustive root search from the second iteration level by using a root-order prediction scheme. However, the root computation in the first iteration level is still carried out by exhaustive search, which accounts for a significant part of the overall factorization latency. In this paper, a novel iterative prediction scheme is proposed to compute the roots in the first iteration level. The proposed scheme can substantially reduce the average latency of the factorization, while only incurs negligible area overhead. Applying this scheme to a (255, 239) RS code, a speedup of 36% can be achieved.

1. INTRODUCTION

Reed-Solomon (RS) codes have very broad applications. They can be found in magnetic and optical recording, spread spectrum wireless systems, as well as satellite and deep-space communications. Since RS codes were introduced in 1960s, tremendous efforts have been devoted to developing efficient and high-gain decoding algorithms. The well-known Berlekamp-Massey algorithm (BMA)[1] can correct up to $d_{min}/2 = (n-k+1)/2$ errors for an (n,k) RS code. Comparatively, list-decoding algorithms attempt to correct more errors by finding all the codewords within a distance that is longer than $d_{min}/2$ from the received word. A breakthrough in list-decoding was achieved by the Sudan [2] and Guruswami-Sudan (GS) [3] algorithms. These algorithms are based on an algebraic interpolation technique. By forcing all the interpolation points in the Sudan algorithm to have higher multiplicities, the GS algorithm can correct up to $n - \sqrt{kn}$ errors. Higher coding gain can be also achieved by soft-decision decoding algorithms through making use of the probability information from the channel. Various soft-decision algorithms have been proposed. However, they can either only achieve relatively low coding gain or have exponential complexity. Recently, Koetter and Vardy extended the GS algorithm by incorporating the probability information into the algebraic interpolation process [4]. By forcing the interpolation points with higher reliability to have higher multiplicities, this algorithm can achieve substantial coding gain while its complexity is polynomial with respect to the codeword length.

The major steps of the Koetter-Vardy (KV) algorithm are the interpolation and factorization. Re-encoding and coordinate transformation have been introduced to reduce the interpolation complexity [5, 6, 7, 8]. Applying these techniques, the number of iterations need to be carried out in the factorization can be also reduced. Each iteration of the factorization mainly consists of root computation over finite fields and polynomial updating. The root computation is traditionally implemented by exhaustive search, which requires long latency for long codes. A fast factorization architecture was proposed to reduce the average latency associated with the root computation [9]. In this architecture, the exhaustive root search from the second iteration level can be circumvented with more than 99% probability by employing a root-order prediction scheme. In addition, applying the root-order prediction scheme, the roots of two adjacent iteration levels can be computed in a short time. Based on this feature, a partial parallel factorization architecture was developed to combine the polynomial updating in adjacent iteration levels [10]. However, the speedup of this architecture comes at the expense of significantly increased area requirement. In both of these architectures, the root computation in the first iteration level is still carried out by exhaustive search, which accounts for a significant part of the overall factorization latency.

In this paper, a novel iterative prediction scheme is proposed to compute the roots in the first iteration level. This scheme carries out up to three trial-and-error direct root computations before exhaustive search is employed. As a result, the average latency associated with the first iteration root computation can be significantly reduced. In addition, all the hardware units required by this scheme can be shared with the units already exist in the architectures of [9] and [10]. Therefore, the extra hardware required by this scheme is negligible. Applying the proposed algorithm to a (255,

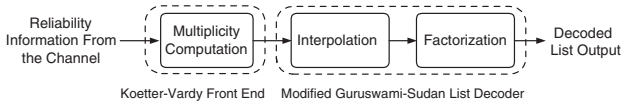

Fig. 1. The KV decoding algorithm

239) RS code, a speedup of 36% can be achieved, while the extra area requirement is around 1%.

The structure of this paper is as follows. Section 2 describes the factorization step in the KV algorithm and the root-order prediction scheme. Section 3 presents the iterative prediction scheme for the first iteration level root computation. Then the factorization architecture, which has incorporated the proposed scheme, is presented in Section 4. Section 5 summarizes this paper.

2. FACTORIZATION IN THE KV ALGORITHM

In this paper, we only consider RS codes constructed over finite fields of characteristic two. Fig. 1 illustrates the block diagram of the KV algorithm. The multiplicity computation step decides on the interpolation points and their relative multiplicities according to the probability information from the channel. In practice, this step can be implemented as multiplying a probability information matrix by a non-negative real number, λ, then followed by the floor function. Comparatively, the interpolation and factorization steps are much more hardware demanding. For an (n, k) code, the interpolation step finds a bivariate polynomial $Q(X, Y)$ with minimum weighted degree that passes each non-trivial interpolation point with at least its associated multiplicity. Then the factorization step finds all factors of $Q(X, Y)$ in the form of $Y - f(X)$ with $\deg(f(X)) < k$. Here $\deg(.)$ denotes the degree of the polynomial. Among various factorization algorithms, the one proposed in [11] is suitable for efficient hardware implementations. This algorithm can be described by the pseudo codes listed in Algorithm A.

Algorithm A: Factorization Algorithm

Initialization: iteration level $i = 0$

$Reconstruct(Q(X, Y), i)$

{

find the largest non-negative integer v, such that X^v divides $Q(X, Y)$

F1: $\bar{Q}(X, Y) = Q(X, Y)/X^v$

F2: find all the roots of $\bar{Q}(0, Y)$ in $GF(2^q)$

for each root γ of $\bar{Q}(0, Y)$ do

$\psi[i] = \gamma$

if $i = k - 1$

output $(\psi[0], \psi[1], \cdots, \psi[k-1])$

else

F3: $\tilde{Q}(X, Y) = \bar{Q}(X, Y + \gamma)$

F4: $\hat{Q}(X, Y) = \tilde{Q}(X, XY)$

call $Reconstruct(\hat{Q}(X, Y), i + 1)$

}

In Algorithm A, the k roots in each output vector form the coefficients of a degree $k - 1$ polynomial. The polynomials corresponding to all output vectors contain the $f(X)$ factors as a subset. Applying re-encoding and coordinate transformation, the number of factorization iteration levels can be reduced from k to 2τ, where τ is the number of errors intend to be corrected in the k most reliable code positions. After the total iteration number has been reduced, further speedup of the factorization can be achieved by optimizing the computations involved in each iteration level. As it can be observed from Algorithm A, each iteration mainly consists of the root computation in the F2 step and the polynomial updating in the F3 step. The F1 and F4 steps are trivial. They can be implemented by using a register to keep track of the address displacements. Traditionally, the root computation over finite fields are implemented by exhaustive search. For long codes, the F2 step has very long latency.

The latency of the root computation from the second iteration level can be reduced by the root-order prediction scheme proposed in [9]. From Algorithm A, it can be only derived that the degree of $\bar{Q}(0, Y)$ is at most the order, r, of the corresponding root, γ, in the previous iteration level. However, it is found from simulations that in most cases the degree of $\bar{Q}(0, Y)$ equals r and $\bar{Q}(0, Y)$ has a single root of order r. Therefore, if a root with order r is found in iteration level i, then it is predicted that the corresponding $\bar{Q}(0, Y)$ in iteration level $i + 1$ can be expressed as

$$\bar{Q}(0, Y) = (Y + \gamma')^r = \bar{q}_{0,0} + \bar{q}_{0,1}Y + \cdots + \bar{q}_{0,r}Y^r \quad (1)$$

where $\bar{q}_{i,j}$ is the coefficient of X^iY^j in $\bar{Q}(X, Y)$. From (1), it can be derived that

$$(\gamma')^w = \bar{q}_{0,r-w}\bar{q}_{0,r}^{-1}, \quad (2)$$

where w is the minimum positive integer satisfying $\binom{r}{w}$ odd. Such a w can be only in the form of $w = 2^e$ $(e \in Z^*)$. In this case, the root, γ', in iteration level $i + 1$, can be computed by cyclically shifting the bits in the normal basis representation of $\bar{q}_{0,r-w}\bar{q}_{0,r}^{-1}$, e, bit positions to the direction of the least significant bit. In addition, it can be derived that $\bar{q}_{0,r}$ does not change in later iteration levels if the predictions are correct. Hence, the value of $\bar{q}_{0,r}^{-1}$ can be stored and reused.

The prediction failure rate of this scheme is very low for practical applications of RS codes. However, ignoring these cases may bring significant performance degradation to the KV algorithm. From the definition of root order, γ' is a rth-order root of $\bar{Q}(0, Y)$ if and only if the coefficients of $1, Y, Y^2, \cdots, Y^{r-1}$ in $\bar{Q}(0, Y + \gamma')$ are zero, while the coefficient of Y^r is not. Since $\bar{Q}(0, Y + \gamma')$ is computed in the F3 step, the checking of whether the prediction is correct can be done by observing the output of this step. In the case of prediction failure, the roots are re-computed either through direct computation or exhaustive search when the degree of $\bar{Q}(0, Y)$ is two or higher, respectively.

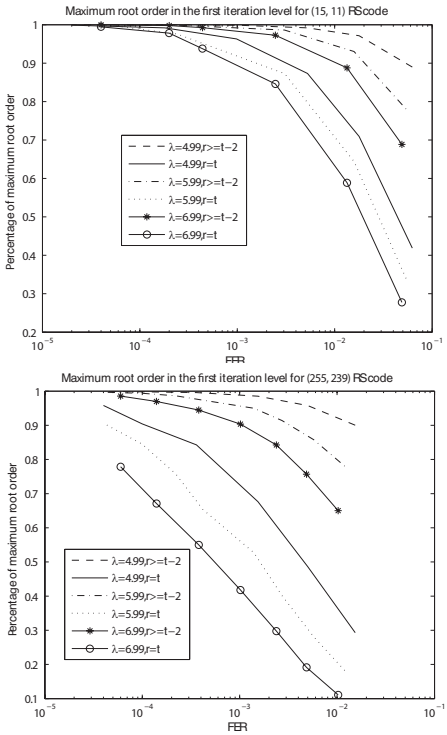

Fig. 2. Simulation results for maximum root order

In the fast factorization architecture of [9], the root computation in the first iteration level is still carried out by exhaustive search, which may account for a significant part of the overall factorization latency. For example, the exhaustive root search for a $(255, 239)$ RS code in the first iteration level can consume 31% of the fast factorization latency. In the partial parallel factorization architecture of [10], the factorization latency is reduced by combining the polynomial updating from adjacent iteration levels. However, the latency associated with the exhaustive root search in the first iteration level does not change. Therefore, the first iteration root computation accounts for a even more significant part of the overall latency in this architecture. In the next section, we introduce a novel iterative prediction scheme for the root computation in the first iteration level. This scheme carries out up to three direct root computation trials before exhaustive search is employed. As a result, the average latency for the first iteration root computation is substantially reduced. In addition, this scheme only requires negligible extra area compared to the area required by the entire factorization architecture.

3. ROOT COMPUTATION IN THE FIRST ITERATION LEVEL

The $\bar{Q}(0, Y)$ in the first iteration level may have multiple roots. From first glance, no information on these roots can be directly derived. However, from simulations, we found it is highly likely that the maximum order of the roots in the first iteration level is very close to the degree of $\bar{Q}(0, Y)$. Assume $\deg(\bar{Q}(0, Y)) = t$ and the maximum order of the roots in the first iteration level is r, Fig. 2 illustrates some simulation results on the probability of $r = t$ and $r \geq t - 2$ for $(15, 11)$ and $(255, 239)$ codes with different values of λ. These results are obtained under AWGN channel with BPSK modulation. From Fig. 2, it can be observed that the probabilities of $r = t$ or $r \geq t - 2$ decrease with increasing λ. In addition, it is more likely that the maximum root order is close to $\deg(\bar{Q}(0, Y))$ in the case that the frame error rate (FER) is low.

In the case of $r = t$, the root can be computed according to (2). However, the probability of this case is low when the FER is high, code is long or λ is large. Fortunately, under certain circumstances, we can still compute a root of $\bar{Q}(0, Y)$ directly from its coefficients if the root order is less than t. Assume among the roots of $\bar{Q}(0, Y)$, there is α with order $r < t$. Then $\bar{Q}(0, Y)$ can be written as

$$\bar{Q}(0, Y) = (Y + \alpha)^r b(Y). \tag{3}$$

Since $\deg(\bar{Q}(0, Y)) = t$, $\deg(b(Y)) = t - r$. Therefore, (3) can be expanded as

$$\bar{Q}(0, Y) = \left(\binom{r}{0} Y^r + \binom{r}{1} \alpha Y^{r-1} + \cdots + \binom{r}{r} \alpha^r \right) \times (b_{t-r} Y^{t-r} + b_{t-r-1} Y^{t-r-1} + \cdots + b_0). \tag{4}$$

Multiplying out the right side of (4), it can be observed that the coefficients of $\bar{Q}(0, Y)$ equals the sum of the terms listed in Table 1. In this table, we assume $r > (t - r)$. Otherwise, the terms with negative powers of α should be replaced by zero.

Since the binomial coefficients are computed over $GF(2)$, they are either zero or one. Therefore, in Table 1, there may exist two columns, in which the nonzero binomial coefficients have the same pattern. In this case, each term in one column equals the the term in the same row of the other column multiplied by α^s, where s is the difference between the indices of the two columns. Accordingly, α^s can be computed by a division between the coefficients of $\bar{Q}(0, Y)$ corresponding to these two columns. For example, in the case of $r = 5$ and $t = 9$, the binomial coefficients are listed in Table 2. It can be observed that the '1's in the column for $\bar{q}_{0,7}$ have the same pattern as those in the column for $\bar{q}_{0,3}$. In addition, the difference of the power of α between the corresponding terms in these two columns is $7 - 3 = 4$. Therefore, α^4 can be computed as $\bar{q}_{0,3}/\bar{q}_{0,7}$. Furthermore, it can be observed from Table 2 that α^4 can be also computed as $\bar{q}_{0,2}/\bar{q}_{0,6}$.

For practical applications, the maximum order of the roots of $\bar{Q}(0, Y)$ does not exceed eight. By exhaustive listing of the binomial coefficient patterns, it can be observed that there exist at least one pair of coefficients that can be

Table 1. The terms for the coefficients of $\bar{Q}(0, Y)$

	$\bar{q}_{0,t}$	$\bar{q}_{0,t-1}$	$\bar{q}_{0,t-2}$	$\cdots$	$\bar{q}_{0,t-r}$	$\bar{q}_{0,t-r-1}$	$\cdots$	$\bar{q}_{0,0}$
$\sum$	$\binom{r}{0} b_{t-r}$	$\binom{r}{1} \alpha b_{t-r}$ $\binom{r}{0} b_{t-r-1}$	$\binom{r}{2} \alpha^2 b_{t-r}$ $\binom{r}{1} \alpha b_{t-r-1}$ $\binom{r}{0} b_{t-r-2}$	$\cdots$ $\cdots$ $\cdots$ $\ddots$ $\cdot$	$\binom{r}{r} \alpha^r b_{t-r}$ $\binom{r}{r-1} \alpha^{r-1} b_{t-r-1}$ $\binom{r}{r-2} \alpha^{r-2} b_{t-r-2}$ $\vdots$ $\binom{r}{r-(t-r)} \alpha^{r-(t-r)} b_0$	$\binom{r}{r} \alpha^r b_{t-r-1}$ $\binom{r}{r-1} \alpha^{r-1} b_{t-r-2}$ $\vdots$ $\binom{r}{r-(t-r-1)} \alpha^{r-(t-r-1)} b_0$	$\binom{r}{r} \alpha^r b_{t-r-2}$ $\ddots$ $\cdots$	$\binom{r}{r} \alpha^r b_0$

Table 2. The binomial coefficients for the case of $r = 5$ and $t = 9$

$\bar{q}_{0,9}$	$\bar{q}_{0,8}$	$\bar{q}_{0,7}$	$\bar{q}_{0,6}$	$\bar{q}_{0,5}$	$\bar{q}_{0,4}$	$\bar{q}_{0,3}$	$\bar{q}_{0,2}$	$\bar{q}_{0,1}$	$\bar{q}_{0,0}$
1	1	0	0	1	1				
	1	1	0	0	1	1			
		1	1	0	0	1	1		
			1	1	0	0	1	1	
				1	1	0	0	1	1

used to directly compute α^s if $deg(b(Y)) = t - r$ satisfies the conditions listed in Table 3. In addition, a possible value of s, which applies to all possible $t - r$ for a given r, is provided in Table 3. For example, if $r = 4$, then as long as $\deg(b(Y)) < 7$, there exist at least one pair of coefficients of $\bar{Q}(0, Y)$ with their quotient equals α^4. It can be also observed from Table 3 that s always equals to some non-negative integer power of two. In this case, α can be computed from α^s by a cyclical shift if normal basis representation for finite field elements is employed.

Table 3. The conditions for directly computing α^s

r	2	3	4	5	6	7	8
$t - r$	< 3	< 3	< 7	< 6	< 7	< 7	< 15
s	2	1	4	4	2	1	8

The maximum order of the roots for $\bar{Q}(0, Y)$ in the first iteration level is unknown. The only information we have is that it is close to the degree of $\bar{Q}(0, Y)$ with high probability. Therefore, we start by predicting that $\bar{Q}(0, Y)$ has a single root γ with order $r = t$, and compute γ by (2). In the case of prediction failure, we make a second prediction: $r = t - 1$, and compute γ through dividing proper coefficients of $\bar{Q}(0, Y)$. In the case that the second prediction fails, r is decreased by one again. This process can be carried out iteratively until a real root is found or the condition in Table 3 is no longer satisfied. The correctness of the predictions can be checked by observing the coefficients of $\bar{Q}(0, Y+\gamma)$.

If a real root is found through the above iterative prediction process, it must be the root with the highest order. However, $\bar{Q}(0, Y)$ may have other roots of lower order, all of them needs to be found in the factorization algorithm. Assume a root, γ, with order r is directly computed. Then

$$\bar{Q}(0, Y) = (Y + \gamma)^r b(Y).$$

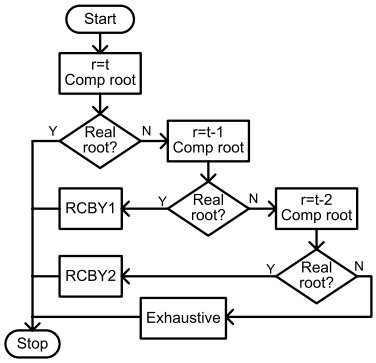

Fig. 3. The procedure for first iteration root computation

Hence the rest of the roots can be found by solving $b(Y) = 0$. The coefficients of $b(Y)$ can be derived through a polynomial expansion and a polynomial division. However, this process requires extra area and incurs long latency. Through the polynomial updating in the F3 step

$$\bar{Q}(0, Y + \gamma) = Y^r b(Y + \gamma)$$

is computed. Therefore, the coefficients of the shifted factor $b(Y + \gamma)$ can be directly observed from the output of the F3 step. Accordingly, $b(Y + \gamma) = 0$ can be solved instead of $b(Y) = 0$. Adding γ back to the computed roots, the roots of $b(Y) = 0$ can be derived. The roots for polynomials with degree higher than two can not be easily computed. In addition, from Fig. 2, the probability for the maximum root order to be larger than or equal to $t-2$ is high. Therefore, we will limit the iterative prediction process to until $r = t - 2$. Another advantage to stop the process at this point is that, as it can be observed from Table 3, it is guaranteed that α^s can be directly computed from the coefficients of $\bar{Q}(0, Y)$ when $t - r \leq 2$. In the case that all predictions failed, exhaustive search is employed to find the roots of $\bar{Q}(0, Y)$. In summary, the root computation for the first iteration level of the factorization can be carried out according to the iterative prediction procedure illustrated in Fig. 3. In this figure, the RCBY1 and RCBY2 blocks implement the root computation for $b(Y)$ when its degree is one and two, respectively.

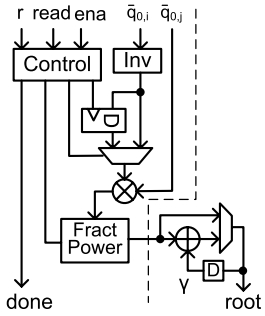

Fig. 4. The Comp root architecture

4. VLSI ARCHITECTURE FOR THE FACTORIZATION STEP

This section first presents detailed architectures for the computation units required by the proposed first iteration root computation scheme. Then we demonstrate how to incorporate these units into the fast factorization architecture proposed in [9] to further reduce the latency.

Employing the iterative prediction scheme, it only takes one inversion and one multiplication to compute a root in the Comp Root block for each prediction. As it was mentioned in the previous section, there may exist more than one pair of coefficients that can be used for root computation. The pairs of coefficients can be carefully selected such that the divisors in these pairs remain the same through the three predictions as much as possible. In this case, the inversions required in later predictions can be saved. For example, if $\deg(\bar{Q}(0,Y)) = 7$, then in the first prediction, γ can be computed as $\bar{q}_{0,6}/\bar{q}_{0,7}$. In the case that this prediction fails, we predict $r = 6$, and γ^2 can be computed as $\bar{q}_{0,5}/\bar{q}_{0,7}$. Similarly, if $r = 5$, γ^4 can be computed as $\bar{q}_{0,3}/\bar{q}_{0,7}$. Therefore, by choosing the pairs of coefficients as above, the complicated inversion can be saved in the latter two predictions. For a degree one polynomial $b(Y) = b_1 Y + b_0$, its root can be computed as $b_1^{-1} b_0$. Hence, the Comp root and RCBY1 can be implemented by similar architectures. In addition, as it can be observed from Fig. 3, the Comp root and RCBY1 are activated at different time. Accordingly, the Comp root architecture illustrated in Fig. 4 can be used to implement both the Comp root and RCBY1 blocks.

In the Comp root architecture, depends on whether the divisor changes or not in the three predictions, the inverse value can be either computed by the Inv block or can be read from the register. The function of the Fract Power block is to compute γ from the value of its power by a cyclical shift. This block also includes the conversion to and from normal basis representations. As it has been discussed in the previous section, it is more efficient to compute the root of $b(Y+\gamma)$ instead of $b(Y)$. Hence, the coefficients fed into this architecture for the RCBY1 implementation are actually proper coefficients of $\bar{Q}(0, Y+\gamma)$. Accordingly, γ needs to be added back to the output of the Fract Power block to derive the actual root. The only difference between the Comp root architecture and the RC3 architecture in [9] is the parts attached to the output of the Fract Power block.

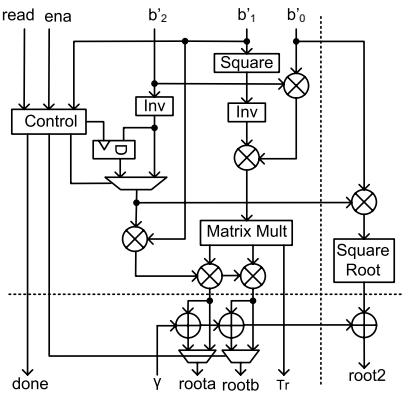

Fig. 5. The architecture for RCBY2 block

The roots of a degree two polynomial can be computed directly by the algorithm developed in [1]. According to this algorithm, the RCBY2 block can be implemented by the architecture introduced in Fig. 5. Similarly, the root computation are actually carried out on $b(Y+\gamma) = b'_2 Y^2 + b'_1 Y + b'_0$. Hence γ needs to be added back to the computed roots. In the case that b'_1 is zero, the second-order root, 'root2', can be computed by the right part of RCBY2 architecture. When $b'_1 \neq 0$, the degree two polynomial only has roots when the trace function (Tr) of $b'_2 b'_0 (b'^{-1}_1)^2$ is zero. In this case, the two simple roots, 'roota' and 'rootb', can be computed as multiplying the vector formed by the standard basis representation of $b'_2 b'_0 (b'^{-1}_1)^2$ by a pre-computed matrix. For detailed explanations, the interested reader is referred to [9]. The top left part of this architecture is the same as the MRC2 block in the fast factorization architecture proposed in [9]. The RCBY2 block is only used in the first iteration root computation, while the MRC2 block is used from the second iteration level. Therefore, the RCBY2 architecture can be shared for both blocks. Accordingly, two multiplexors are added to the 'roota' and 'rootb' outputs, as illustrated at the bottom of Fig. 5, to enable resource sharing.

Incorporating the iterative prediction scheme for the first iteration level root computation, the factorization algorithm can be implemented by the architecture illustrated in Fig. 6. Compared to the fast factorization architecture in [9], the only differences are the blocks shown in shade: the Comp root block replaced one of the RC3 block, the RCBY2 block replaced the MRC2 block and two multiplexors are added. In addition, the Control block is slightly different and some connections are added. The PU and PS units implement the F3 step, the F4 and F1 steps, respectively. The detailed description of how the fast factorization architecture works can be found in [9]. Compared to the RC3 and MRC2 architecture, the Comp root and RCBY2 architecture as illustrated in Fig. 4 and Fig. 5 requires an extra of four adders, three 2:1

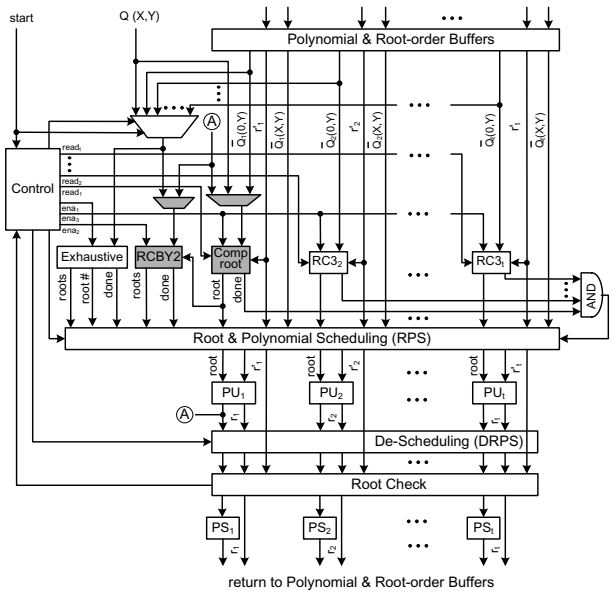

Fig. 6. The factorization architecture

multiplexors, one multiplier and one matrix multiplication for the square root block. For RS codes of length 255, the computations are carried out over $GF(2^8)$. Each $GF(2^8)$ multiplier requires 77 XOR gates and 64 AND gates, while each adder requires 8 XOR gates. On average, the multiplication of a 8×8 binary matrix takes $7 \times 8/2 = 28$ XOR gates. In addition, each 1-bit 2:1 multiplexer is equivalent to an XOR gate in terms of area and delay. Hence, the RCDY2 and Comp root blocks require extra 161 XOR gates and 64 AND gates. Taking the 56 XOR gates required by the two shaded multiplexors into account, the iterative prediction scheme only adds 227 XOR gates and 64 AND gates, and slightly larger control logic. This extra area only accounts for around 1% of the overall area of the fast factorization architecture.

Table 4. The # of pipelining stages for building blocks

	# of stages		# of stages
RCBY2	5	Comp root	4
Exhaustive	3	RC3	4
RPS	1	PU	3
DRPS	1	Root Check	1

The factorization architecture is pipelined, and the critical path consists of 10XOR gates and 1 AND gate. Table 4 summarizes the number of pipelining stages each block in Fig. 6 can be divided into. Assume the conditional prediction failure rates for the first through the third trials are $PFR1$, $PFR2$ and $PFR3$, iteratively. Then the number of clock cycles needed for the first iteration root computation is $4+PFR1\times(1+3+1+1+(1-PFR2)\times(4+1+3+4)+PFR2\times(4+1+3+1+1+(1-PFR3)\times(4+1+3+5)+PFR3\times(4+1+3+1+1+256+3)))$. For a (255,239) code with $\lambda = 6.99$, $PFR1 = 58\%$, $PFR2 = 44\%$ and $PFR3 = 37\%$ at $FER = 10^{-3}$ In this case, the first iteration root computation takes an average of 42 clock cycles. Compared to the exhaustive search approach, which requires 256+3 clock cycles out of the overall latency of 823 clock cycles, the proposed iterative prediction scheme brings an speedup of 36%. Since the prediction failure rates decrease with FER, the speedup can be brought by this scheme will be more significant for applications requiring lower FER.

5. CONCLUSION

A novel iterative prediction scheme is proposed in this paper to speed up the first iteration root computation in the factorization step of the KV algorithm. This scheme carries out up to three trial-and-error direct root computations before exhaustive search is employed. In addition, all the computation units required for this scheme can be shared with the blocks already exist in the fast factorization architecture. Hence, the extra area required by incorporating this scheme is negligible. The speedup can be brought by the proposed scheme is heavily dependent on code rate, code length, maximum interpolation multiplicity and channel model. Further study for codes with different settings need to be carried out to optimize this scheme.

6. REFERENCES

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

[2] M. Sudan, "Decoding of Reed-Solomon codes beyond the error correction bound," *Journal of Complexity*, vol. 12, pp. 180-193, 1997.

[3] 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.

[4] R. Koetter and A. Vardy, "Algebraic soft-decision decoding of Reed-Solomon codes," *IEEE Trans. Inform. Theory*, vol. 49(11), pp. 2809-2825, Nov. 2003.

[5] W. J. Gross, F. R. Kschischang, R. Koetter and P. Gulak, "A VLSI architecture for interpolation in soft-decision decoding of Reed-Solomon codes," *Proceedings of the 2002 IEEE Workshop on Signal Processing Systems*, pp. 39-44, San Diego, Oct. 2002.

[6] R. Koetter and A. Vardy, "A complexity reducing transformation in algebraic list decoding of Reed-Solomon codes," *Proceedings of ITW2003*, Paris, France, Mar. 2003.

[7] A. Ahmed, R. Koetter and N. Shanbhag, "VLSI architecture for soft-decision decoding of Reed-Solomon codes," *Proceedings of ICC2004*, Paris, France, Jun. 2004.

[8] W. J. Gross, *Implementation of Algebraic Soft-Decision Reed-Solomon Decoders*, Ph.D. dissertation, Dept. of Elec. Comp. Engr., University of Toronto, Toronto, QC, Canada, 2003.

[9] X. Zhang and K. K. Parhi, "Fast factorization in soft-decision Reed-Solomon decoding," *IEEE Trans. on VLSI Systems*, vol. 13(4), pp. 413-426, Apr. 2005.

[10] X. Zhang, "Partial parallel factorization in soft-decision Reed-Solomon decoding," *Proc. of ACM Great Lake Symposium*, Philadelphia, PA, Apr. 2006.

[11] R. M. Roth and G. Ruckenstein, "Efficient decoding of Reed-Solomon codes beyond half the minimum distance," *IEEE Trans. Inform. Theory*, vol. 46, no. 1, Jan.2000.