Efficient Codeword Recovery Architecture for Low-complexity Chase Reed-Solomon Decoding

Xinmiao Zhang and Yu Zheng
Case Western Reserve University
e-mails: {xinmiao.zhang, yu.zheng3}@case.edu

***Abstract*—Compared to other soft-decision decoding algorithms of Reed-Solomon (RS) codes, the low-complexity Chase (LCC) algorithm can achieve better performance-complexity tradeoff. In order to reduce the complexity of the LCC decoding, the re-encoding and coordinate transformation techniques can be applied for high-rate codes. In this case, to recover a codeword for an (n, k) RS code, an erasure decoding needs to be employed after the errors in the k most reliable symbols are corrected by making use of the interpolation output. Alternatively, it can be done by reversing the coordinate transformation at the interpolation output followed by evaluating a polynomial of degree k over finite field elements. These approaches lead to significant overhead. A novel scheme and efficient architectures are proposed in this paper for codeword recovery in the LCC decoding. Our scheme computes the codeword symbols consecutively by applying the Chien search over the interpolation output and a polynomial of degree $n - k$. The Chien search can be implemented by constant multipliers, which cost much less area than general multipliers required by previous approaches. For the LCC decoding with eight test vectors for a (255, 239) RS code, adopting the proposed codeword recovery architecture can lead to 15% area reduction without sacrificing the throughput.**

I. Introduction

Reed-Solomon (RS) codes are widely used error-correcting codes. In algebraic soft-decision decoding (ASD) [1]–[3] of RS codes, the reliability information from the channel is incorporated into an algebraic interpolation process. With polynomial complexity, these algorithms can achieve significant coding gain over traditional hard-decision decoding. Despite having 2^η test vectors, the low-complexity Chase (LCC) ASD algorithm [3] can achieve better complexity-performance tradeoff than other ASD algorithms. This is mainly because that the multiplicities of the interpolation points in the LCC decoding are one. Accordingly, the interpolation and factorization steps, which are the major steps of ASD algorithms, can be greatly simplified.

For high-rate codes, the re-encoding and coordinate transformation [4], [5] can be employed to reduce the complexity of the interpolation to practical level. The throughput and area requirement of the LCC interpolation can be further improved by using the architectures in [6]–[9]. In addition, it was discovered in [10] that, for an (n, k) RS code, the output of the re-encoded and transformed interpolation can be directly used as locator and evaluator to correct the errors in the k most reliable code positions in the LCC decoding. Hence, the factorization step can be eliminated. However, an erasure decoding is required at the end to recover the $n - k$ least reliable codeword symbols. Another approach to recover the codeword is to reverse the coordinate transformation at the interpolation output to find a message polynomial, and then evaluate the polynomial over finite field elements to derive the codeword symbols [3]. Nevertheless, the message polynomial has degree $k-1$, and its evaluation is hardware demanding.

This material is based upon work supported by the National Science Foundation under Grant No.0802159

This paper proposes a novel scheme and efficient architectures for codeword recovery in the LCC decoding. Our scheme reverses the coordinate transformation after the interpolation and does not require erasure decoding. On the other hand, the proposed scheme evaluates polynomials of degree at most $n - k$ and then simple calculations are done on the evaluation values to derive the codeword symbols. In addition, the codeword symbols are calculated one after another for consecutive positions, as opposed to being computed for discrete positions in previous work. Accordingly, the Chien search that only involves constant multipliers can be adopted for the polynomial evaluations in the proposed scheme. Compared to a general finite field multiplier, a constant multiplier has much lower complexity. Applying the proposed architecture to the LCC decoding with eight test vectors for a (255, 239) RS code, an overall area reduction of 15% can be achieved without sacrificing the throughput.

The structure of this paper is as follows. Section II introduces the LCC decoding. Section III and IV detail the proposed codeword recovery scheme and architectures, respectively. Complexity analysis is provided in Section V, and Section VI draws conclusions.

II. Re-encoded and Transformed LCC Decoding

Without loss of generality, this paper considers an (n, k) RS code constructed over finite field $GF(2^q)$ ($q \in Z^+$), where $n = 2^q - 1$ for primitive codes. Assume that $f_i \in GF(2^q)$ is the ith information symbol and the corresponding message polynomial is $f(x) = f_0 + f_1 x + \cdots + f_{k-1} x^{k-1}$. The encoding of RS codes can be carried out by evaluating $f(x)$ at n distinct nonzero field elements, denoted by $\alpha_0, \alpha_1, \cdots, \alpha_{n-1}$, and the corresponding codeword is $c = [(f(\alpha_0), f(\alpha_1), \cdots, f(\alpha_{n-1})]$. Although $f(x)$ can be recovered by interpolating through the points $(\alpha_i, f(\alpha_i))$ $(0 \le i < n)$, $f(\alpha_i)$ are unknown at the receiver since they can be corrupted by channel noise. ASD algorithms try to recover $f(x)$ by interpolating through those more reliable points using higher multiplicities, and accordingly can correct more errors than traditional hard-decision decoding.

ASD algorithms have different multiplicity assignment schemes, but share the same interpolation and factorization steps. The multiplicity assignment decides the interpolation points and their multiplicities using the reliability information from the channel. Among various ASD algorithms, the LCC algorithm [3] can achieve better performance-complexity tradeoff. The multiplicity assignment of this algorithm first chooses η least reliable code positions, and assign two points (α_i, r_i) and (α_i, r_i') to each of these code positions. Here r_i is the hard-decision of the ith symbol in the received word r, and r_i' is different from r_i in only the least reliable bit. For each of the $n-\eta$ remaining positions, only one point, (α_i, r_i), is assigned. All points have multiplicity one. Then 2^η test vectors are formed by picking one point for each code position. In the interpolation, a bivariate polynomial, $Q(x, y)$, with minimum $(1, k-1)$ weighted degree is generated to pass all the n points in each test vector. After that, the

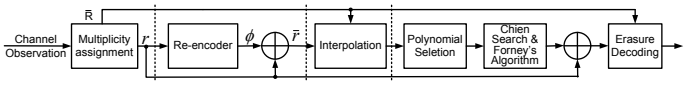

Fig. 1. The re-encoded LCC decoding

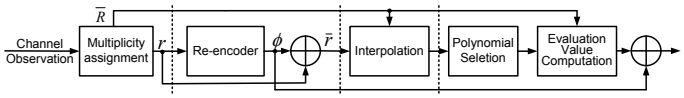

Fig. 2. The re-encoded LCC decoder with codeword recovery based on polynomial evaluation

factorization finds $f(x)$ by computing the factors of $Q(x,y)$ in the format of $y - f(x)$ with $\deg(f(x)) \leq k-1$.

Re-encoding and coordinate transformation [4] can be employed to simplify the interpolation in ASD algorithms. Denote the set of the k most reliable code positions in r by R. The re-encoding carries out erasure decoding to find a codeword ϕ, such that $\phi_i = r_i$ for $i \in R$. Then the decoding is done on $\bar{r} = r + \phi$ instead. Assume e is the error vector, *i.e.* $r = c + e$. Then $\bar{r} = c + \phi + e = \bar{c} + e$ is another codeword with the same error vector. In addition, $\bar{r}_i = r_i + \phi_i = 0$ for $i \in R$. Hence, the interpolation over the code positions in R can be pre-computed, and it only needs to be carried out on the points in the rest $n-k$ code positions. In the case of the LCC decoding, all points have multiplicity one. Hence the pre-computed polynomials have a factor $v(x) = \prod_{i \in R}(x + \alpha_i)$. This factor can be taken out by coordinate transformation, so that the length of the interpolation polynomials can be reduced by k.

Various schemes can be adopted to reduce the complexity of the LCC decoding. The backward interpolation scheme enables the sharing of the interpolation results for different test vectors. In the case of small η, the most efficient interpolation architecture is the unified backward-forward architecture in [9], which can generate the interpolation result for the next vector in a single iteration. The polynomial selection scheme in [9] passes the interpolation result of only one test vector to later decoding steps. In the case of the LCC decoding for high-rate codes, $Q(x,y)$ can be written as $q_1(x)y + q_0(x)$. The polynomial selection scheme chooses the $Q(x,y)$, in which $q_1(x)$ has its root number equals its degree.

To recover c, one approach is to first use $q_1(x)$ and $q_0(x)$ to find the errors in the code positions in R through Chien search and Forney's algorithm [10]. Although this approach does not require factorization, another erasure decoding is required to recover the $n-k$ symbols in $\bar{R}$ after the errors in R are corrected. The block diagram of the LCC decoding using this approach is shown in Fig. 1. Alternatively, $v(x)$ can be multiplied back to the interpolation output to derive the message polynomial $\bar{f}(x)$ of $\bar{c}$ [3]. After computing $\bar{c}_i = \bar{f}(\alpha_i)$, c can be recovered as $\bar{c} + \phi$ as shown in Fig. 2. Nevertheless, $\bar{f}(x)$ has degree $k-1$ and its evaluation has high complexity.

In the next section, a novel codeword recovery scheme is presented. Our scheme follows the approach in Fig 2, and does not require erasure decoding. On the other hand, only evaluations of low-degree polynomials are involved. Moreover, our design enables the adoption of the Chien search, which can be implemented by simple constant multipliers.

III. Proposed Codeword Recovery

Assume that the interpolation output is $Q(x,y) = q_1(x)y + q_0(x)$ when the re-encoding and coordinate transformation are applied. Reverse the coordinate transformation, the interpolation output corresponding to $\bar{r}$ should be

$$Q(x,y) = q_1(x)y + q_0(x)v(x) = q_1(x)(y - q_0(x)v(x)/q_1(x)).$$

Therefore, the factorization result can be directly derived from the above equation and the message polynomial $\bar{f}(x)$ corresponding to the codeword $\bar{c}$ is [3]

$$\bar{f}(x) = q_0(x)v(x)/q_1(x). \tag{1}$$

Since the degree of $v(x) = \prod_{i \in R}(x+\alpha_i)$ is $|R| = k$, its computation and the polynomial multiplication in (1) is hardware demanding for high-rate codes. Alternatively, denote $\prod_{i \in \bar{R}}(x + \alpha_i)$ by $w(x)$. Then $v(x)w(x) = \prod_{0 \leq i < n}(x + \alpha_i) = x^n - 1$ for primitive codes. Therefore, (1) can be rewritten as

$$\bar{f}(x) = q_0(x)(x^n - 1)/(q_1(x)w(x)). \tag{2}$$

Since $\deg(w(x)) = n-k$, the computation of $w(x)$ is much simpler than that of $v(x)$. In addition, the polynomial multiplication in the numerator of (2) is trivial because $\deg(q_0(x)) < n$. The codeword recovery scheme in [3] is to first compute $\bar{f}(x)$, and then evaluate $\bar{f}(x)$ at α_i to get $\bar{c}_i$. Although computing $\bar{f}(x)$ according to (2) instead has lower complexity, it does not change the degree of $\bar{f}(x)$, which can be as high as $k-1$. Evaluating a polynomial of such high degree leads to either long latency or large silicon area.

Instead of computing $\bar{f}(x)$ first and then evaluating $\bar{f}(x)$ at α_i, we propose to calculate $\bar{f}(\alpha_i)$ directly based on (2). However, $q_1(\alpha_i)$ and/or $w(\alpha_i)$ can be zero. Hence, the L'Hôpital rule needs to be applied in order to derive the evaluation values. This rule says if $d(x) = a(x)/b(x)$ and $a(\alpha_i) = b(\alpha_i) = 0$ for a certain α_i, then $d(\alpha_i) = a'(x)/b'(x)|_{\alpha_i}$, where $a'(x)$ denotes the derivative of $a(x)$. The following cases need to be considered:

Case A: $i \in R$ and $q_1(\alpha_i) = 0$. For primitive codes, $\alpha_i^n - 1$ is always zero. Moveover, $w(\alpha_i) \neq 0$ since $i \in R$ in this case. Hence the L'Hôpital rule needs to be applied to $(x^n - 1)/q_1(x)$. In addition, using the polynomial selection proposed in [9], the $q_1(x)$ of the chosen interpolation output only has simple roots. Hence $q_1'(\alpha_i) \neq 0$. Considering that $n = 2^q - 1$ is an odd number for primitive codes,

$$\bar{f}(\alpha_i) = \frac{q_0(x)}{w(x)} \frac{(x^n-1)'}{q_1'(x)}\Big|_{\alpha_i} = \frac{q_0(x)}{w(x)} \frac{x^{-1}}{q_1'(x)}\Big|_{\alpha_i}.$$

Case B: $i \in R$ and $q_1(\alpha_i) \neq 0$. In this case, $w(\alpha_i) \neq 0$ and

$$\bar{f}(\alpha_i) = \frac{q_0(x)}{q_1(x)} \frac{(x^n-1)}{w(x)}\Big|_{\alpha_i} = 0.$$

Case C: $i \in \bar{R}$ and $q_1(\alpha_i) \neq 0$. Since $w(\alpha_i) = 0$ for $i \in \bar{R}$, and $w(x)$ only has simple roots,

$$\bar{f}(\alpha_i) = \frac{q_0(x)}{q_1(x)} \frac{(x^n-1)'}{w'(x)}\Big|_{\alpha_i} = \frac{q_0(x)}{q_1(x)} \frac{x^{-1}}{w'(x)}\Big|_{\alpha_i}.$$

Case D: $i \in \bar{R}$ and $q_1(\alpha_i) = 0$. It has been proved in [10] that $q_1(x)$ and $q_0(x)$ can be written in the format of

$$\begin{cases} q_0(x) = p(x)\delta(x) \\ q_1(x) = p(x)\prod_{i \in R, e_i \neq 0}(x + \alpha_i), \end{cases} \tag{3}$$

where $p(x)$ does not have any factor $(x + \alpha_i)$ for $i \in R$. In the case that $i \in \bar{R}$ and $q_1(\alpha_i) = 0$, $p(\alpha_i)$ must be zero. Accordingly $q_0(\alpha_i)$ also equals zero. Moreover, $w(\alpha_i) = 0$ for $i \in \bar{R}$. Therefore,

$$\bar{f}(\alpha_i) = \frac{q_0'(x)}{q_1'(x)} \frac{(x^n-1)'}{w'(x)}\Big|_{\alpha_i} = \frac{q_0'(x)}{q_1'(x)} \frac{x^{-1}}{w'(x)}\Big|_{\alpha_i}.$$

Define the sum of the odd terms in a polynomial $a(x)$ by $a_{odd}(x)$. It can be derived that $xa'(x) = a_{odd}(x)$ over $GF(2^q)$. Accordingly,

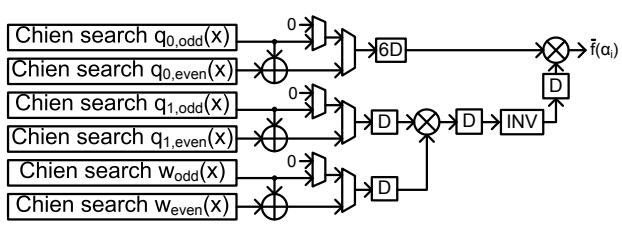

Fig. 3. The proposed FCSB codeword recovery architecture

the computation of $\bar{f}(\alpha_i)$ can be summarized as

$$\bar{f}(\alpha_i) = \begin{cases} \frac{q_0(\alpha_i)}{q_{1,odd}(\alpha_i)} \frac{1}{w(\alpha_i)} & if\ i \in R,\ q_1(\alpha_i) = 0 \\ \frac{q_0(\alpha_i)}{q_1(\alpha_i)} \frac{1}{w_{odd}(\alpha_i)} & if\ i \in \bar{R},\ q_1(\alpha_i) \neq 0 \\ \frac{q_{0,odd}(\alpha_i)}{q_{1,odd}(\alpha_i)} \frac{1}{w_{odd}(\alpha_i)} & if\ i \in \bar{R},\ q_1(\alpha_i) = 0 \\ 0 & if\ i \in R,\ q_1(\alpha_i) \neq 0 \end{cases} \quad (4)$$

As it can be observed from (4), only evaluation values of $q_1(x)$, $q_0(x)$ and $w(x)$ or the odd parts of these polynomials are needed to compute the codeword symbols in $\bar{c}$. The decoding trial over each of the test vectors in the LCC algorithm is a hard-decision decoding. For an (n, k) RS code, hard-decision decoding can correct at most $(n-k)/2$ errors. Hence, the degrees of the selected $q_0(x)$ and $q_1(x)$ can be at most $(n-k)/2$. Moreover, $\deg(w(x)) = n-k$. They are much lower than the degree of $\bar{f}(x)$, which can be $k-1$. Therefore, the polynomial evaluation in our proposed scheme has much lower complexity than that in [3]. In addition, our scheme does not need to compute $\bar{f}(x)$, and hence the polynomial multiplication and division in (1) or (2) can be saved. Compared to the codeword recovery approach in [10], which includes a Chien search & Forney's algorithm and an erasure decoder as shown in Fig. 1, the proposed scheme also has significantly lower complexity since it does not need an extra erasure decoder at the end.

IV. VLSI Architectures for the Proposed Codeword Recovery

Which equation in (4) to use depends on whether $q_1(\alpha_i)$ equals zero. Assume that $\alpha_0, \alpha_1, \cdots, \alpha_{n-1} = 1, \omega, \omega^2, \cdots, \omega^{2^q-2}$, where ω is a primitive element of $GF(2^q)$. The Chien search can be adopted to evaluate $q_1(x)$ over α_i for $i = 0, 1, \cdots n-1$. In addition, a polynomial can be divided into even and odd parts, and the evaluation values of the odd part from the Chien search can be reused in the computations in (4). $q_1(x)$ can have at most $(n-k)/2$ roots in R. Therefore, for $i \in R$, at most $(n-k)/2$ nonzero $\bar{f}(\alpha_i)$ will be computed using the first equation in (4). The computations of these values, as well as the $(n-k)$ values for $i \in \bar{R}$, can start after the corresponding $q_1(\alpha_i)$ is calculated in the Chien search. Although at most $3/2(n-k)$ values need to computed using (4) in this case, regular finite field multipliers are needed to compute $q_0(\alpha_i)$ and $w(\alpha_i)$, since those $i \in R$ with $q_1(\alpha_i) = 0$ and $i \in \bar{R}$ are not consecutive. This approach is denoted by partial Chien search based (PCSB) codeword recovery.

Compared to a general finite field multiplier, a constant multiplier can be implemented by much simpler logic. For example, a general multiplier for $GF(2^8)$ can be implemented by 64 XOR gates and 48 AND gates. However, the multiplication by a primitive element can be done by three XOR gates. Inspired by this, we propose to carry out Chien search for $q_0(x)$ and $w(x)$ simultaneously with the Chien search for $q_1(x)$. Then depending on whether $q_1(\alpha_i) = 0$ and $i \in R$, proper evaluation values are picked to compute $\bar{f}(\alpha_i)$

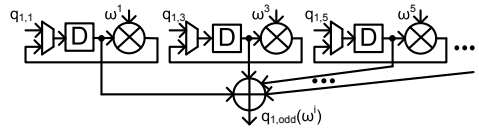

Fig. 4. The Chien search architecture for $q_{1,odd}(x)$

according to (4). This full Chien search based (FCSB) codeword recovery architecture is illustrated in Fig. 3. The second column of multiplexors in this figure pick evaluation values according to (4), and registers are added for the purpose of pipelining. In the case that $i \in R$ and $q_1(\alpha_i) \neq 0$, zeros will be passed through the first column of multiplexors in Fig. 3, so that the switching activities of the later circuits are reduced. This is of great importance in order to keep the power consumption of the FCSB codeword recovery scheme competitive, because $\bar{f}(\alpha_i)$ equals zero in this case and is otherwise not computed in the PCSB approach.

As an example, the Chien search architecture for $q_{1,odd}(x)$ is shown in Fig. 4. At the beginning, the polynomial coefficients are passed through the multiplexors, and the output is $q_{1,odd}(1)$. After that, the multiplexors select the outputs of the constant multipliers, and $q_{1,odd}(\omega), q_{1,odd}(\omega^2), \cdots$ will be computed in successive clock cycles.

V. Hardware Complexity Analysis

In this section, the complexity of the proposed FCSB codeword recovery for an example (255, 239) RS code over $GF(2^8)$ is first compared with those of the PCSB scheme and the approach in [10], which includes Chien search & Forney's algorithm and erasure decoding. Then the saving can be brought by adopting the proposed FCSB architecture in the overall LCC decoder with $\eta = 3$ is analyzed. The codeword recovery in [3] involves evaluating a polynomial of degree $k-1$. The polynomial evaluation requires $(k-1)/(2(n-k)) = 7.4$ times higher complexity compared to our proposed approach. Hence, the scheme in [3] is not further considered for comparison.

Table I lists the complexities of the codeword recovery schemes. The critical paths of these architectures are all one multiplier, one adder and one multiplexor, and each of the adders and multiplexors in the table has two inputs. Since the degree of $q_1(x)$ and $q_0(x)$ can be at most $(n-k)/2 = 8$, and $\deg(w(x)) = n-k = 16$, the Chien search over these polynomials in the FCSB codeword recovery can be done in 255 clock cycles using 32 multiplexor-constant multiplier-register loops shown in Fig. 4. To reduce the area, the finite field inverter in Fig. 3 can be implemented using composite field arithmetic, and pipelined into four stages. Hence, the FCSB codeword recovery takes $255 + 7 = 262$ clock cycles considering the pipelining latency.

In the PCSB scheme, the Chien search over $q_1(x)$ is first carried out using $\deg(q_1(x)) = 8$ constant multipliers in 255 clock cycles. The $9th$ constant multiplier is used to keep track of the polynomial representations of the finite field elements evaluated. Then at most $3(n-k)/2$ evaluation values of $q_0(x)$ and $w(x)$ over the roots of $q_1(x)$, as well as the code positions in $\bar{R}$, are computed using $\deg(q_0(x)) + \deg(w(x)) = 24$ regular multipliers. After that, the codeword symbols are computed using an architecture similar to the right part of Fig. 3.

To further compare the FCSB and PCSB architectures, synthesis and power analysis have been carried out using $0.13\mu m$ SMIC library with 1.08v power supply for a sample codeword with five errors in R, and results are listed in Table I. Although $q_0(x)$ and $w(x)$ are evaluated over all n field elements in the FCSB scheme, they are implemented by Chien search consisting of constant multipliers,

TABLE I

HARDWARE COMPLEXITY COMPARISON FOR CODEWORD RECOVERY SCHEMES FOR $(255, 239)$ LCC DECODER

	$GF(2^8)$ Multiplier	$GF(2^8)$ Adder	$GF(2^8)$ Inverter	Constant Multiplier	Mux	Register (bits)	Latency (# of clks)	Area (mm^2)	Power (mW)
FCSB codeword recovery	2	33	1	32	328	408	262	0.96	9.492
PCSB codeword recovery	26	35	1	9	136	1048	272	2.81	8.804
Chien search Forney's algorithm & erasure decoder	21	65	2	48	528	1080	333	-	-

TABLE II

HARDWARE REQUIREMENT OF LCC DECODERS WITH $\eta = 3$ FOR A $(255, 239)$ RS CODE

	$GF(2^8)$ Multiplier	$GF(2^8)$ Adder	$GF(2^8)$ Inverter	Constant Multiplier	Mux	ROM (bits)	RAM (bits)	Register (bits)	Latency (# of clks)
Re-encoder	16	41	1	16	208	0	0	824	360
Interpolation	21	19	1	0	120	0	544	216	363
Polynomial selection	0	104	0	104	200	576	0	376	19
Pipelining for LCC decoder with FCSB recovery	0	0	0	0	0	0	7445	0	N/A
Pipelining for previous LCC decoder	0	0	0	0	0	0	6808	0	N/A
LCC decoder with FCSB recovery	39	197	3	152	856	576	7989	1824	363
Previous LCC decoder [10]	58	229	4	168	1056	576	7352	2496	363

which are much smaller than general multipliers. In addition, the FCSB scheme does not need extra registers to hold the results of the Chien search on $q_1(x)$ until the evaluation values on $q_0(x)$ and $w(x)$ are available to compute codeword symbols. As a result, the FCSB architecture only requires around 1/3 the area of the PCSB architecture, and consumes slightly more power.

The formula for the Forney's algorithm for error magnitude computation is very similar to the equations in (4). In addition, the erasure decoder can be implemented efficiently by using the architecture in [11], which computes the erasure locator and evaluator polynomials directly and reformulates the involved equations to simplify the computations. It can be observed from Table I that, apparently, the codeword recovery employing erasure decoding has much higher hardware complexity than the proposed FCSB scheme.

Since the proposed PCSB codeword recovery has higher complexity than the FCSB scheme, it is not further evaluated for overall decoder complexity reduction. The complexity of the LCC decoder employing the FCSB codeword recovery is listed in Table II. The re-encoder mainly consists of an erasure decoder. Its architecture can be found in [11]. For small η, the most efficient LCC interpolation architecture is the unified backward-forward architecture in [9]. One unified interpolator can be employed to take care of the interpolation over eight test vectors in 363 clock cycles. To match the speed of the interpolation, parallel Chien search is adopted in the polynomial selection block. For the purpose of conciseness, the complexity of the FCSB codeword recovery is not listed in Table II again.

For the previous decoder [10] using erasure decoder for codeword recovery shown in Fig. 1, the same re-encoder, interpolation and polynomial selection architectures can be used. Moreover, pipelining can be applied to the overall decoder according to the cutsets shown by the dashed lines in Fig. 1 and 2. To reduce the area requirement, pipelining can be done by using memories instead of registers. Although the decoder in Fig. 1 needs to store the code positions in $\bar{R}$ until the last stage, it does not need to record a flag bit for each position to tell whether it belongs to R. Hence, the memory for pipelining in the previous decoder is smaller. In a pipelined decoder, the achievable throughput is decided by the pipelining stage with the longest latency. Hence, the parallel processing factors of the building blocks can be adjusted so that each pipelining stage achieves about the same latency with minimum hardware in order to increase the hardware utilization efficiency. This guideline is used to scale each decoder component and derive the numbers listed in Table II.

The decoder with the FCSB codeword recovery has the same critical path as the previous design, and requires the same number of clock cycles to decode a word as can be observed from Table II. Therefore, the two decoders can achieve the same throughput. From architectural analysis, it can be estimated that the area of the decoder with FCSB codeword recovery is equivalent to 23093 XOR gates. Compared to the 27216 XOR gates required by the previous decoder, this design needs 15% less area.

VI. CONCLUSION

This paper proposed a novel codeword recovery scheme for the LCC decoding. The proposed scheme does not require erasure decoding. Instead, it carries out Chien search on polynomials of low degree. Compared to previous approaches, significant area and power reduction can be achieved without sacrificing the throughput. Adopting the proposed scheme, the area requirement of the overall LCC decoder can be substantially reduced. Future work will address further reducing the complexity of the LCC decoder with large η.

REFERENCES

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

[2] J. Jiang and K. Narayanan, "Algebraic soft-decision decoding of Reed-Solomon codes using bit-level soft information," *IEEE Trans. Info. Theory*, vol. 54, no. 9, pp. 3907-3928, Sep. 2008.

[3] J. Bellorado and A. Kavcic, "Low-complexity soft-decoding algorithms for Reed-Solomon codes -part I: an algebraic soft-in hard-out Chase decoder," *IEEE Trans. on Info. Theory*, vol. 56, pp. 945-959, Mar. 2010.

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

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

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

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

[8] J. Zhu, X. Zhang and Z. Wang, "Backward interpolation architecture for algebraic soft-decision Reed-Solomon decoding," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 17, no. 11, pp. 1602-1615, Nov. 2009.

[9] X. Zhang and J. Zhu, "Algebraic soft-decision decoder architectures for long Reed-Solomon codes," *IEEE Trans. on Circuits and Syst. -II*, vol. 57, no. 10, pp. 787-792, Oct. 2010.

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

[11] J. Zhu and X. Zhang, "High-speed reencoder design for algebraic soft-decision decoding of Reed-Solomon codes," *Proc. IEEE Intl. Symp. on Circuits and Syst.*, Paris, France, Mar. 2010.