A Fully Parallel BCH Codec with Double Error Correcting Capability for NOR Flash Applications

Chia-Ching Chu, Yi-Min Lin, Chi-Heng Yang, and Hsie-Chia Chang

Department of Electronics Engineering and Institute of Electronics, National Chiao Tung University
Hsinchu, Taiwan 300, R.O.C.
E-mail: ccchu@oasis.ee.nctu.edu.tw

***Abstract*—A double error correcting (DEC) BCH codec is designed for NOR flash memory systems to improve reliability. Due to the latency constraint less than 10 ns, the fully parallel architecture with huge hardware cost is utilized to process both the encoding and decoding scheme within one clock cycle. Notice that encoder and decoder will not be activated simultaneously in NOR flash applications, so we combine the encoder and syndrome calculator based on the property of minimal polynomials in order to efficiently arrange silicon area. Furthermore, a new error location polynomial is developed to reduce the number of constant finite filed multipliers (CFFMs) in Chien search. According to 90 nm CMOS technology, our propose DEC BCH codec can achieve 2.5 ns latency with 41,705 μm^2 area.**

I. Introduction

NOR flash memories, one of the most popular non-volatile memories, are widely used in portable devices such as mobile phones and game consoles. As mobile appliances are prevailing in our daily life, the demand for high speed, high density, and low-cost NOR flash memories with the multi-level-cell (MLC) technique grows rapidly. Accordingly, the memory process is scaling down aggressively to achieve these requirements but damage the reliability dramatically. In NOR flash applications, the bit error rate (BER) should be less than 10^{-12}, but the BER is predicted to be higher than 10^{-6} in the advanced process technologies 45nm. As a result, error correction code (ECC) plays an important role to ensure adequate reliability [1]–[4].

In general, single error correcting (SEC) ECC such as Hamming code is extensively applied in NOR flash memories due to its simple architecture and low latency. For shrinking devices and increasing page size, SEC ECC becomes inefficient to compensate the high BER, as shown in Fig. 1. As a result, double error correcting (DEC) BCH codes are necessary; however, their iterative processing are not suitable for the latency-constrained memories. Thus, fully parallel architecture is proposed in this paper at the cost of increasing little area. By using matrix operations, a new method to combine encoder and syndrome calculator is developed. Additionally, a new error location polynomial is defined to reduce the degree of error location polynomial, so that the hardware cost in Chien search will be sufficiently reduced.

This paper is organized as follows. Section II gives the background of fully parallel BCH codecs. In Section III, a new encoding algorithm is developed for fully parallel architectures. The proposed error location polynomial for DEC BCH decoder is presented in Section IV. Based on the proposed methods, Section V demonstrates the implementation and comparison results. Finally, Section VI gives the conclusion.

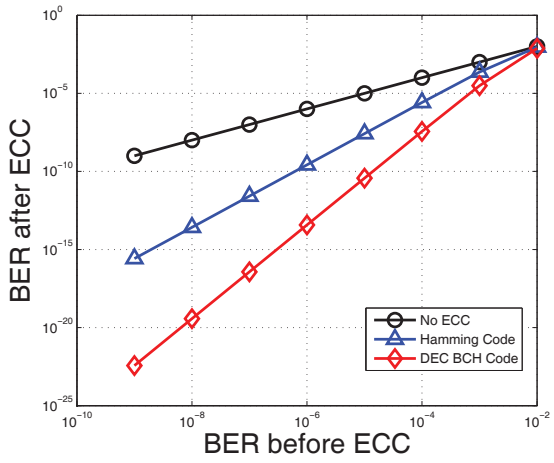

Fig. 1. Error correction capability of Hamming code and DEC BCH code with 256-bit data length [2].

II. Background

For fully parallel architecture, the BCH codec processes all bits simultaneously and therefore, the processing can be considered as a series of matrix operations. In this section, several matrix operations used in this paper are introduced.

A. Constant Finite Field Multiplier

Each arbitrary element over $GF(2^m)$ can be presented as $\lambda = \sum_{i=0}^{m-1} \lambda_i \alpha^i$ with the binary coordinate λ_i and the basis $\{\alpha^0, \alpha^1, \cdots, \alpha^{m-1}\}$ [5]. Hence, the multiplication between

two arbitrary symbols, $\alpha^j \times \lambda$, can be expressed as

$$\begin{aligned}\alpha^j \times \lambda &= \alpha^j \times (\lambda_0 + \lambda_1\alpha + \cdots + \lambda_{m-1}\alpha^{m-1}) \\ &= \begin{bmatrix} \alpha_0^j & \alpha_0^{j+1} & \cdots & \alpha_0^{j+m-1} \\ \alpha_1^j & \alpha_1^{j+1} & \cdots & \alpha_1^{j+m-1} \\ \vdots & & \ddots & \vdots \\ \alpha_{m-1}^j & \alpha_{m-1}^{j+1} & \cdots & \alpha_{m-1}^{j+m-1} \end{bmatrix} \begin{bmatrix} \lambda_0 \\ \lambda_1 \\ \vdots \\ \lambda_{m-1} \end{bmatrix} \quad (1) \\ &= \mathbf{C}^j\lambda,\end{aligned}$$

where the binary element α_l^p stands for the l-th coordinate of α^p and $\mathbf{C}^j$ is considered as a multiplicator matrix of constant finite field multiplier (CFFM) with the constant α^j.

B. Finite Field Square

Each arbitrary element, λ, over $GF(2^m)$ can be considered as a polynomial with binary coefficients, so the calculation of finite field square can be defined as

$$\begin{aligned}\lambda^2 &= (\lambda_0 + \lambda_1\alpha + \cdots + \lambda_{m-1}\alpha^{m-1})^2 \\ &= \lambda_0 + \lambda_1\alpha^2 + \cdots + \lambda_{m-1}\alpha^{2(m-1)} \\ &= (\lambda_0 + \lambda_1 x + \cdots + \lambda_{m-1}x^{m-1}) \mid_{x=\alpha^2} \\ &= \begin{bmatrix} \alpha_0^0 & \alpha_0^2 & \cdots & \alpha_0^{2(m-1)} \\ \alpha_1^0 & \alpha_1^2 & \cdots & \alpha_1^{2(m-1)} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_{m-1}^0 & \alpha_{m-1}^2 & \cdots & \alpha_{m-1}^{2(m-1)} \end{bmatrix} \begin{bmatrix} \lambda_0 \\ \lambda_1 \\ \vdots \\ \lambda_{m-1} \end{bmatrix} \\ &= \mathbf{Q}\lambda \quad (2)\end{aligned}$$

From (2), for each arbitrary polynomial with binary coefficients $f(x)$, the computation of $f(x^2)$ can be derived from $f^2(x)$.

III. Combined Encoder and Syndrome Calculator

An $(n, k; t)$ BCH code over $GF(2^m)$ with codeword length of n bits and message length of k bits can correct up to t error bits. As shown in Fig. 2, the conventional BCH decoding contains three major blocks : *syndrome calculator, key equation solver and Chien search*. In this section, the conventional BCH encoder and syndrome calculator are introduced. Then, a new encoding algorithm is presented.

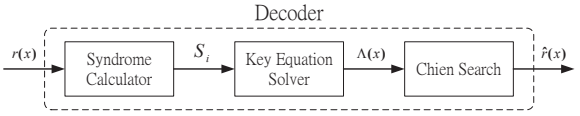

Fig. 2. BCH Decoder Block Diagram.

A. Conventional BCH Encoding Algorithm

A message sequence can be considered as a polynomial $u(x) = u_{k-1}x^{k-1} + u_{k-2}x^{k-2} + \cdots + u_0$ and a generator polynomial $g(x)$ of degree $(n-k)$ is defined as

$$\begin{aligned}g(x) &= LCM\{M_1(x), M_2(x), \cdots, M_{2t}(x)\} \\ &= M_1(x) \times M_3(x) \times \cdots \times M_{2t-1}(x) \\ &= g_{n-k}x^{n-k} + \cdots + g_2x^2 + g_1x + g_0, \quad (3)\end{aligned}$$

where $M_i(x)$ is the minimal polynomial of α^i with binary coefficients. The systematic encoding

$$u(x)x^{n-k} = q(x)g(x) + p(x) \quad (4)$$

can provide a codeword polynomial $c(x) = u(x)x^{n-k} + p(x)$, where the parity polynomial $p(x)$ with binary coefficients can be expressed as

$$\begin{aligned}p(x) &= p_{n-k-1}x^{n-k-1} + p_{n-k-2}x^{n-k-2} + \cdots + p_0 \\ &= (u_{k-1}x^{n-1} + u_{k-2}x^{n-2} + \cdots + u_0x^{n-k}) \textbf{ mod } g(x) \\ &= u_{k-1}x^{n-1} \textbf{ mod } g(x) + u_{k-2}x^{n-2} \textbf{ mod } g(x) + \cdots \\ &\quad + u_0x^{n-k} \textbf{ mod } g(x) \quad (5)\end{aligned}$$

The coefficients of $p(x)$ also can be rewritten as following matrix:

$$\begin{aligned}\begin{bmatrix} p_0 \\ p_1 \\ \vdots \\ p_{n-k-1} \end{bmatrix} &= \begin{bmatrix} d_{n-1,0} & d_{n-2,0} & \cdots & d_{n-k,0} \\ d_{n-1,1} & d_{n-2,1} & \cdots & d_{n-k,1} \\ \vdots & \vdots & \ddots & \vdots \\ d_{n-1,n-k-1} & d_{n-2,n-k-1} & \cdots & d_{n-k,n-k-1} \end{bmatrix} \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{k-1} \end{bmatrix} \\ &= \mathbf{D} \cdot \mathbf{U} \quad (6)\end{aligned}$$

where $d_{i,0} + d_{i,1}x + \cdots + d_{i,n-k-1}x^{n-k-1}$ denotes x^i **mod** $g(x)$ and $i = n-1, n-2, \cdots, n-k$. From (6), a fully parallel BCH encoder can process n bits simultaneously.

B. Syndrome Calculator

For BCH decoding algorithms, the syndrome calculator firstly calculates the syndrome values $S_i = r(x) \mid_{x=\alpha^i}$, where $i = 1, 3, \cdots, 2t-1$ and $r(x)$ is a received codeword polynomial. Since the multi-cycle latency is not tolerable for NOR flash memories, the syndrome calculator should process n bits simultaneously to calculate the syndrome value, Notice that the syndrome calculator can be considered as the multiplication of the receive codeword and the parity check matrix $\mathbf{H}_{\text{SYN}}$, which defined in (7), in fully parallel architecture.

$$\begin{aligned}\begin{bmatrix} S_1 \\ S_3 \\ \vdots \\ S_{2t-1} \end{bmatrix} &= \begin{bmatrix} 1 & \alpha & \alpha^2 & \cdots & \alpha^{n-1} \\ 1 & \alpha^3 & \alpha^6 & \cdots & \alpha^{3(n-1)} \\ \vdots & & & \ddots & \vdots \\ 1 & \alpha^{2t-1} & \alpha^{2(2t-1)} & \cdots & \alpha^{(2t-1)(n-1)} \end{bmatrix} \begin{bmatrix} r_0 \\ r_1 \\ \vdots \\ r_{n-1} \end{bmatrix} \\ &= \mathbf{H}_{\text{SYN}} \cdot \mathbf{r} \quad (7)\end{aligned}$$

$$\begin{aligned}S_i &= r_0 + r_1\alpha^i + r_2\alpha^{2i} + \cdots + r_{n-1}\alpha^{i(n-1)} \\ &= \begin{bmatrix} 1 & \alpha_0^i & \alpha_0^{2i} & \cdots & \alpha_0^{i(n-1)} \\ 0 & \alpha_1^i & \alpha_1^{2i} & \cdots & \alpha_1^{i(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \alpha_{m-1}^i & \alpha_{m-1}^{2i} & \cdots & \alpha_{m-1}^{i(n-1)} \end{bmatrix} \begin{bmatrix} r_0 \\ r_1 \\ \vdots \\ r_{n-1} \end{bmatrix}\end{aligned}$$

C. Proposed Encoding Algorithm

According to (3), we can find that $\alpha, \alpha^3, \cdots, \alpha^{2t-1}$ and their conjugates are the roots of $g(x)$. Taking all the $n-k$ roots into (4), we can obtain $n-k$ equations, and express as

$$p(x) = u(x)x^{n-k} \mid_{x=\alpha^{\text{i}} \text{ and their conjugates}}, \quad (8)$$

where $i = 1, 3, \cdots, 2t-1$. From the $n-k$ equations in (8), which can support us to find the coefficients of parity

polynomial $p(x)$, can be also rewritten as : $\mathbf{BP} = \mathbf{AH}_{\text{EN}}\mathbf{U}$, where

$$\mathbf{B} = \begin{bmatrix} 1 & \alpha & \cdots & \alpha^{n-k-1} \\ 1 & \alpha^2 & \cdots & \alpha^{2\times(n-k-1)} \\ \vdots & & \ddots & \vdots \\ 1 & \alpha^{(2t-1)\times l_{2t-1}} & \cdots & \alpha^{(2t-1)\times l_{2t-1}\times(n-k-1)} \end{bmatrix}$$

$$\mathbf{A} = \begin{bmatrix} \alpha^{n-k} & 0 & \cdots & 0 \\ 0 & \alpha^{2(n-k)} & \cdots & 0 \\ \vdots & \ddots & \vdots & \\ 0 & \cdots & 0 & \alpha^{(2t-1)\times l_{2t-1}\times(n-k)} \end{bmatrix}$$

$$\mathbf{H}_{\text{EN}} = \begin{bmatrix} 1 & \alpha & \cdots & \alpha^{k-1} \\ 1 & \alpha^2 & \cdots & \alpha^{2(k-1)} \\ \vdots & & \ddots & \vdots \\ 1 & \alpha^{(2t-1)\times l_{2t-1}} & \cdots & \alpha^{(2t-1)\times l_{2t-1}\times(k-1)} \end{bmatrix}$$

$$\mathbf{U} = \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{k-1} \end{bmatrix}, \quad \mathbf{P} = \begin{bmatrix} p_0 \\ p_1 \\ \vdots \\ p_{n-k-1} \end{bmatrix}. \tag{9}$$

Note that l_i in (9) denote the number of α^i's conjugates.

In addition, because $g(x) = M_1(x) \times M_3(x) \times \cdots \times M_{2t-1}(x)$, the roots of $g(x)$ can be divided into t groups based on minimal polynomials. For example, the roots of $M_1(x)$, $\{\alpha, \alpha^2, \cdots, \alpha^{2^{m-1}}\}$, is one of the t groups. Hence, we choose one root from each group to take into $u(x)$ instead of all roots of $g(x)$. In this paper, we choose $\alpha, \alpha^3, \cdots, \alpha^{2t-1}$ to obtain $u(\alpha), u(\alpha^3), \cdots, u(\alpha^{2t-1})$. With the property in (2), we can derive $u(x) \mid_{x=\alpha^{i\times 2^j}}$ from $u(\alpha^i)$ multiply by $\mathbf{Q}^j$ where $j = 1, 2, \cdots, l_i$. Note that l_i is the value of the number of α^i's conjugates and $\mathbf{Q}^j$ denotes matrix $\mathbf{Q}$ to the power degree of j. Therefore $\mathbf{H}_{\text{EN}}\mathbf{U}$ can be factorized as $\mathbf{H}_{\text{EN}}\mathbf{U} = \hat{\mathbf{Q}}\mathbf{H'}_{\text{EN}}\mathbf{U}$, where $\hat{\mathbf{Q}}$ is defined by (10) and $\mathbf{H'}_{\text{EN}}$ is defined by (11).

$$\hat{\mathbf{Q}} = \begin{bmatrix} \mathbf{Q'} & 0 & \cdots & 0 \\ 0 & \mathbf{Q'} & \cdots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & \cdots & 0 & \mathbf{Q'} \end{bmatrix}, \mathbf{Q'} = \begin{bmatrix} \mathbf{Q}^0 \\ \mathbf{Q}^1 \\ \vdots \\ \mathbf{Q}^{l_i} \end{bmatrix} \tag{10}$$

$$\mathbf{H'}_{\text{EN}} = \begin{bmatrix} 1 & \alpha & \alpha^2 & \cdots & \alpha^{k-1} \\ 1 & \alpha^3 & \alpha^6 & \cdots & \alpha^{3(k-1)} \\ \vdots & & & \ddots & \vdots \\ 1 & \alpha^{2t-1} & \alpha^{2(2t-1)} & \cdots & \alpha^{(2t-1)(k-1)} \end{bmatrix} \tag{11}$$

Note that $\mathbf{Q}^0$ is an identity matrix and $\mathbf{H'}_{\text{EN}}$, a sub-matrix of parity check matrix, can be shared with syndrome calculator. Consequently, the equation $\mathbf{BP} = \mathbf{AH}_{\text{EN}}\mathbf{U}$ can be reformulated as $\mathbf{BP} = \mathbf{A}\hat{\mathbf{Q}}\mathbf{H'}_{\text{EN}}\mathbf{U}$. In addition, the matrix $\mathbf{B}$ is non-singular, so we can drive

$$\mathbf{P} = \mathbf{B}^{-1}\mathbf{A}\,\hat{\mathbf{Q}}\,\mathbf{H'}_{\text{EN}}\,\mathbf{U} = \mathbf{E}\,\mathbf{H'}_{\text{EN}}\,\mathbf{U} \tag{12}$$

to get parity bits, where $\mathbf{B}^{-1}$ is the inverse matrix of $\mathbf{B}$ and matrix $\mathbf{E}$ is equal to $\mathbf{B}^{-1}\mathbf{A}\hat{\mathbf{Q}}$.

TABLE I
SYNTHESIS RESULTS FOR JOINT ENCODER AND SYNDROME CALCULATOR IN 90 NM CMOS TECHNOLOGY

(n, k, t)	This work (Gate count)	*Conventional method (Gate count)	Area Saving
(274, 256, 2)	2671	4621	42%
(283, 256, 3)	4100	6904	41%
(532, 512, 2)	5105	9234	45%
(542, 512, 3)	7745	13707	43%

*An extra syndrome calculator is required to support syndrome values evaluation.

**The clock period in this table is 2.5 ns.

Therefore, the message matrix $\mathbf{U}$ is firstly multiply by matrix $\mathbf{H'}_{\text{EN}}$. Then the product of matrix $\mathbf{E}$ and $\mathbf{H'}_{\text{EN}}\mathbf{U}$ can support us to get the parity bits. Notice that we can regard 0 or 1 as a m-bits symbol over $GF(2^m)$, $0 = [0, 0, \cdots, 0]^T$ and $1 = [1, 0, \cdots, 0]^T$. And the elements in the result of $\mathbf{E}$ times $\mathbf{H'}_{\text{EN}}\mathbf{U}$ are $0 = [0, 0, \cdots, 0]^T$ or $1 = [1, 0, \cdots, 0]^T$ since the parity bits are binary. However, the elements of parity bits matrix $\mathbf{P}$ are only one bit, instead of (1), each CFFM in matrix $\mathbf{E}$ can be reduced to one row as (13).

$$\begin{aligned} \alpha^j \times \lambda &= \alpha^j \times (\lambda_0 + \lambda_1\alpha + \cdots + \lambda_{m-1}\alpha^{m-1}) \\ &= \begin{bmatrix} \alpha_0^j & \alpha_0^{j+1} & \cdots & \alpha_0^{j+m-1} \end{bmatrix} \begin{bmatrix} \lambda_0 \\ \lambda_1 \\ \vdots \\ \lambda_{m-1} \end{bmatrix} \\ &= \mathbf{C}_{\mathbf{E}}^j\lambda \end{aligned} \tag{13}$$

where $\mathbf{C}_{\mathbf{E}}^j$ is called a CFFM with α^j as the constant multiplicator in the matrix $\mathbf{E}$. After the simplification and sharing the parity check matrix, a $n-k$ by $n-k$ matrix is used to encode the information bits instead of using $\mathbf{D}$, which is a k by $n-k$ matrix.

The conventional BCH encoder in fully parallel architecture is implemented by the matrix $\mathbf{D}$ in (6), which is used to find the result of $u(x)x^{n-k}$ **mod** $g(x)$. Apparently, the matrix $\mathbf{D}$ is totally different from the parity check matrix $\mathbf{H}_{\text{SYN}}$, which generate the syndrome value $S_i = r(x) \mid_{x=\alpha^i}$, so that encoder can not be shared with syndrome calculator. In this section, we develop a new encoding method for BCH codes in fully parallel architecture in order to combine encoder and syndrome calculator. As listed in Table I, the hardware cost of encoder can be significantly reduced.

IV. LOW COMPLEXITY BCH DECODER IN FULLY PARALLEL ARCHITECTURE

After syndrome calculator provides the syndrome values, the error location polynomial can be acquired by key equation solver. Then Chien search can support us to find the error location. Note that the hardware complexity in Chien search, which dominates the hardware complexity of decoder, depends on the number of CFFMs and dramatically increases with the degree of error location polynomial. In order to efficiently arrange silicon area, this section provides a new method to reduce the degree of error location polynomial.

A. Conventional Error Location Polynomial

With the syndrome values, the error location polynomial $\Lambda(x)$ can be obtained by key equation solver. To eliminate the time-consuming iterations, Peterson's algorithm is proposed in [6] instead of using iterative Berlekamp-Massey(BM) algorithm. Notice that Peterson's algorithm is not suitable for the large error correcting capability. For DEC BCH codes, Peterson's algorithm provides the error location polynomial

$$\Lambda(x) \quad = \quad 1 + S_1 x + (S_1^2 + \frac{S_3}{S_1})x^2 \tag{14}$$

However, complicated division still exists in (14); thus, a inversion-less error location polynomial $\Lambda'(x)$ provided by [7] is adopted by [2].

$$\Lambda'(x) \quad = \quad S_1 + S_1^2 x + (S_1^3 + S_3)x^2 \tag{15}$$

B. Proposed Error Location Polynomial

Instead of using (15), the *reverse error location polynomial* (RELP) is proposed. For a DEC BCH decoder, the RELP $\widetilde{\Lambda}(x) = (x + S_1)^3 + x^3 + S_3$ is given by [8] [9]. Because it is too complicate to compute $(x + S_1)^3$ in $\widetilde{\Lambda}(x)$, we can reformulate $\widetilde{\Lambda}(x)$ as

$$\begin{aligned} \widetilde{\Lambda}(x) &= (x + S_1)^3 + x^3 + S_3 \\ &= x^3 + S_1 x^2 + S_1^2 x + S_1^3 + x^3 + S_3 \\ &= S_1 x^2 + S_1^2 x + (S_1^3 + S_3) \end{aligned} \tag{16}$$

Note that $x = [x_0, x_1, \cdots, x_{m-1}]^T$ can be presented as $x_0 + x_1\alpha + \cdots + x_{m-1}\alpha^{m-1}$, (16) can be expressed as

$$\begin{aligned} \widetilde{\Lambda}'(x) &= S_1(x_0 + x_1\alpha^2 + \cdots + x_{m-1}\alpha^{2(m-1)}) \\ &+ S_1^2(x_0 + x_1\alpha + \cdots + x_{m-1}\alpha^{m-1}) + S_1^3 + S_3 \\ &= \mathbf{T} \cdot [x_0,\ x_1,\ \cdots,\ x_{m-1}]^T + S_1^3 + S_3 \end{aligned} \tag{17}$$

where matrix $\mathbf{T}$ is defined as

$$\mathbf{T} \ = \ [S_1 + S_1^2,\ \ S_1\alpha^2 + S_1^2\alpha,\ \ \cdots,\ \ S_1\alpha^{2(m-1)} + S_1^2\alpha^{m-1}]$$

With (1) and (2), matrix $\mathbf{T}$ can be expressed as

$$\begin{aligned} \mathbf{T} &= \left[(\mathbf{C}^0 + \mathbf{Q})S_1,\ (\mathbf{C}^2 + \mathbf{C}^1\mathbf{Q})S_1, \cdots, (\mathbf{C}^{2(m-1)} + \mathbf{C}^{m-1}\mathbf{Q})S_1\right] \\ &= \left[\mathbf{K}_0 \cdot S_1,\quad \mathbf{K}_1 \cdot S_1,\quad \cdots,\quad \mathbf{K}_{m-1} \cdot S_1\right] \end{aligned}$$

Apparently, the dimension of matrix $\mathbf{K}_i$ ($i = 1 \sim m-1$) is the same as that of CFFMs. Therefore, the hardware cost we need to construct matrix $\mathbf{T}$ can be regarded as the cost of m CFFMs.

Finally, *Chien search* is performed to find out the error locations by substituting $\alpha^0, \alpha^1, \cdots$ and α^{n-1} into (17). If α^i is the root of (17), the i-th bit is erroneous. For the decoding latency constraint, all the substitutions are applied in parallel. Note that the number of CFFMs we need to applied to *Chien search* is $2n$ when (15) is employed. However, we can reduce the number of CFFMs to $n + m$ by using our proposed error location polynomial.

V. Experiment Results and Comparison

This section demonstrates the synthesis results for a DEC BCH codec with our proposed architecture. For the low reliability NOR flash memories, our design provides double error correcting capacity to protect 256 bits information. TABLE II demonstrates the synthesis results and comparison results between proposed design and the design in [2]. Base on 90 nm CMOS technology, our design is 14,789 gate-counts. Moreover, the latency is 2.5 ns, which can fit well for latency-constrained flash memory systems.

TABLE II
Synthesis Results of Proposed DEC BCH Codec

	Proposed Work	CICC'09 [2]
Component	Encoder + Decoder	Decoder
Technology	90 nm	0.18 μm
Cycle	1	1
Clock period (Synthesis)	2.5 ns	4.51 ns
Data size (bits)	256	256
Throughput	102.4Gb/s	56.76Gb/s
Area (μm^2)	41,705	283,512
Gate-count	14,789	N/A

VI. Conclusion

In this paper, a fully parallel DEC BCH codec for NOR flash memories is presented. Based on our proposed encoding method, the combined encoder and syndrome calculator architecture is introduced to reduce the hardware cost. Moreover, the number of CFFMs, which dominate the hardware complexity of fully parallel Chien search unit, is significantly decreased from $2n$ to $n + m$. After implemented in 90 nm CMOS technology, the proposed fully parallel (274,256;2) BCH codec can achieve 2.5 ns latency with 41,705 μm^2 silicon area.

References

[1] S. Gregori, A. Cabrini, O. Khouri, and G. Torelli, "On-chip error correcting techniques for new-generation flash memories," *Proceedings of the IEEE*, vol. 91, no. 4, pp. 602–616, Apr. 2003.

[2] X. Wang, D. Wu, C. Hu, L. Pan, and R. Zhou, "Emdedded high-speed BCH decoder for new-generation NOR flash memories," in *IEEE Custom Integrated Circuits Conference*, Sep. 2009, pp. 195–198.

[3] P. Ankolekar, R. Isaac, and J. Bredow, "Multibit error-correction methods for latency-constrained flash memory systems," *IEEE Transactions on Device and Materials Reliability*, vol. 10, no. 1, pp. 33–39, Mar. 2010.

[4] R. Naseer and J. Draper, "DEC ECC design to improve memory reliability in sub-100nm technologies," in *IEEE International Conference on Electronics, Circuits and Systems*, Sep. 2008, pp. 586 –589.

[5] Y.-M. Lin, C.-H. Yang, C.-H. Hsu, H.-C. Chang, and C.-Y. Lee, "A MPCN-based parallel architecture in BCH decoders for NAND flash memories," *IEEE Transactions on Circuits and Systems II*, vol. 58, no. 9, pp. 1–5, Sep. 2011.

[6] W. W. Peterson and J. E. J. Weldon, *Error-Corecting Code*, 2nd ed. The MIT Press, 1972.

[7] H. Burton, "Inversionless decoding of binary BCH codes," *IEEE Transactions on Information Theory*, vol. 17, no. 4, pp. 464 – 466, Jul. 1971.

[8] G. Davida and J. Cowles, "A new error-locating polynomial for decoding of BCH codes," *IEEE Transactions on Information Theory*, vol. 21, pp. 235–236, Mar. 1975.

[9] H.-C. Chang and C. Shung, "New serial architecture for the Berlekamp-Massey algorithm," *IEEE Transactions on Communications*, vol. 47, no. 4, Apr. 1999.