Interpolation-based Chase BCH Decoder

Xinmiao Zhang
Case Western Reserve University
Email: xinmiao.zhang@case.edu

***Abstract*—BCH codes are adopted in many applications, such as flash memory and optical communications. Compared to hard-decision decoders, the soft-decision Chase BCH decoder can achieve significant coding gain by trying multiple test vectors. Previously, one-pass Chase BCH decoding schemes based on the Berlekamp's algorithm are used to share intermediate results among the decoding trials. In this paper, it will be shown that the interpolation-based one-pass Chase BCH decoder has much lower hardware complexity. Techniques for simplifying the implementation architecture for each step of the interpolation-based decoder are summarized. For a (4200, 4096) BCH code, the interpolation-based Chase decoder with 16 test vectors has 2.2 times higher hardware efficiency than that based on the Berlekamp's algorithm in terms of throughput-over-area ratio.**

I. Introduction

BCH codes are used in many systems such as flash memory and optical communications. Traditional hard-decision decoding (HDD) algorithms of BCH codes, such as the Berlekamp's algorithm, can only correct $t = \lfloor d_{min}/2 \rfloor$ errors, where d_{min} is the minimum distance of the code. By flipping the η least reliable bits, and carrying out decoding trails over 2^η test vectors, the soft-decision Chase decoding algorithm can correct up to $t + \eta$ errors. Another advantage of the Chase algorithm is that its error-correcting capability can be easily tuned by changing η. When better performance is needed, more vectors are tested. This requires no change in the codeword length, code rate, or the en/decoder architecture. This feature may help to combat time-varying channel condition in communications and process variation in flash memory.

To share computations among the decoding trials, one-pass schemes were proposed for BCH codes [1], [2]. They adopt the Berlekamp's algorithm to compute the error locator of the first test vector. Then the locators of the other vectors are derived in one run by updating the results of the Berlekamp's algorithm. The scheme in [2] was further simplified and implementation architectures were designed in [3]. Nevertheless, the Chase BCH decoding can be more efficiently implemented using an interpolation-based scheme [4].

The interpolation-based Chase BCH decoder is developed through extending the designs for interpolation-based Reed-Solomon (RS) decoder [5]. Although many simplification techniques for individual decoding steps have been proposed, it is difficult to follow the overall Chase BCH decoder architecture from available literature. This paper assembles the best design for each decoding step, and provides a comprehensive description of the implementation of the interpolation-based Chase BCH decoder. Particularly, systematic re-encoding [6], backward-forward interpolation [7], [8], polynomial selection based on binary testing [4], and simplified Chien-search-based codeword recovery [4] will be detailed. Simulation results and hardware complexity comparisons are provided for an example (4200, 4096) BCH code. The interpolation-based Chase decoder with 16 test vectors can achieve 2.2 times higher hardware efficiency in terms of throughput-over-area-ratio than that based on the Berlekamp's algorithm [3] without sacrificing the error-correcting performance.

This material is based upon work supported by NSF grant 0846331.

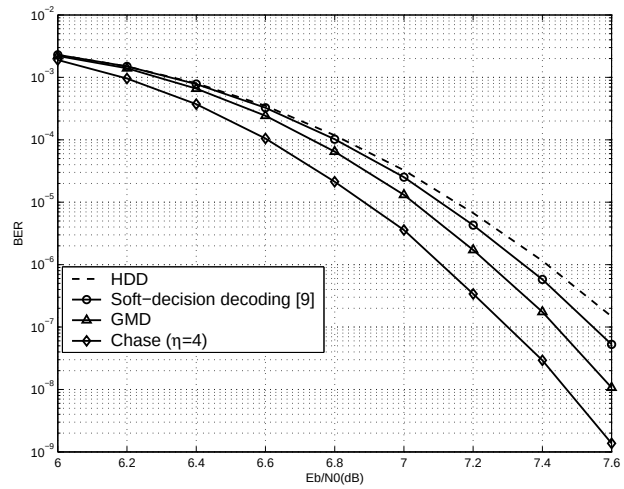

Fig. 1. BERs of BCH decoding for (4200, 4096) code over AWGN channel

The structure of this paper is as follows. Section II introduces interpolation-based Chase BCH decoding. The most efficient implementation scheme for each decoding step is presented in Section III. Complexity comparisons are provided in Section IV, and conclusions follow in Section V.

II. Interpolation-based Chase BCH decoding

This paper considers an (n, k) t-bit error-correcting binary BCH code constructed over $GF(2^p)$. In the Chase algorithm, the η least reliable bits in the received word are flipped to form 2^η test vectors, and the decoding is done for each vector. Hence, the Chase algorithm can correct up to $t+\eta$ errors. The bit error rates (BERs) of several decoding algorithms over the AWGN channel are shown in Fig. 1 for a shortened (4200, 4096) BCH code constructed over $GF(2^{13})$ [4]. $t = 8$ for this code. With $\eta = 4$, the Chase decoder significantly outperforms both the generalized minimum-distance (GMD) decoder and the soft-decision decoder in [9], which assumes at most one error is not located in the $2t$ least reliable bits.

The interpolation-based decoding is developed through interpreting the codeword symbols as evaluation values of the

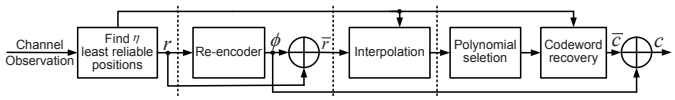

Fig. 2. Re-encoded interpolation-based Chase decoder

message polynomial. However, binary BCH codes can not be encoded using such an evaluation map method. Hence, interpolation-based BCH decoding is done through considering an (n, k) t-bit error-correcting binary BCH codeword as a codeword of an (n, k') t-symbol error-correcting RS code constructed over the same finite field.

For an (n, k') RS code over $GF(2^p)$, the codeword symbol c_i equals $f(\alpha_i)$ $(0 \leq i < n)$ using evaluation map encoding. $f(x)$ is the degree $k' - 1$ message polynomial, and $\alpha_0, \alpha_1, \ldots, \alpha_{n-1}$ are n distinct nonzero elements of $GF(2^p)$. Apparently, $f(x)$ can be recovered by interpolating (α_i, c_i) for $0 \leq i < n$. However, a test vector, r, may be different from the codeword, c, due to the channel noise. If the number of errors in r is at most t, then a polynomial $Q(x, y)$ that passes each point, (α_i, r_i), with $(1, k'-1)$ minimum weighted degree has a factor $y - f(x)$ [5]. Such $Q(x, y)$ can be computed using bivariate interpolation, which is the most complicated step in the decoder.

Although the interpolation-based decoding originally consists of two steps: interpolation and factorization, it can be implemented according to Fig. 2 after simplification schemes are applied. The re-encoding technique [10] reduces the interpolation complexity. The basic idea is to find another codeword ϕ that equals the hard-decision of the received word, r, in k positions denoted by the set R. The rest code positions are represented by $\bar{R}$. Since $\bar{r} = r + \phi = (c + \phi) + e$ is another codeword $\bar{c} = c + \phi$ corrupted by the same error vector, e, the decoding can be done on $\bar{r}$ instead. The advantage is that $\bar{r}_i = 0$ for $i \in R$. Hence, the interpolation over the corresponding point can be pre-solved as $v(x) = \prod_{i \in R}(x + \alpha_i)$, and the bivariate interpolation only needs to be done on the remaining $n - k$ points in $\bar{R}$. Moreover, $v(x)$ is excluded from the interpolation process using coordinate transformation. As a result, the interpolation only needs to be carried out on $(\alpha_i, \beta_i = (r_i + \phi_i)/v(\alpha_i))$ for $i \in \bar{R}$.

Ordering the test vectors in a Gray code manner, *i.e.* adjacent vectors only have one different bit, the interpolation results for all test vectors are derived in one run by using a backward-forward interpolation scheme [7], [8]. To avoid repeating the remaining decoding steps for each test vector, only the interpolation output polynomial most likely leading to successful decoding is selected and passed to the remaining steps. When the re-encoding and coordinate transformation are adopted, the interpolation output is a polynomial passing the transformed points in the code positions in $\bar{R}$ of the corresponding test vector. $v(x)$ does not have to be multiplied back to this polynomial to derive $f(x)$. For high-rate codes, the y-degree of the polynomials involved in the interpolation is only one in the Chase decoding. In this case, the interpolation output polynomial can be directly used as the error locator and evaluator to compute the errors in R [11]. Further simplifications were made in [12] to recover $\bar{c}$ directly from the interpolation output using a Chien search architecture.

III. Simplification schemes for Chase BCH decoder

Many techniques have been proposed to simplify the interpolation-based decoder. This section highlights the most efficient design of each decoding step for BCH codes.

A. Systematic re-encoding

The interpolation-based decoding was first developed for algebraic soft-decision (ASD) decoding of RS codes [5]. For general ASD decoders, the interpolation points in those more reliable code positions have higher multiplicities, and excluding them from the interpolation through re-encoding leads to more complexity reduction. Hence, in previous re-encoder designs for general ASD decoders, the k most reliable code positions were selected as R. Finding a codeword ϕ equaling to r in k arbitrary positions needs erasure decoding, which has complicated erasure locator and evaluator polynomial calculations. In the Chase decoding, each interpolation point has the same multiplicity. Picking any k of them leads to the same complexity reduction in the interpolation. To simplify the re-encoder, the last k positions, which are also called the systematic positions, can be chosen [6]. Accordingly, the re-encoding is reduced to systematic encoding, which is implemented by a simple linear feedback shift register (LFSR), and the architecture can be found in [13].

B. Unified backward-forward interpolation

Previously, the interpolation problem is solved by using the Kötter's algorithm [14]. This algorithm adds points to the interpolation curve, and hence is called a forward interpolation algorithm. For the Chase decoding of high-rate codes, only two polynomials are involved in the interpolation. They are initialized as $Q^{(0)}(x, y) = 1$ and $Q^{(1)}(x, y) = y$, and is forced to pass a point (α_i, β_i) using Algorithm A, in which w denotes the weighted degree of the polynomial. More points can be added to the interpolation curve by carrying out Algorithm A iteratively.

Algorithm A: Forward interpolation for adding (α_i, β_i)

compute $Q^{(l)}(\alpha_i, \beta_i)$ $(l = 0, 1)$

$u = \arg\min_l (w_l | Q^{(l)}(\alpha_i, \beta_i) \neq 0)$, $v = \{0, 1\} \backslash u$

$Q^{(v)}(x,y) \Leftarrow Q^{(u)}(\alpha_i,\beta_i) Q^{(v)}(x,y) + Q^{(v)}(\alpha_i,\beta_i) Q^{(u)}(x,y)$

$Q^{(u)}(x,y) \Leftarrow Q^{(u)}(x, y)(x + \alpha_i)$, $w_u \Leftarrow w_u + 1$

When re-encoding is applied, decoding is done on $\bar{r}$ and $\bar{r}_i = 0$ for $i \in R$. Hence, the corresponding interpolation points are $(\alpha_i, 0)$. Following the forward interpolation in Algorithm A, the interpolation result over these points is $\{Q^{(0)}(x, y) = \prod_{i \in R}(x + \alpha_i),\ Q^{(1)}(x, y) = y\}$. Apply coordinate transformation $y \Leftarrow y / \prod_{i \in R}(x + \alpha_i)$, and take $v(x) = \prod_{i \in R}(x + \alpha_i)$ out of both polynomials. Then

the interpolation can start with the same initial polynomials $\{Q^{(0)}(x,y)=1, Q^{(1)}(x,y)=y\}$, but only needs to be done for the transformed points $(\alpha_i,(r_i+\phi_i)/v(\alpha_i))$ in the remaining $n-k$ positions in $\bar{R}$. Also $(1,-1)$ weighted degree should be used in the interpolation since $k-1-\deg(v(x))=-1$.

Repeating the interpolation for each test vector from the beginning would waste much computation. The test vectors in the Chase algorithm can be ordered so that adjacent vectors only have one pair of different points (α_i,β_i) and (α_i,β_i'). To share computations among the interpolation for different test vectors, a backward interpolation technique was proposed in [7]. Assume that $Q^{(l)}(x,y)=q_0^{(l)}(x)+q_1^{(l)}(x)y$ $(l=0,1)$ is a pair of polynomials passing (α_i,β_i) computed by the forward interpolation. (α_i,β_i) can be eliminated from the interpolation result using Algorithm B regardless of the order of the points that have been interpolated.

Algorithm B: Backward interpolation for deleting (α_i,β_i)

compute $q_1^{(l)}(\alpha_i)$ $(l=0,1)$
$u=\arg\min_l(w_l|q_1^{(l)}(\alpha_i)\neq 0)$, $v=\{0,1\}\backslash u$
$Q^{(v)}(x,y)\Leftarrow(q_1^{(u)}(\alpha_i)Q^{(v)}(x,y)+q_1^{(v)}(\alpha_i)Q^{(u)}(x,y))/(x+\alpha_i)$
$Q^{(u)}(x,y)\Leftarrow Q^{(u)}(x,y)$, $w_v\Leftarrow w_v-1$

Given the interpolation result of the current test vector, the result of the next test vector is directly derived in only two iterations, one for deleting (α_i,β_i) using the backward interpolation, and another for adding (α_i,β_i') using the Kötter's forward interpolation. Therefore, the interpolation for all test vectors in the Chase decoding is done by following the procedure in Algorithm C. For each test vector, the polynomial of lower weighted degree is selected as the interpolation output.

Algorithm C: Interpolation for Chase Decoding

Initialization: $Q^{(0)}(x,y)=1$, $w_0=0$
$Q^{(1)}(x,y)=y$, $w_1=-1$
for each point (α_i,β_i) in the **first vector**
add (α_i,β_i) (forward interpolation)
for **each of the next vector** (has (α_i,β_i') instead of (α_i,β_i))
C1: delete (α_i,β_i) (backward interpolation)
C2: add (α_i,β_i') (forward interpolation)

In the backward-forward interpolation for deriving the result of the next test vector, the forward interpolation iteration (line C2 in Algorithm C) needs to wait until the backward interpolation (line C1 in Algorithm C) is completed due to data dependency. This prohibits further speedup. To solve this problem, a unified backward-forward interpolation was developed by adopting a look-ahead technique to compute the evaluation values needed in the forward interpolation iteration based on the input polynomials to the backward interpolation iteration [8]. For the purpose of clarity, denote the input polynomials to the forward interpolation iteration, which are the results of the backward interpolation iteration, by $F^{(l)}(x,y)$ $(l=0,1)$. It was shown in [8] that the evaluation values in the forward interpolation can be computed as

$$F^{(v)}(\alpha_i,\beta_i')=q_1^{(u)}(\alpha_i)\frac{\partial Q^{(v)}(x,y)}{\partial x}\bigg|_{(\alpha_i,\beta_i')}+q_1^{(v)}(\alpha_i)\frac{\partial Q^{(u)}(x,y)}{\partial x}\bigg|_{(\alpha_i,\beta_i')}$$

$$F^{(u)}(\alpha_i,\beta_i')=Q^{(u)}(\alpha_i,\beta_i')$$

Accordingly, the evaluation values for both the backward and forward interpolations can be computed simultaneously based on the input polynomials to the backward interpolation. Then the remaining polynomial updating for both iterations are combined. $\frac{\partial Q^{(l)}(x,y)}{\partial x}\big|_{(\alpha_i,\beta_i')}=(q_1^{(l)}(x))'\big|_{\alpha_i}\beta_i'+(q_0^{(l)}(x))'\big|_{\alpha_i}$, and the derivative of an univariate polynomial over finite fields of characteristic two is the collection of the odd-degree terms. Hence many units can be shared among the computations of $q_1^{(l)}(\alpha_i)$ and $F^{(l)}(\alpha_i,\beta_i')$. As a result, with small silicon area increase, the unified backward-forward interpolator only needs one single iteration to derive the interpolation result of the next test vector. Detailed VLSI architectures for the unified backward-forward interpolator can be found in [8].

There is one more issue to be addressed when systematic re-encoding is employed. Some of the η least reliable code positions may belong to the systematic positions. The points in these positions do not exist in the interpolation output since they have been taken out by coordinate transformation. On the other hand, they need to be deleted from the interpolation result in order to apply the backward-forward interpolation. To solve this dilemma, it was proposed in [6] to modify the coordinate transformation on the fly and bring the points back to the interpolation output when needed. Specifically, when (α_m,β_m) with $m\in R$ needs to be deleted from the interpolation output for the first time, another coordinate transformation $y\Leftarrow y(x+\alpha_m)$ is applied. In addition, $(x+\alpha_m)$ is taken from $v(x)$ and multiplied back to the interpolation polynomials. As a result, the polynomials are updated as follows before the backward-forward interpolation continues.

$$\begin{cases}Q^{(0)}(x,y)\Leftarrow q_0^{(0)}(x)(x+\alpha_m)+q_1^{(0)}(x)y\\ Q^{(1)}(x,y)\Leftarrow q_0^{(1)}(x)(x+\alpha_m)+q_1^{(1)}(x)y\end{cases}$$

To follow the modified coordinate transformation, $(\alpha_i+\alpha_m)$ needs to be multiplied to the β_i coordinate of any point that will be backward or forward interpolated for later test vectors. The unified backward-forward interpolation architecture accommodating systematic re-encoding has been developed in [6]. Adopting systematic re-encoding only incurs small overhead to the interpolator.

For the decoding of a code over $GF(2^p)$, the hardware complexities of the interpolators are summarized in Table I. In this table, d_x denotes the maximum x-degree of the polynomials in the corresponding iteration. It increases gradually during the forward interpolation iterations for the first test vector, and remains about the same during the backward-forward interpolation for later vectors. The interpolators process the coefficients in $q_0^{(l)}(x)$ and $q_1^{(l)}(x)$ serially, and overlap the polynomial updating of an iteration with the evaluation value

TABLE I
HARDWARE COMPLEXITIES OF INTERPOLATORS IN THE CHASE DECODING WITH 2^η TEST VECTORS FOR (n, k) RS CODE OVER $GF(2^p)$

	Forward-only	Backward-forward [7]	Unified backward-forward for syst. re-encoding [6]
# of iter.	$(n-k)+\eta(2^\eta-1)$	$(n-k)+2(2^\eta-1)$	$(n-k)+2^\eta-1$
# of clks/iter.	d_x+3	d_x+4	d_x+5
Hardware requirement			
GF mult.	12	14	22
GF add.	10	12	21
GF inv.	0	0	1
Mux	$2p$	$6p$	$22p$
RAM(bit)	$(8(n-k)-4\eta)p$	$4(n-k)p$	$4(n-k)p$
Register	$8p$	$16p$	$32p$

computation of the next iteration. Hence, the number of clock cycles needed for an iteration is d_x plus the pipelining latency of the architecture. All the interpolators are designed to have the same critical path. The complexity of the backward-forward interpolator [7] in Table I does not include that needed to accommodate systematic re-encoding.

Originally, when only forward interpolation is available, the intermediate interpolation result over the $n-k-\eta$ common points is stored. Then the interpolation for the remaining η points in each test vector starts from this stored result and takes η iterations. Although the unified interpolator has longer pipelining latency and larger silicon area requirement, it requires only one single iteration for the next test vector. Without storing any intermediate result, it can achieve almost η times speedup over the forward-only interpolator when η is not small.

C. Polynomial selection based on binary testing

Using the backward-forward interpolation, the interpolation results for all test vectors are computed in one run. However, it is very difficult to share computations among the remaining steps for different test vectors. Instead, only the interpolation output polynomial most likely leading to successful decoding is passed to the remaining steps. For BCH codes, a simple method was proposed in [4] to select such a polynomial.

As explained previously, to facilitate the application of interpolation-based decoding, t-bit error-correcting (n, k) BCH codewords are considered as codewords of a t-symbol error-correcting (n, k') RS code constructed over the same field. $n-k'$ is always $2t$, while $n-k$ is equal to or slightly less than pt. There are 2^k binary BCH codewords. They form a tiny proportion of the $2^{pk'}$ codewords of the (n, k') RS code over $GF(2^p)$. Hence, if a test vector is undecodable, the chance of returning a binary BCH codeword is extremely small, especially for long codes. Making use of this property, it was proposed in [4] to select the interpolation output polynomial based on whether it will lead to a binary codeword.

Testing whether each symbol in the corresponding recovered codeword c is binary requires the entire decoding to be finished first. This would invalidate the purpose of having polynomial selection. It has been analyzed in [4] that using the Chien-search-based codeword recovery [12], the recovered c_i for $i \in R$ are mostly binary even if the corresponding test vector is undecodable. On the other hand, c_i for $i \in \bar{R}$ are mostly non-binary in the undecodable cases. Therefore, it was proposed in [4] to select the interpolation output polynomial whose corresponding c_0 and c_1 are binary. Simulation results showed that if the test vectors are interpolated in Gray code order and according to decreasing reliability as much as possible, selecting only the first interpolation output polynomial meeting this criterion does not lead to any noticeable performance loss. Actually, testing any two symbols in $\bar{R}$ would lead to the same performance. Since a symbol in $\bar{R}$ can be accidentally binary, testing only one symbol has tiny performance degradation.

If a test vector is decodable, the message polynomial $\bar{f}(x)$ corresponding to $\bar{c}$ using evaluation map encoding is computed as [6]

$$\bar{f}(x) = \frac{q_0(x)}{q_1(x)} \frac{v(x)}{\prod_{j \in S_F}(x+\alpha_j)}, \tag{1}$$

where S_F is the set of systematic code positions that have been flipped. Then $c_i = \bar{c}_i + \phi_i = \bar{f}(\alpha_i) + \phi_i$. $\bar{f}(\alpha_i)$ is derived by multiplying and dividing $q_0(\alpha_i)$, $q_1(\alpha_i)$, $v(\alpha_i)$ and $\prod_{j \in S_F}(\alpha_i+\alpha_j)$ according to (1). In the case that a term in the denominator of (1) has zero evaluation value, there must be a term in the numerator with zero evaluation value [6]. Then the L'Hôpital rule is applied to compute $\bar{f}(\alpha_i)$. This rule says that if $d(x) = a(x)/b(x)$ and $a(\alpha_i) = b(\alpha_i) = 0$ for a certain α_i, then $d(\alpha_i) = a'(x)/b'(x)|_{\alpha_i}$. To make the computation of c_0 and c_1 easier, assume that $\{\alpha_0, \alpha_1, \alpha_2, \dots\} = \{1, \omega, \omega^2, \dots\}$, where ω is a primitive element of $GF(2^p)$. Then $c_0 = \bar{c}_0 + \phi_0 = \bar{f}(1) + \phi_0$ and $c_1 = \bar{f}(\omega) + \phi_1$, and they can be computed by simple hardware as shown in [4].

D. Simplified Chien-search-based codeword recovery

If the RS code is encoded systematically, only the symbols in the systematic positions, S, need to be recovered, and they can be done using (1). For binary BCH codes, $\bar{c}_i$ for $i \in S$ is either '0' or '1'. Hence, we only need to tell whether $\bar{c}_i$ is zero or nonzero for these positions, and the computation of $\bar{f}(\alpha_i)$ can be simplified accordingly. Through analyzing the $(x+\alpha_i)$ factors $q_0(x)$ and $q_1(x)$ can have and applying the L'Hôpital rule, $\bar{c}_i$ for $i \in S$ is computed using the following equations for BCH decoding [4].

$$\bar{c}_i = \bar{f}(\alpha_i) = \begin{cases} 0,\ i \in S/S_F \& q_1(\alpha_i) \neq 0 \\ 1,\ i \in S/S_F \& q_1(\alpha_i) = 0 \\ 1,\ i \in S \cap S_F \& q_0(\alpha_i) \neq 0 \\ 0,\ i \in S \cap S_F \& q_0(\alpha_i) = 0 \& q_1(\alpha_i) \neq 0 \\ 0,\ i \in S \cap S_F \& q_0'(\alpha_i) = 0 \& q_1(\alpha_i) = 0 \\ 1,\ i \in S \cap S_F \& q_0'(\alpha_i) \neq 0 \& q_1(\alpha_i) = 0 \end{cases} \tag{2}$$

When $\{\alpha_0, \alpha_1, \alpha_2, \dots\} = \{1, \omega, \omega^2, \dots\}$, $q_0(x)$ and $q_1(x)$ need to be evaluated over k consecutive finite field elements to compute $\bar{c}_i$ for $i \in S$. Hence, the evaluation values are computed using a Chien-search architecture that consists of simple constant multipliers and adders. Also the derivative of a polynomial over finite fields of characteristic two is the

TABLE II
HARDWARE COMPLEXITIES OF BCH DECODERS FOR A (4200, 4096) CODE OVER $GF(2^{13})$

	GF Mult.	GF Add.	GF Inv.	GF Squarer	Const. Mult.	Mux (bit)	Memory (bit)	Register (bit)	# of clks	Total equivalent gate count (XOR)
Systematic re-encoder	1	321	0	0	320	0	208	208	230	-
Interpolator	44	42	2	0	2	572	1768	832	228	-
Proposed polynomial selection	10	22	0	0	12	104	0	624	12	-
Proposed codeword recovery	0	323	0	0	320	208	0	949	217	-
Pipelining RAM	0	0	0	0	0	0	13010	0	-	-
Interpolation-based Chase decoder	55	708	2	0	654	884	14986	2613	230	77374
Chase decoder based on Berlekamp's [3]	17	2987	1	9	2976	33891	13146	5551	207	191891
GMD decoder based on Berlekamp's	51	2044	3	16	2000	18876	13666	4628	200	146903
Soft-decision decoder [9]	23	135	0	17	9	1209	106496	2288	69743	124361
HDD	2	545	0	8	544	117	8400	1183	128	34637

collection of the odd-degree terms. Hence, the Chien search over the even and odd terms of $q_0(x)$ is done separately so that $q_0'(\alpha_i)$ does not require extra computation. The detailed architecture for implementing (2) can be found in [4].

IV. BCH DECODER COMPLEXITY COMPARISONS

In this section, the complexity of BCH decoders are compared by using an 8-bit error-correcting (4200, 4096) BCH code constructed over $GF(2^{13})$ as an example. The BCH codewords are considered as codewords of an 8-symbol error-correcting (4200, 4184) RS code over $GF(2^{13})$ in order to apply the interpolation-based decoding.

The complexity of BCH decoders are summarized in Table II [4]. The critical path of each decoder is similar, and consists of a multiplier, an adder and a couple of multiplexors. On 40nm CMOS technology, they can reach a clock frequency of 500Mhz. To achieve higher throughput, the interpolation-based decoder is pipelined according to the cutsets shown by the dashed lines in Fig. 2. Accordingly, the interpolation-based Chase decoder with $\eta = 4$ can reach 10Gbps throughput with around 77k logic gates. In terms of throughput-over-area-ratio, it is 2.2 times more efficient than the Chase BCH decoder based on the Berlekamp's algorithm [3]. The major reason is that the polynomial selection is done by testing a few symbols in the interpolation-based decoder. However, choosing which error locator leads to correct codeword in the Chase decoder based on the Berlekamp's algorithm requires expensive highly-parallel Chien search.

The GMD decoder can be also implemented by a one-pass scheme based on the Berlekamp's algorithm. Despite the performance loss, its gate count is almost twice of the interpolation-based decoder. The soft-decision decoder in [9] has inferior performance and requires more logic gate than the interpolation-based decoder as well. More importantly, it needs to try 2^{2t} test vectors and the achievable throughput is 300 times lower. Compared to the HDD based on the Berlekamp's algorithm, the complexity of the interpolation-based decoder is 4 times higher. However, it can correct up to $t + \eta = 12$ instead of 8 errors as in HDD.

V. CONCLUSIONS

This paper summarized the most efficient architecture available for each step of the interpolation-based Chase BCH decoder. BCH codewords are considered as RS codewords in order to apply the interpolation-based decoding. The re-encoding and interpolation architectures developed for RS codes are directly borrowed for implementing BCH decoding. By making use of the binary property, low-complexity polynomial selection and simplified codeword recovery have been designed exclusively for binary BCH codes. Without losing coding gain, the interpolation-based Chase BCH decoder has much lower hardware complexity than that based on the Berlekamp's algorithm.

REFERENCES

[1] N. Kamiya, "On algebraic soft-decision decoding algorithms for BCH codes," *IEEE Trans. Info. Theory*, vol. 47, no. 1, pp. 45-58, Jan. 2001.

[2] Y. Wu, "Fast Chase decoding algorithms and architectures for Reed-Solomon codes," *IEEE Trans. Info. Theory*, vol. 58, no. 1, Jan. 2012.

[3] X. Zhang, J. Zhu and Y. Wu, "Efficient one-pass Chase soft-decision BCH decoder for multi-level cell NAND flash memory," *Proc. IEEE Intl. Midwest Symp. Circuits and Syst.*, Aug. 2011.

[4] X. Zhang, "An efficient interpolation-based BCH Chase decoder," *IEEE Trans. on Circuits and Syst.-II*, vol. 60, no. 4, pp. 212-216, Apr. 2013.

[5] 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.

[6] X. Zhang and Y. Zheng, "Systematically re-encoded algebraic soft-decision Reed-Solomon decoder," *IEEE Trans. on Circuits and Syst.-II*, vol. 59, no. 6, pp. 376-380, Jun. 2012.

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

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

[9] Y.-M. Lin, H.-C. Chang, and C.-Y. Lee, "An improved soft BCH decoder with one extra error compensation," *Proc. IEEE Intl. Symp. Circuits and Syst.*, pp. 3941-3944, May 2010.

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

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

[12] X. Zhang and Y. Zheng, "Efficient codeword recovery architecture for low-complexity Chase Reed-Solomon decoding," *Proc. Info. Theory and Appl. Workshop*, San Diego, CA, Feb. 2011.

[13] X. Zhang and K. K. Parhi, "High-speed architectures for parallel long BCH encoders" *IEEE Trans. on VLSI Syst.*, vol. 13, no. 7, pp. 872-877, Jul. 2005.

[14] R. Kötter, *On Algebraic Decoding of Algebraic-Geometric and Cyclic Codes*, Ph.D. dissertation, Dept. of Elec. Engr., Linköping University, Linköping, Sweden, 1996.