Hardware Complexities of Algebraic Soft-decision Reed-Solomon Decoders and Comparisons

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

***Abstract*—Algebraic soft-decision (ASD) decoding of Reed-Solomon (RS) codes can achieve significant coding gain over hard-decision decoding. For practical implementation purpose, ASD algorithms with simple multiplicity assignment schemes are preferred. This paper answers the question of which practical ASD algorithm has lower hardware complexity. In addition, decoder complexity comparisons for an example (458, 410) RS code constructed over $GF(2^{10})$ are presented. This paper also provides discussions on how the hardware complexities of ASD decoders change with codeword length, code rate and other parameters.**

I. Introduction

Reed-Solomon (RS) codes can be found in many digital communication and storage systems due to their good burst error-correcting capability. Recently, algebraic soft-decision (ASD) decoding algorithms of RS codes have been developed [1], [2], [3], [4], [5], [6], [7], [8]. In these algorithms, the reliability information from the channel is incorporated into the algebraic interpolation process proposed by Guruswami and Sudan [9], [10]. As a result, these algorithms can achieve significant coding gain over hard-decision decoding (HDD), such as the Berlekamp-Massey algorithm (BMA) [11] and modified Euclidean algorithm (MEA) [12], with a complexity that is polynomial with respect to codeword length.

ASD algorithms have three steps: multiplicity assignment, interpolation and factorization. ASD algorithms are different in the multiplicity assignment. However, they share the same interpolation and factorization steps. Although the multiplicity assignment schemes proposed in [2], [3], [4] lead to ASD algorithms with higher error-correcting capability, these schemes themselves require very complex computations. For the purpose of practical implementation, ASD algorithms with simple multiplicity assignment schemes are preferred. Therefore, this paper considers the implementation of the Kötter-Vardy (KV) [1], bit-level generalized minimum distance (BGMD) [6], and test-vector-based low-complexity Chase (LCC) [5] ASD algorithms. The algorithm in [7] combines the Chase and generalized minimum distance (GMD) [13] decoding. The algorithm in [8] repeats error-and-erasure decoding for every possible pattern of erasures from a large number of unreliable code positions. Despite that these two algorithms may achieve higher coding gain than the LCC algorithm, they need to test a much larger number of vectors and thus their complexities are significantly higher. Hence, the hardware implementations of these two algorithms are not considered in this paper.

This work is supported in part by the NSF under grant ECCS-0846331 and AFOSR under grant FA9550-09-1-0289

Various techniques have been developed to simplify the hardware implementation of the interpolation and factorization steps of ASD algorithms. This paper first summarizes how the hardware complexities of the three practical ASD algorithms, namely the KV, BGMD and LCC algorithms, can be reduced by available techniques. Then the question of which ASD algorithm has lower complexity is answered. As an example, the hardware complexity analyses for the three ASD decoders for a (458, 410) RS code constructed over $GF(2^{10})$ are provided. The decoder complexity varies with parameters such as codeword length, code rate, and test vector number. How the decoder complexity changes with different parameters is also discussed in this paper.

The structure of this paper is as follows. Section II introduces ASD algorithms and shows their error-correcting performance. Section III presents available techniques for decoder complexity reduction. The decoder complexity analysis for the example code is provided in Section IV. Then Section V discusses how the complexities of ASD decoders change with different parameters. Conclusions are drawn in Section VI.

II. ASD decoding algorithms

Without loss of generality, this paper considers an (n, k) RS code constructed over $GF(2^q)$ $(q \in Z^+)$. For a primitive code, $n = 2^q - 1$. Assume that the k message symbols $f_0, f_1, f_2, \cdots f_{k-1}$ form the coefficients of a message polynomial $f(x) = f_0 + f_1 x + \cdots f_{k-1}x^{k-1}$, and the ith symbol in the corresponding codeword c is denoted by c_i $(0 \leq i < n)$. Using evaluation mapping encoding, $c_i = f(\alpha_i)$, where $\alpha_0, \alpha_1, \cdots, \alpha_{n-1}$ are distinct nonzero elements of $GF(2^q)$ in fixed order. Considering this encoding, the message polynomial can be recovered by interpolating over the points $(\alpha_0, c_0), (\alpha_1, c_1), \cdots, (\alpha_{n-1}, c_{n-1})$. However, due to the noise from the channel, given the observation of a received symbol, the transmitted symbol can be any finite field element. Hence, the interpolation points for the ith code position may include (α_i, ω_j) for any $\omega_j \in GF(2^q)$. In order to increase the probability that the correct message polynomial can be recovered, ASD algorithms put higher weight on those more reliable points during the interpolation.

ASD algorithms have three steps: multiplicity assignment, interpolation and factorization. The multiplicity assignment decides the interpolation points and their multiplicities using the reliability information from the channel. ASD algorithms are different in the multiplicity assignment. However, they share the same interpolation and factorization steps. The interpolation step finds a bivariate polynomial $Q(x, y)$ with minimum $(1, k-1)$ weighted degree that passes each interpolation point with its associated multiplicity. The (w_x, w_y) weighted degree of a bivariate polynomial $Q(x, y) = \sum_{i=0}^{\infty}\sum_{j=0}^{\infty} q_{i,j}x^i y^j$ is defined as the maximum of $iw_x + jw_y$ such that $q_{i,j} \neq 0$. In addition, a bivariate polynomial $Q(x, y)$ is said to pass a point (α, β) with multiplicity m if $Q(x + \alpha, y + \beta)$ contains a monomial $x^a y^b$ with degree $a + b = m$, but does not contain any monomial with degree less than m. Then the factorization step finds the factors of $Q(x, y)$ in the format of $y - f(x)$ with the degree of $f(x)$ less than k. The computed $f(x)$ form a list of possible message polynomials.

Simple multiplicity assignment schemes are necessary for practical applications. In the KV algorithm, the multiplicity assignment can be implemented by constant multiplications followed by the floor function, which does not require any computation in hardware. Assume that the maximum multiplicity is m_{max}. The multiplicity of an interpolation point in the KV algorithm can range from 1 to m_{max}. The BGMD multiplicity assignment can be implemented by comparators. The ith code position can have a point (α_i, β_i) with multiplicity m_{max}, two interpolation points (α_i, β_i) and (α_i, β_i') with multiplicity $m_{max}/2$, or no interpolation point, depending on the number of bits in the ith received symbol with reliability lower than a threshold. Here β_i is the hard-decision symbol and β_i' is the second most likely symbol for the ith code position. In addition, multiple decoding iterations with different thresholds can be carried out in the BGMD algorithm to correct more errors. The LCC algorithm carries out decoding on 2^η $(\eta \in Z^+)$ test vectors, each of which consists of n interpolation points with multiplicity one. Although the multiplicity of each point is the same, the reliability information from the channel is incorporated into the decision of the interpolation points. For each of the η most unreliable code positions, there can be two interpolation points (α_i, β_i) and (α_i, β_i'). For each of the rest code positions, only one interpolation point (α_i, β_i) is assigned. Then the test vectors are formed by taking one interpolation point for each code position. Usually η is not large. Hence simple parallel comparators are required to sort out the η most unreliable code positions.

The multiplicity assignment in the Chase-GMD algorithm [7] can be also implemented easily. However, this algorithm expands each test vector in the LCC algorithm to $(n - k - \eta)/2 + 1$ vectors by carrying out GMD decoding on the next $(n - k - \eta)$ least reliable code positions. In [8], error-and-erasure decoding is carried out for every possible erasure pattern from a large number of unreliable code positions. This scheme leads to not only much more test vectors, but also irregular erasure patterns, which make it harder to share computations among the decoding for different vectors. Accordingly, the algorithms in [7], [8] are not further considered in this paper.

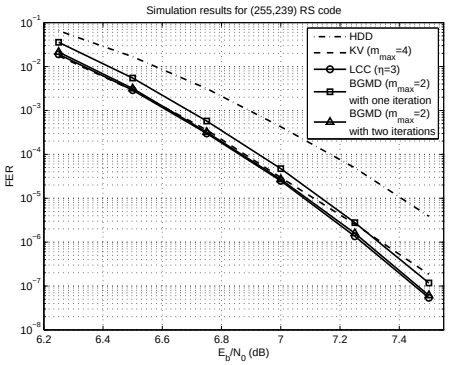

Fig. 1. FERs of ASD algorithms for a (255, 239) RS code over AWGN channel

The multiplicity assignment scheme not only decides the error-correcting performance of the ASD algorithm, but also affects the complexity of the interpolation and factorization steps. The number of computations required in the interpolation is proportional to m_{max}^5. Hence, small multiplicities are necessary to keep the complexity of the interpolation at practical level. Usually $m_{max} \leq 4$ is considered. On the other hand, smaller multiplicities do not always lead to inferior error-correcting performance. Fig. 1 shows the frame error rates (FERs) of ASD algorithms for a (255, 239) RS code constructed over $GF(2^8)$ under the additive white Gaussian noise (AWGN) channel. For the purpose of comparison, the FER of HDD is also included. It can be observed from this figure that the BGMD algorithm with $m_{max} = 2$ and two decoding iterations can achieve similar or higher coding gain than the KV algorithm with $m_{max} = 4$. Although the LCC algorithm has multiplicity one, with $\eta = 3$, it can also achieve similar or better performance than the KV algorithm with $m_{max} = 4$. In addition, the performance of the LCC algorithm will further improve with larger η.

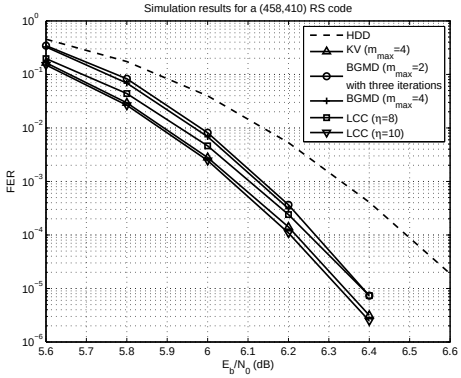

Fig. 2. FERs of ASD algorithms for a (458, 410) RS code over AWGN channel

When the code is longer, the BGMD algorithm with $m_{max} = 2$ can not achieve as good performance as the KV algorithm with $m_{max} = 4$. This can be observed from

Fig. 2, which shows simulation results for a RS (458, 410) code constructed over $GF(2^{10})$ under the AWGN channel. It has also been derived from our simulations that the extra coding gain can be achieved by the BGMD algorithm becomes negligible if more than three decoding iterations are carried out. In addition, in terms of hardware complexity, the BGMD algorithm with $m_{max} = 4$ does not have any advantage over the KV algorithm with the same maximum multiplicity. Hence, the BGMD algorithm with $m_{max} \geq 4$ is not further pursued in this paper. The $n - k$ for the (458, 410) code is also larger than that for the (255, 239) code. Accordingly, larger η needs to be used in the LCC decoding to improve the performance. Fig. 2 shows that $\eta = 10$ is required in order to achieve similar performance as the KV algorithm with $m_{max} = 4$.

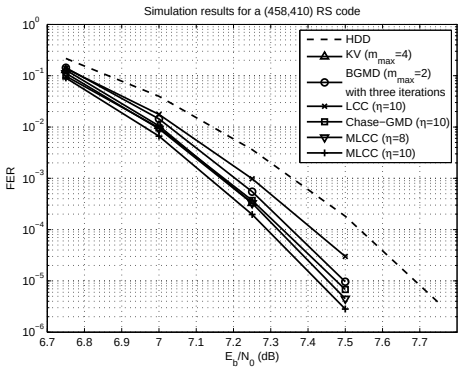

Fig. 3. FERs of ASD algorithms for a (458, 410) RS code on longitudinal magnetic recording channel equalized to the EPR4 target with 100% AWGN.

One major application of RS codes is magnetic recording, which requires RS codes with codeword length 4Kbits or longer. For this application, the performance of the LCC algorithm is degraded by the inter-symbol interference (ISI) in the channel. Fig. 3 shows the FERs of ASD algorithms for a (458, 410) code over $GF(2^{10})$ on longitudinal magnetic recording channel equalized to the EPR4 target with 100% AWGN. It can be seen that the relative performance of the LCC algorithm is worse than that for the AWGN channel. To improve the performance of the LCC algorithm in applications with ISI, a modified LCC (MLCC) algorithm is proposed in [14]. Similar to the Chase-GMD algorithm in [7], erasures are included in the test vectors. However, in the MLCC algorithm, erasures are assigned to the code positions that have more than one bit with reliability lower than a threshold. Then the η most unreliable non-erasure positions are picked and the test vectors are formed in the same way as in the LCC algorithm. Since erasures are assigned to the same code positions in all test vectors, the complexity of the MLCC decoding will not exceed that of the LCC decoding with the same η. Actually, the MLCC decoder can be implemented by the same architecture as the LCC decoder, except that less points need to be interpolated over for each vector. Hence, the complexity of the MLCC decoder is not discussed separately in this paper. The FER curves of the MLCC decoding are also included in Fig. 3. It can be observed that the MLCC decoding can achieve much better performance than the LCC decoding with the same η.

III. Complexity-reducing techniques

The interpolation problem can be solved by the Kötter's [15], [16] and Lee-O'Sullivan [17] algorithms. Although the Lee-O'Sullivan algorithm can potentially lead to higher efficiency when $m_{max} \leq 2$ [18], it does not allow interpolation points or their multiplicities to be changed after the interpolation started. This feature prohibits sharing intermediate interpolation results in the iterative BGMD and LCC decoding. Hence, this paper focuses on the interpolator implementation using the Kötter's algorithm. This algorithm first initializes a set of $t + 1$ polynomials: $Q^{(0)}(x, y) = 1, Q^{(1)}(x, y) = y, \cdots, Q^{(t)}(x, y) = y^t$, where $t = m_{max}$ for high-rate codes. Then for an interpolation point (α, β) with multiplicity m, one coefficient $x^a y^b$ with $a + b < m$ in $Q^{(l)}(x + \alpha, y + \beta)$ $(0 \leq l \leq t)$ is forced to zero in each iteration with minimum increase in the weighted degree. Each iteration consists of discrepancy coefficients computation and polynomial updating based on the computed coefficients. Points can be added and multiplicities can be increased by carrying out more iterations of this algorithm. Hence, the Kötter's interpolation algorithm is referred to as a 'forward' interpolation algorithm in this paper.

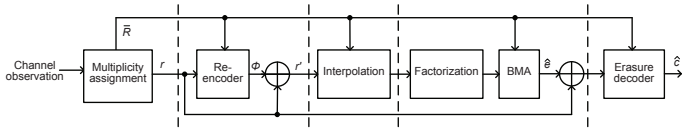

Fig. 4. Block diagram of ASD decoder

The interpolation complexity can be reduced by the re-encoding and coordinate transformation techniques [19], [20]. Assume that the received word is r, and denote the k most reliable code positions in r by the set R. The re-encoding is to find a codeword ϕ that has the same symbols as r in the code positions in R. This can be done by erasure decoding. Since $r' = \phi + r$ is a word corrupted by the same error vector as r, the decoding can be carried out on r' instead. In r', the symbols in the positions in R are zero, and hence the β coordinates of the corresponding interpolation points are zero. As a result, the interpolation over these points can be pre-solved by simple univariate interpolation. In addition, the polynomials resulted from the univariate pre-interpolation can be factored out using a coordinate transformation. After that, the polynomials can be initialized the same as before, but (1,-1) weighted degree should be used in the interpolation. A coordinate transformation and re-encoding architecture based on the BMA can be found in [21]. By reformulating the involved equations, a more efficient re-encoder architecture was proposed in [22]. The factorization step can be carried out on the re-encoded and transformed interpolation output directly. In this case, the factorization output can be used as syndromes in HDD, such as the BMA, to recover the errors in R. It is expected that the number of errors in those reliable code positions is small. Hence, the number of syndromes need to be computed is much less than k, the number of coefficients for $f(x)$ need to be originally computed from the factorization. After the errors in R are corrected, an erasure decoding can

be applied to recover the entire codeword. Therefore, ASD decoding can be carried out according to the block diagram in Fig. 4.

In the Kötter's interpolation, a point of multiplicity m requires $m(m+1)/2$ interpolation iterations and m_{max} polynomials are involved for high-rate codes. In addition, the degrees of the involved polynomials increase with iteration number. For a given number of processing units, polynomials of higher degree lead to longer latency in each iteration. Therefore, the interpolation complexity of the KV algorithm with $m_{max} = 4$ is significantly higher than that of the BGMD algorithm with $m_{max} = 2$ even if three iterations need to be carried out. It is also higher than that of the LCC algorithm with moderate η. In [23], power representation is used for finite field elements, such that carry-save adders can replace finite field multipliers in the feedback loops for discrepancy coefficients computation to increase the maximum achievable clock frequency. The clock frequency can be also increased by applying the slow-down technique [24]. However, these techniques do not change the relative interpolation complexity of the three ASD algorithms. Although some discrepancy coefficients are zero and the corresponding polynomial updating can be skipped [25], this property is very hard to be incorporated to simplify the hardware due to the complicated control.

By making use of the properties of the points and their multiplicities, further simplifications can be made to the interpolation in the BGMD and LCC algorithms. The test vectors in the LCC algorithm can be arranged in an order such that adjacent vectors only have one pair of different points (α, β) and (α, β'). Given the interpolation result of the current vector, the point (α, β) can be eliminated in one iteration by using the backward interpolation scheme proposed in [26], regardless of the location of the point in the vector. Then (α, β') can be added using one iteration of the Kötter's interpolation. The backward and Kötter's forward interpolation can be also carried out by a unified architecture in one iteration [27]. Hence, once the interpolation result of the first test vector is available, the interpolation for each of the other vectors can be done in one single iteration. These techniques greatly reduced the number of iterations in the LCC interpolation, especially when η is not small. The backward interpolation was also extended for the BGMD algorithm with $m_{max} = 2$ to enable the sharing of interpolation results in adjacent decoding iterations [28]. Further speedup of the BGMD interpolation can be achieved by computing the discrepancy coefficients from multiple iterations over the same point in a look-ahead manner, and then combining the polynomial updating in those iterations [29]. In addition, this combined interpolation can be incorporated with the backward interpolation to achieve even higher speed [30]. In the LCC decoding, the interpolation points in a vector all have different α. In the KV algorithm, the look-ahead discrepancy coefficients computation needs to take care of much more possibilities, since the multiplicities are larger. Therefore, applying the combined interpolation technique to the LCC or KV decoding would lead to large area overhead.

The factorization problem can be solved by the iterative algorithm proposed by Roth and Ruckenstein [31]. Each iteration of this algorithm involves root computations over finite fields for a polynomial, whose degree can be as large as t. When the degree of the polynomial is higher than one, root computations over finite fields are traditionally carried out by exhaustive search. The prediction-based schemes in [32], [33] can be employed to circumvent the exhaustive search in most cases. If the prediction is correct, only one field multiplication and constant binary matrix multiplications are required to find the root. When the degree is at most four, the polynomial can be also converted to an affine format, in which the roots can be computed directly [34]. Particularly, when the degree of the polynomial is two, the conversion to the affine format and root computation are much less complicated. In addition, it was discovered in [35] that the factorization can be actually skipped when $t = 1$, which is the case in the LCC decoding for high-rate codes. When $t = 1$, the interpolation output can be written as $Q(x, y) = q_0(x) + q_1(x)y$. In this case, $q_1(x)$ can be used as the error locator polynomial to compute the errors in the positions in R. Hence, the factorization and the key equation solver (KES) step in the following BMA can be eliminated. In the LCC decoding, it would require very high complexity if the rest decoding steps are carried out on the interpolation result for each test vector. A polynomial selection scheme is proposed in [36] to pass only one interpolation output to later decoding steps through testing whether the degree of $q_1(x)$ equals its root number. From simulations, this scheme does not result in any performance degradation, except for high-rate short codes.

$m_{max} \geq 4$ is usually required in the KV algorithm to achieve good performance, while smaller m_{max} can be used in the BGMD algorithm and $m_{max} = 1$ in the LCC algorithm. In addition, the backward, combined, and/or unified architectures can be applied to the BGMD algorithm with $m_{max} = 2$ and LCC algorithm to further simplify the interpolation. Accordingly, the interpolation in the KV algorithm typically has much higher complexity than those in multi-iteration BGMD decoding with smaller multiplicity and LCC decoding with moderate η. The KV algorithm also has more involved factorization step since $m_{max} \geq 4$ and the roots need to be calculated through complex schemes. Although the BGMD decoding may have simpler interpolation than the LCC algorithm with moderate η, it requires factorization, which needs to be carried out in each decoding iteration. Therefore, the LCC algorithm with moderate η has the lowest complexity among the three ASD algorithms.

IV. Complexity analyses and comparisons

Pipelining can be applied to ASD decoders according to the dashed lines in Fig. 4 to achieve higher speed. Among the pipelining stages, the stage for the interpolation has longer latency, and hence decides the throughput of the pipelined decoder. The average interpolation latency changes with not only the parameters of the RS code, but also channel condition.

In addition, the decoder blocks are scalable. Proper parallel processing factors can be chosen such that each pipelining stage has about the same latency in order to increase the hardware utilization efficiency. Considering these issues, it is very difficult to derive a single formula to express the hardware complexity of ASD decoders. In this section, the hardware complexities of the three ASD decoders are compared using an example (458, 410) RS code constructed over $GF(2^{10})$. Then how the decoder complexities change with different parameters are discussed in the next section. In the comparison, we consider the KV algorithm with $m_{max} = 4$, BGMD algorithm with $m_{max} = 2$ and three decoding iterations, and LCC algorithm with $\eta = 10$. Using larger m_{max} and η can improve the performance of the KV and LCC algorithms, respectively. We are interested in the relative complexities of the KV and LCC decoders that can achieve similar error-correcting performance. From Fig. 2 and 3, the LCC or MLCC decoder with $\eta = 10$ can achieve similar or higher coding gain than the KV algorithm with $m_{max} = 4$ for the (458, 410) code. As mentioned previously, the MLCC and LCC decoders are only different in the multiplicity assignment step, and the rest decoding steps can be implemented by the same architectures. Therefore, the LCC decoder with $\eta = 10$ is considered in the comparison. $m_{max} = 2$ is picked for the BGMD decoding because larger m_{max} will make this algorithm lose its advantage in terms of hardware complexity. In addition, increasing the decoding iteration number beyond three would only lead to negligible additional coding gain. It should be noted that this BGMD decoder can not achieve as good performance as the KV and LCC decoders for the RS (458, 410) code.

A. Complexity of the KV decoder

The re-encoder in Fig. 4 consists of an erasure decoder and hardware for coordinate transformation. It can be implemented by the scalable architecture in [22], which computes the erasure locator and evaluator directly by polynomial multiplications. In addition, the involved equations are reformulated to further simplify the computations. Compared to the re-encoder in [21], this architecture can achieve much higher speed with smaller area. With minor modifications, this architecture can be also used to implement the erasure decoder at the end of ASD decoding. The interested reader is referred to [22] for the details of the re-encoder.

Using the Kötter's interpolation, a point with multiplicity m requires $m(m+1)/2$ iterations. Although the maximum y-degree of the polynomials does not change, the maximum x-degree of the polynomials, denoted by d_x, increases with interpolation iterations. If the coefficients of a polynomial are updated serially, the latency of an iteration equals the sum of $d_x + 1$ and the number of pipelining stages in the interpolation architecture. Hence, the number of clock cycles required for the KV interpolation can be very large. It can easily exceed 30000 for the (458, 410) code with $m_{max} = 4$. When m_{max} and hence t is larger than two, interpolation iterations can not be combined without incurring large area overhead. Instead, the involved polynomials can be broken down into multiple pieces, and parallel processing can be applied to have these pieces processed simultaneously. From simulations, the average numbers of clock cycles, n_{kv}, required for the interpolation in the KV decoder with $m_{max} = 4$ for the (458, 410) code under the AWGN channel using 4-parallel processing are derived as listed in Table I. When $E_b/N_0 = 6.4dB$, 7859 clock cycles are required. This number will increase significantly with E_b/N_0.

TABLE I
INTERPOLATION LATENCY OF THE KV DECODER WITH $m_{max} = 4$ AND 4-PARALLEL PROCESSING FOR THE (458, 410) CODE UNDER AWGN CHANNEL

E_b/N_0	5.6dB	5.8dB	6dB	6.2dB	6.4dB
n_{kv}	3583	4587	5660	6773	7859

In the prediction-based root computation schemes for the factorization [32], [33], exhaustive search still needs to be carried out when prediction fails. To reduce the worst-case latency, the factorization architecture in [34] based on direct root computations can be used when $m_{max} = 4$. The BMA can be implemented using the architecture in [37]. This architecture is also scalable.

Table II shows the hardware requirement and latency of the blocks in the KV decoder with $m_{max} = 4$ for the RS (458, 410) code. In this table, a Mux refers to a 1-bit 2:1 multiplexor. Compared to a regular multiplier, a constant multiplier not only needs less gates, but also has shorter critical path. The matrix reduction required for the factorization is used for finding the roots of affine polynomials. It can be implemented by around 1000 logic gates [34]. To correct τ errors in R using the BMA, 2τ syndromes need to computed from the factorization. Simulations show that setting τ to 32 only leads to negligible performance degradation. Proper parallel processing is applied to the decoder components to balance the number of clock cycles required in each pipelining stage. A fully-folded re-encoder is employed since its latency is still shorter than that of the interpolation step. In the case of $m_{max} = 4$, the factorization step can output at most four syndrome vectors. Hence, the BMA decoder needs to run four times in the worst case. Accordingly, 10-folding is applied to the KES architecture in [37], such that the total latency of the factorization and BMA is about the same as that of the interpolation. Similarly, the erasure decoder needs to run at most four times. It is scaled to finish four decoding in about the same time as the interpolation with minimum area. RAMs instead of registers are used for pipelining to reduce the area. It can be observed from Table II that the decoding of a word using the pipelined KV decoder requires 7859 clock cycles. The critical path of the KV decoder lies in the matrix reduction part of the factorization architecture. It consists of 6 AND gates, 5 OR gates, 1 NOT gate, and 1 Mux.

B. Complexity of the LCC decoder

In the LCC decoding, the factorization and the KES in the BMA can be eliminated. Hence, the LCC decoder can

TABLE II
HARDWARE REQUIREMENT OF THE KV DECODER WITH $m_{max} = 4$ FOR A (458, 410) CODE

	$GF(2^{10})$ Multiplier	$GF(2^{10})$ Adder	$GF(2^{10})$ Inverter	Constant Multiplier	Mux	Matrix Reduction	RAM (bit)	Register (bit)	Latency (# of clock cycles)
Re-encoder	5	54	1	48	520	0	960	1540	6912
Interpolation	421	315	0	0	2600	0	49000	1610	7859
Factorization	25	35	9	6	530	3	12800	1570	3301
BMA	22	122	1	112	1320	0	2560	3170	1108×4
Erasure Decoder	16	62	2	48	1080	0	480	1630	1728×4
Pipelining RAM	0	0	0	0	0	0	24080	0	-
KV Decoder	489	588	13	214	6050	3	89880	9520	7859

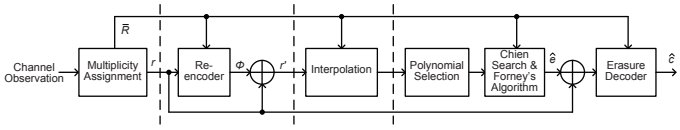

Fig. 5. Block diagram of the re-encoded LCC decoder

be simplified as in Fig. 5. The Chien search and Forney's algorithm in this figure are the two remaining steps of the BMA.

TABLE IV
LATENCY STATISTICS OF THE LCC INTERPOLATION WITH $\eta = 10$ OVER AWGN CHANNEL FOR A (458, 410) RS CODE

E_b/N_0	5.6dB	5.8dB	6dB	6.2dB	6.4dB
d_{n-k}	13	14	16	18	19
n_{lcc}	30531	30579	30675	30771	30819

The interpolation over the first test vector in the LCC decoding requires $n-k$ iterations using the Kötter's algorithm. Then the interpolation for each additional test vector can be done in one unified backward-forward interpolation iteration [27]. Assume that the interpolation output for a test vector is written as $Q(x, y) = q_1(x)y+q_0(x)$. The polynomial selection scheme in [36] searches for roots of $q_1(x)$ over all the n nonzero finite field elements used in the evaluation mapping encoding. The first interpolation output with the degree of $q_1(x)$ equals its root number is passed to the following decoding steps. Although the interpolation over the rest test vectors can be skipped, the other decoder blocks can not be adjusted dynamically to meet the latency of the interpolation. Hence the worse case latency of interpolating over all vectors is considered when adjusting the parallel processing factors of other decoder blocks.

The average d_x in the $n - k$ Kötter's forward interpolation iterations for the first test vector changes with channel conditions. This average d_x is denoted by d_{n-k}, and the corresponding values for the (458, 410) code derived from simulations over the AWGN channel are listed in Table IV. If the interpolation output of the first test vector satisfies the polynomial selection criterion, then the interpolation stops. Otherwise, the unified backward-forward interpolation will be carried out until an interpolation result passes the polynomial selection. From simulations, the d_x in the unified backward-forward interpolation iterations need to be carried out is 24 a majority of the time for the (458, 410) code, and can be 25 at most. An explanation for this is that the d_x at the end of the $(n - k)th$ interpolation iteration for the first test vector will be higher if there are less errors. This is because during each Kötter's interpolation iteration for a point (α, β), the polynomial with nonzero discrepancy coefficient and minimum weighed degree is multiplied by $(x-\alpha)$, while the degrees of other polynomials do not change. If there are less errors, then the factor $(x - \alpha)$ tends to be multiplied to the same polynomial in more iterations. Also the first test vector in our design is always the hard-decision vector, the most reliable one. If d_x is larger than $(n - k)/2 = 24$ in the interpolation result for the first test vector, this vector will most likely correspond to the correct codeword and pass the polynomial selection. Accordingly, the unified backward-forward interpolation does not need to be done at all. On the other hand, if the backward-forward interpolation needs to be carried out, the d_x remains at 24 most of the time if the corresponding test vector has more than $(n - k)/2$ errors. In the worst case, all 2^η vectors need to be tested. Hence, the number of clock cycles required for the interpolation in the LCC decoding can be computed as $n_{lcc} = (n - k) \times (d_{n-k} + 1+\xi_{lcc})+(2^\eta-1)\times((n-k)/2+1+\xi_{lcc})$, where ξ_{lcc} denotes the pipelining stage number in the interpolation architecture. From the architecture in [27], $\xi_{lcc} = 4$. The n_{lcc} for different E_b/N_0 are listed in Table IV. When $E_b/N_0 = 6.4dB$, 30819 clock cycles are required in the LCC interpolation with $\eta = 10$. This number will increase slightly when E_b/N_0 is higher.

Since the interpolation step takes much longer time than other blocks in the LCC decoder, it would lead to low hardware utilization efficiency. Fortunately, parallel processing can be easily applied to the LCC interpolation. The 2^η test vectors can be divided into multiple groups. Inside each group, the test vectors can still be ordered such that there is only one pair of different points in adjacent vectors. Accordingly, multiple interpolators can be employed to carry out the interpolation for different groups of test vectors simultaneously. Dividing the test vectors into p_{lcc} groups, the interpolation can be finished in $(n-k)\times(d_{n-k}+1+\xi_{lcc})+(\lceil 2^\eta/p_{lcc}\rceil-1)\times((n-k)/2+1+\xi_{lcc})$ clock cycles by p_{lcc} interpolators.

To avoid data build-up, the polynomial selection on the current interpolation output should be completed before the next interpolation output is generated. The computation of an interpolation output takes at least $(n - k)/2 + 1 + \xi_{lcc} = 29$ clock cycles. Hence, the root computation for $q_1(x)$ over n

TABLE III
HARDWARE REQUIREMENT OF THE LCC DECODER WITH $\eta = 10$ FOR A RS (458, 410) CODE

	$GF(2^{10})$ Multiplier	$GF(2^{10})$ Adder	$GF(2^{10})$ Inverter	Constant Multiplier	Mux	RAM (bit)	Register (bit)	Latency (# of clock cycles)
Re-encoder	7	57	1	48	540	960	1550	4608
Interpolation	21×8	19×8	1×8	0	150×8	1960×8	270×8	4835
Polynomial Selection	0	384×8	0	384×8	240×8	0	450×8	29
Chien Search	0	24	0	24	240	0	240	460
Forney's Algorithm	2	72	1	72	720	0	940	8
Erasure Decoder	9	61	1	48	580	480	1590	4032
Pipelining RAM	0	0	0	0	0	16340	0	-
LCC Decoder	186	3438	11	3264	5200	33460	10080	4864

finite field elements, which is done by exhaustive-search-based Chien search, needs to be finished in the same amount of time. Accordingly, $\lceil 458/29 \rceil = 16$-parallel Chien search is required for the root computation in the polynomial selection. The parallel Chien search can be implemented by the architecture in [38]. In addition, p_{lcc} parallel Chien search engines need to employed to simultaneously process the outputs of p_{lcc} interpolators.

The error locations in R, which are the roots of $q_1(x)$, have already been found during the polynomial selection. However, parallel Chien search is employed for each interpolation output, and multiple roots can be found in the same clock cycle. Recording these roots either requires large memory or complicated control logic that can not be finished in one clock cycle. Therefore, in the polynomial selection, only the root numbers are counted, and the roots are not stored. Once the single interpolation output is chosen by the polynomial selection, a serial Chien search is carried out on the corresponding $q_1(x)$ again to find the roots. After the error locations are found, error magnitudes can be calculated by an equation similar to that of the Forney's algorithm.

The hardware requirement and latency of the building blocks in the LCC decoder are listed in Table III. Eight interpolators are employed in our design. Accordingly, 4835 clock cycles are required for the interpolation. Pipelining can be applied to the LCC decoder according to the cutsets shown by the dashed lines in Fig. 5. In addition, proper parallel processing is applied to other decoder components to best match the number of clock cycles required in each pipelining stage. From Table III and Fig. 5, it can be derived that the LCC decoding with $\eta = 10$ for a (458, 410) RS word can be completed in 4835 clock cycles. In addition, the critical path of the LCC decoder has one finite field multiplier, one finite field adder and one Mux.

C. Complexity of the BGMD decoder

TABLE V
INTERPOLATION LATENCY OF THE BGMD DECODER WITH $m_{max} = 2$ AND THREE ITERATIONS FOR A (458,410) CODE OVER AWGN CHANNEL

E_b/N_0	5.6dB	5.8dB	6dB	6.2dB	6.4dB
n_{bgmd}	3117	3370	3694	3927	4034

As mentioned previously, the BGMD decoder can achieve the best performance-complexity tradeoff when $m_{max} = 2$ and three decoding iterations are carried out. The combined-backward interpolation architecture [30] is employed for the BGMD decoder because it can achieve the highest efficiency. In this architecture, the three interpolation iterations for a point with $m = 2$ are combined into one iteration. In addition, the interpolation iterations over two points with $m = 1$ and the same α coordinate are also combined. Accordingly, the interpolation in the first decoding iteration can be finished in at most $n - k$ iterations. Besides, the backward interpolation is applied, so that only the different points need to be taken care of in the interpolation of the second and later decoding iterations. As a result, the number of clock cycles required by the interpolation in the iterative BGMD decoding is affected by not only the interpolation points and their multiplicities, but also how they are different from those in later decoding iterations. The average numbers of clock cycles required for the interpolation, n_{bgmd}, in the BGMD decoding with $m_{max} = 2$ and three decoding iterations for a (458, 410) RS code over the AWGN channel are listed in Table V. It can be observed that 4034 clock cycles are required when $E_b/N_0 = 6.4dB$.

Table VI lists the hardware requirement and latency of the BGMD decoder with $m_{max} = 2$ and three decoding iterations. This decoder is also pipelined according to the dashed lines in Fig. 4. Converting a polynomial with degree two to affine format can be done easily. Hence the factorization architecture in this BGMD decoder has shorter latency and smaller area than that in the KV decoder with $m_{max} = 4$. However, the factorization needs to be carried out three times for the three decoding iterations. The only parallel factorization architecture was proposed in [39]. This architecture can achieve less than two times speed with much more than two times area. Hence, instead of resorting to parallel processing inside the factorization engine, three factorization engines are employed. Since $m_{max} = 2$, each factorization engine generates at most two syndrome vectors. To process these syndromes in time, three properly-scaled BMA decoders are employed and each of them takes care of two syndrome vectors. The erasure decoder occupies a pipelining stage itself. One erasure decoder is used, and parallel processing has been adopted such that it can decode six vectors in about the same time as the computations in other pipelining stages. The critical path of the BGMD decoder has one finite field multiplier, two finite field adders and three Muxes.

TABLE VI
HARDWARE REQUIREMENT OF THE BGMD DECODER WITH $m_{max} = 2$ AND THREE DECODING ITERATIONS FOR A (458, 410) CODE

	$GF(2^{10})$ Multiplier	$GF(2^{10})$ Adder	$GF(2^{10})$ Inverter	Constant Multiplier	Mux	RAM (bit)	Register (bit)	Latency (# of clock cycles)
Re-encoder	9	61	1	48	580	960	1590	4032
Interpolation	41	51	1	0	1440	16740	1240	4034
Factorization	9×3	5×3	3×3	0	140×3	3720×3	270×3	2338
BMA	42×3	132×3	1×3	112×3	1520×3	1280×3	3120×3	778×2
Erasure Decoder	64	107	5	48	1300	480	2040	650×6
Pipelining RAM	0	0	0	0	0	22160	0	-
BGMD Decoder	267	630	19	432	8300	55340	15040	4034

TABLE VII
COMPARISONS OF ASD DECODERS FOR A RS (458, 410) CODE

	KV ($m_{max} = 4$)	BGMD ($m_{max} = 2$ 3 iterations)	LCC ($\eta = 10$)	LCC ($\eta = 8$)
Area (# of XOR gates)	226625	174465	188410	78521
Critical Path (# of gates)	13	11	8	8
Latency (# of clks)	7859	4034	4835	4835
Throughput (normalized)	1	2.30	2.64	2.64
Efficiency (normalized)	1	2.99	3.18	7.62

D. Comparisons

A $GF(2^{10})$ multiplier can be implemented by 101 XOR gates and 100 AND gates with 5 XOR gates and 1 AND gate in the critical path [34]. In addition, a $GF(2^{10})$ inverter using composite field arithmetic needs 164 XOR gates, 160 AND gates, 36 OR gates and 5 NOT gates. The gate number of a constant multiplier depends on the constant multiplicand. Each AND or OR gate requires $3/4$ the area of an XOR, while each NOT gate requires $1/4$ the area of an XOR. Moreover, each Mux or memory cell has the same area as an XOR. In addition, each register occupies about three times the area of an XOR. These assumptions are used in estimating the total area requirement of the ASD decoders for a (458, 410) RS code listed in Table VII.

Since the decoder architectures are scalable, higher throughput can be achieved by using larger area. Hence the efficiency in terms of speed-over-area ratio is a better criterion to compare different decoders. As it can be observed from Table VII, the LCC decoder with $\eta = 10$ is 218% more efficient than the KV decoder with $m_{max} = 4$. It is also slightly more efficient than the BGMD decoder with $m_{max} = 2$ and three decoding iterations, which can not even achieve the same performance as the LCC decoder with $\eta = 8$ over the AWGN channel. When $\eta = 8$, the interpolation latency of the LCC decoding can be reduced to around 25%. Accordingly, only two interpolators and parallel Chien search engines are required to achieve the same decoding latency. In this case, the gate count of the LCC decoder can be reduced significantly as shown in Table VII. It can be calculated that the LCC decoder with $\eta = 8$ can achieve 155% higher efficiency than the BGMD decoder. In addition, the MLCC decoder with $\eta = 8$, which has the same hardware complexity as the LCC decoder with $\eta = 8$, can achieve similar performance as the KV decoder and better performance than the BGMD decoder in magnetic recording channel. Therefore, the same error-correcting performance can be achieved by the LCC or MLCC decoder with much lower hardware complexity. It should also be considered that one codeword needs be picked from the list generated by the BGMD and KV decoders. This selection is usually based on either the reliabilities of the codewords or Euclidean distances between the codewords and the hard-decision received word. Both schemes would lead to performance loss, which has not been incorporated in the simulation curves in Fig. 1, 2 or 3. Moreover, the additional hardware requirement for the selection has not been included in the KV or BGMD decoder complexity shown in Table II and VI.

V. DISCUSSION

As it can be observed from the ASD decoder designs, the interpolation step has longer latency than other steps. Hence, pipelining is applied according to the dashed lines in Fig. 4 or 5, and the interpolation latency decides the throughput of the overall decoder. When re-encoding and coordinate transformation are applied, the interpolation latency is not affected by the codeword length n directly. Instead, it is affected by $n - k$. In the KV decoder, the interpolation iteration number changes almost proportionally with $n - k$. In addition, $n - k$ plays a significant role in deciding the BGMD interpolation latency. However, in the LCC decoding, $n - k$ only affects the interpolation iteration number for the first test vector. Hence the effect of $n - k$ on the interpolation latency of the LCC decoding seems less significant. However, for codes with higher rates, which translate to smaller $n-k$, the LCC decoder can achieve similar error-correcting performance as the KV and BGMD decoders with a smaller η. Our observations show that the required η is almost proportional to $n - k$. From Fig. 1 for the (255, 239) code, when $n - k = 16$, the LCC decoder with $\eta = 3$ can achieve similar performance as the KV decoder with $m_{max} = 4$. It has also been observed from our simulations that for a (440, 410) code constructed over $GF(2^{10})$ with $n - k = 30$, the LCC decoder with $\eta = 6$ can match the performance of the KV decoder. The LCC interpolation iteration number reduces almost proportionally with 2^{η} when η is not small. To achieve the same latency, the number of interpolators and polynomial selection engines can be reduced by a factor of two for each reduction by one in η. It can be calculated from Table III that a copy of the interpolator and polynomial selection engine accounts for around 10% of the overall area of the LCC decoder for the (458, 410) code. Hence, when η is less, the area requirement of the LCC decoder can be reduced substantially. Therefore, the efficiency of the LCC decoder over the other two ASD decoders will

increase significantly when $n-k$ becomes smaller. If $n-k$, and hence η further decreases, a single copy of the interpolator can be used in the LCC decoding. In this case, the interpolation latency over the second and later vectors will be reduced by half for each reduction by one in η. Although other decoder blocks need to employ higher parallel processing factors to match the faster interpolation, they only lead to small increase in the decoder area. This is because that the memory and registers hardly change with parallel processing factors and they accounts for a large proportion of the area of other decoder blocks. Accordingly, the LCC decoder can achieve even higher efficiency if η further decreases.

In ASD decoding, the interpolation points and/or their multiplicities change with E_b/N_0. Accordingly, the interpolation latency changes. The decoder should be designed taking into account the worst-case average interpolation latency for the range of E_b/N_0 required by the application. When E_b/N_0 increases, the interpolation latency of the LCC decoder increases by multiples of $n-k$ as shown in the last row of Table IV. This is because E_b/N_0 only changes the average d_x in the forward interpolation for the first test vector, which takes $n-k$ iterations. The average d_x can be at most $(n-k)/2$ in these iterations. Accordingly, the interpolation latency for the (458, 410) code can be increased to at most $(n-k)\times(24+1+\xi_{lcc})+127\times(24+1+\xi_{lcc})=5075$ clock cycles in the LCC decoding with $\eta=10$ using eight interpolators. Compared to the 4835 clock cycles required in the example LCC decoder, it is less than 5% increase. Hence, slightly lower level parallel processing may be applied to other decoder components to reduce the area. However, the overall area requirement does not change much. As a result, the efficiency of the LCC decoder only changes slightly when higher E_b/N_0 is considered. In the KV decoder, the average multiplicity can be very close to m_{max} when E_b/N_0 is high. Since both the interpolation iteration number and average d_x grow with the square of multiplicity, the interpolation latency of the KV decoder can increase significantly. In the case that all points have multiplicity m_{max}, the interpolation has $(n-k)\times(m_{max}\times(m_{max}+1)/2)$ iterations, and the average d_x can be $(n-k)\times m_{max}/2$ in the worst case. Accordingly, the worst-case latency of the KV interpolation with $m_{max}=4$ using 4-parallel processing is 12120 clock cycles, which is more than 50% increase from the latency in the example KV decoder. Similarly, smaller parallel processing factors may be employed in other decoder clocks when the interpolation latency is longer. However, the change in the overall area requirement is much less than the change in the interpolation latency. Therefore, the efficiency of the KV decoder will be much lower when E_b/N_0 is higher. In the first iteration of the BGMD decoding, the number of interpolation iterations equals the number of code positions in $\bar{R}$ with non-trivial interpolation points if the combined interpolation architecture is adopted. This number increases with E_b/N_0, and it can be at most $n-k=48$. In addition, the average d_x in these iterations increases with E_b/N_0. On the other hand, if E_b/N_0 further increases, the numbers of different interpolation points in the second and later decoding iterations will decrease, and thus the number of combined-backward interpolation iterations will decrease. Overall, the interpolation latency in the BGMD decoding first increases with E_b/N_0 as can be observed from Table V. Simulations have also been carried out for $6.4dB < E_b/N_0 < 7.2dB$. The latency keeps increasing slightly with E_b/N_0 in this range. At a certain higher E_b/N_0 when almost each code position has an interpolation point with m_{max} in the first decoding iteration, the interpolation latency may stop increasing or even decrease by a small amount. When the interpolation latency in the BGMD decoding is slightly longer, the area of other decoder blocks may be reduced insignificantly. Hence, the overall efficiency of the BGMD decoder does not change much if higher E_b/N_0 is considered.

The codeword length, n, mainly affects the complexity of the syndrome computations in the re-encoder and erasure decoder, as well as that of the Chien search. If the code is shorter, smaller parallel processing factors can be used for these blocks, or these blocks require less clock cycles so that the other decoder components in the same pipelining stage can be scaled down without reducing the overall decoding speed. It can be observed from Table III that the Chien-search-based polynomial selection can consume a significant part of the decoder area due to the high-speed needs to be achieved. On the other hand, the syndrome computation and serial Chien search in the BMA occupies much less area. Therefore, shorter codeword length will lead to more noticeable area reduction in the LCC decoder than in the KV and BGMD decoders.

In summary, with substantially lower hardware complexity, the LCC or MLCC decoder can achieve similar or better error-correcting performance than the KV and BGMD decoders. This is mainly because that the interpolation multiplicity is always one in the LCC algorithm. Accordingly, the factorization step can be eliminated. In addition, the test vectors can be arranged such that adjacent vectors have only one pair of different points with the same α coordinate. As a result, the backward or unified backward-forward interpolation can be applied with small area overhead. The novel polynomial selection scheme also contributes to the lower complexity of the LCC decoder. Otherwise, the remaining decoding steps need to be carried out for each test vector.

VI. Conclusions

This paper summarized available techniques that can be used to reduce the hardware complexity of ASD decoders. Taking into account applicable techniques, complexity comparisons for three practical ASD decoders were carried out. In addition, this paper provided discussions on how the decoder complexities change with codeword length, code rate, channel condition and η. It was derived that the LCC decoder and its variation, the MLCC decoder, can achieve similar or higher coding gain with lower complexity for high-rate codes.

References

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

[2] F. Parvaresh and A. Vardy, "Multiplicity assignments for algebraic soft-decoding of Reed-Solomon codes," *Proc. Intl. Symp. Info. Theory*, pp. 205, Yokohama, Japan, Jul. 2003.

[3] M. El-Khamy and R. J. McEliece, "Interpolation multiplicity assignment algorithms for algebraic soft-decision decoding of Reed-Solomon codes," *AMS-DIMACS volume on Algebraic Coding Theory and Info. Theory*, vol. 68, 2005.

[4] N. Ratnakar and R. Koetter, "Exponential error bounds for algebraic soft-decision decoding of Reed-Solomon codes," *IEEE Trans. on Info. Theory*, vol. 51 pp. 3899-3917, Nov. 2005.

[5] J. Bellorado and A. Kavcic, "A low-complexity method for Chase-type decoding of Reed-Solomon codes," *Proc. Intl. Symp. Info. Theory*, pp. 2037-2041, Seattle, Washington, Jul. 2006.

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

[7] H. Xia, H. Wang and J. Cruz, "A Chase-GMD algorithm for soft-decision decoding of Reed-Solomon codes on perpendicular recording channels," in *Proc. of IEEE International Conference on Communications*, pp. 1977-1981, Beijing, China, May 2008.

[8] S. Lee and B. Kumar, "Soft-decision decoding of Reed-Solomon codes using successive error-and-erasure decoding," in *Proc. of IEEE Global Telecommunications Conference*, pp. 1-5, New Orleans, LA, Nov. 2008.

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

[10] V. Guruswami and M. Sudan, "Improved decoding of Reed-Solomon and algebraic-geometric codes," *IEEE Trans. on Info. Theory*, vol. 45, pp. 1755-1764, Sep. 1999.

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

[12] T. K. Truong, W. L. Eastman, I. R. Reed and I. S. Hsu, "Simplied procedure for correcting both errors and erasures of Reed-Solomon code using Euclidean algorithm," *IEE Proceedings*, vol. 135(6), Nov. 1988.

[13] G. D. Forney, "Generalized minimim distance decoding," *IEEE Trans. Inform. Theory*, vol. IT-12, pp. 125-131, 1966.

[14] X. Zhang, J. Zhu and W. Zhang, "Modified low-complexity Chase soft-decision decoder of Reed-Solomon codes," submitted to *Springer Journal of Signal Processing Systems*.

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

[16] R. R. Nielson, *List Decoder of Linear Block Codes*, Ph.D thesis, Dept. of Mathematics, Technical University of Denmark, 2001.

[17] K. Lee and M. E. O'Sullivan, "An interpolation algorithm using Gröbner bases for soft-decision decoding of Reed-Solomon codes," *IEEE Intl. Symp. Info. Theory*, Seattle, Washington, Jul. 2006.

[18] J. Zhu and X. Zhang, "Efficient VLSI architecture for soft-decision decoding of Reed-Solomon codes," *IEEE Trans. on Circuits and Systems-I*. vol. 55(10), pp. 3050-3062, Nov. 2008.

[19] W. J. Gross, *et. al.*, "A VLSI architecture for interpolation in soft-decision decoding of Reed-Solomon codes," *Proc. IEEE Workshop on Signal Processing Systems*, pp. 39-44, San Diego, Oct. 2002.

[20] R. Koetter and A. Vardy, "A complexity reducing transformation in algebraic list decoding of Reed-Solomon codes," *Proc. Info. Theory Workshop*, pp. 10-13, Paris, France, Mar. 2003.

[21] J. Ma, A. Vardy and Z. Wang, "Reencoder design for soft-decision decoding of an (255,239) Reed-Solomon code," *Proc. IEEE Intl. Symp. on Circuits and Systems*, pp. 3550-3553, Island of Kos, Greece, May 2006.

[22] J. Zhu and X. Zhang, "High-speed re-encoder design for algebraic soft-decision Reed-Solomon decoding," Proc. of IEEE International Symposium on Circuits and Systems, Paris, France, May 2010.

[23] Z. Wang and J. Ma, "High-speed interpolation architecture for soft-decision decoding of Reed-Solomon codes," *IEEE Trans. on VLSI Systems*, vol. 14, no. (9), pp. 937-950, Sep. 2006.

[24] X. Zhang and J. Zhu, "Efficient interpolation architecture for soft-decision Reed-Solomon decoding by applying slow-down," *Proc. IEEE Workshop on Signal Processing Systems*, Washington D.C., Oct. 2008.

[25] X. Zhang, "Reduced complexity interpolation architecture for soft-decision Reed-Solomon decoding," *IEEE Trans. on VLSI Systems*, vol. 14(10), pp. 1156-1161, Oct. 2006.

[26] J. Zhu and X. Zhang and Z.Wang "Novel interpolation architecture for low-complexity Chase soft-decision decoding of Reed-Solomon codes," *Proc. IEEE Intl. Symp. on Circuits and Systems*, pp. 3078-3081, Seattle, WA, May 2008.

[27] X. Zhang and J. Zhu, "Algebraic soft-decision decoder architectures for long Reed-Solomon codes," submitted to *IEEE Trans. on Circuits and Systems-II*.

[28] J. Zhu, X. Zhang and Z. Wang, "Backward interpolation architecture for algebraic soft-decision Reed-Solomon decoding," *IEEE Trans. on VLSI Systems*, vol. 17 (11), pp. 1602-1615, Nov. 2009.

[29] J. Zhu, X. Zhang and Z. Wang, "Combined interpolation architecture for soft-decision decoding of Reed-Solomon codes," *Proc. IEEE International Conference on Computer Design*, pp. 526-531, Lake Tahoe, CA, Oct. 2008.

[30] X. Zhang and J. Zhu, "High-throughput interpolation architecture for algebraic soft-decision Reed-Solomon decoding," *IEEE Trans. on Circuits and Systems-I*, in press.

[31] R. M. Roth and G. Ruckenstein, "Efficient decoding of Reed-Solomon codes beyond half the minimum distance,"*IEEE Trans. on Info. Theory*, vol. 46, no. 1, pp. 246-257, Jan. 2000.

[32] X. Zhang and K. K. Parhi, "Fast factorization in soft-decision Reed-Solomon decoding," *IEEE Trans. on VLSI Systems*, vol. 13, no. 4, pp. 413-426, Apr. 2005.

[33] X. Zhang, "Further exploring the strength of prediction in the factorization of soft-decision Reed-Solomon decoding," *IEEE Trans. on VLSI Systems*, vol. 15, no. 7, pp. 811-820, Jul. 2007.

[34] J. Ma, A. Vardy and Z. Wang, "Low latency factorization architecture for algebraic soft-decision decoding of Reed-Solomon codes," *IEEE Trans. on VLSI Systems*, vol. 15, no. 11, pp. 1225-1238, Nov. 2007.

[35] J. Zhu and X. Zhang, "Factorization-free low-complexity Chase soft-decision decoding of Reed-Solomon codes," *Proc. IEEE Intl. Symp. on Circuits and Systems*, pp. 2677-2680, Taiwan, May 2009.

[36] J. Zhu and X. Zhang, "Efficient architecture for generalized minimum-distance decoder of Reed-Solomon codes," *Proc. of the IEEE International Conference on Acoustic, Speech and Signal Processing*, Dallas, TX, Mar. 2010.

[37] D. V. Sarwate and N. R. Shanbhag, "High-speed architectures for Reed-Solomon decoders," *IEEE Trans. on VLSI Systems*, vol. 9, no. 5, pp641-655, Oct. 2001.

[38] Y. Chen and K. K. Parhi, "Area efficient parallel decoder architecture for long BCH codes," in *Proc. of IEEE International Conference on Acoustics, Speech, and Signal Processing*, pp. 73-76, Montreal, Canada, May 2004.

[39] X. Zhang, "Partial parallel factorization in soft-decision Reed-Solomon decoding," *Proc. ACM Great Lakes Symp. VLSI*, pp. 272-277, Philadelphia, PA, Apr. 2006.