Low latency BCH decoder using the affine polynomial over the finite field

Pham Khac Hoan[1*], Nguyen Tien Thai[1], Vu Son Ha[2]

[1]Military Technical Academy;
[2]Academy of Military Science and Technology.
[*]Corresponding author: hoanpk2012@gmail.com
Received 10 Sep 2022; Revised 25 Nov 2022; Accepted 15 Dec 2022; Published 30 Dec 2022.
DOI: https://doi.org/10.54939/1859-1043.j.mst.CSCE6.2022.105-113

ABSTRACT

The paper proposes a low latency BCH decoder with low complexity using parallel computation and simplifying locating errors by finding the roots of the affine polynomial over finite fields. The proposed design can be implemented on low-cost hardware platforms while applicable in very low latency information systems.

Keywords: Error-correcting code; Finite field; Affine polynomial; BCH code.

1. INTRODUCTION

Several problems are associated with solving equations over finite fields, such as error correcting code that requires the key equation when decoding BCH, Reed-Solomon, Goppa code, and decryption cryptosystems based on codes. Berlekamp is one of the authors who has significantly contributed to solving the problem of factorization of polynomial in finite fields [1].

Some indirect methods to solve equations over finite fields include: implementing iterative algorithms such as Chien procedures and using Fourier transform in the Galois field. However, these methods are often associated with relatively sizeable computational delay and high complexity [2-4].

The BCH code is widely applied in digital information transmission, storage, and processing systems such as optical communication systems, digital television, WBAN information, flash memory protection, etc. The classical BCH coding methods use the Belekamp-Massey algorithm to solve the key equation and the Chien procedure to find the root of the error locator polynomial by checking all the elements of the field [5-7]. However, these methods have a significant processing delay and implementation complexity, especially when a large field size. In some cases, when the number of errors to be corrected is not too large, it is possible to use the direct method thanks to Peterson's algorithm to calculate the error locator polynomial, thereby reducing the implementation complexity of the decoder [4, 8].

Linearized polynomials and affine polynomials have several remarkable properties that allow them to simplify finding roots. However, only a tiny number of polynomials can be linearized and affine polynomials. Based on algebraic transformations and finding the affine polynomial is a multiple of a given polynomial, roots of this polynomial can be found among the roots of the affine polynomial [2, 8].

This paper investigates the method of decoding BCH based on combining the Peterson algorithm to calculate the error locator polynomial coefficient and finding the root of the error locator polynomial by finding the root of the affine polynomial, which is a multiple of the polynomial whose roots need be found. The results obtained allow application in different cases and can be extended to fields with arbitrary sizes. The

proposed method reduces the complexity and processing delay of the decoder so that it can be applied in the design of BCH encoder and decoder for low-latency information systems such as WBAN, significant capacity memories, and optical communication networks...The rest of the paper is organized as follows. Section II outlines the basics of BCH codes and proposes a method of BCH decoding using Peterson's direct method for finding error locator polynomials. Section III studies a particular class of polynomials that are linearized in finite fields and proposes a 3-error correction decoding method based on finding the roots of the affine polynomial. Section 4 presents comparisons and analyses of the complexity and processing delay of the proposed method. Finally, some conclusions are drawn.

2. OVERVIEW Of BCH CODES AND A DIRECT DECODING OF BCH CODES

2.1. BCH code and decoding algorithms

Let us consider a primitive binary BCH code in the narrow sense *t*-error-correcting with length code $n = 2^m - 1$.

**Algebraic methods for decoding BCH*

Given $r(x) = v(x) + e(x)$ representing the polynomial from the received codeword, the error polynomial is defined as:

$$e(x) = e_{j_1} x^{j_1} + e_{j_2} x^{j_2} + \cdots + e_{j_v} x^{j_v} \tag{1}$$

where $v \le t$ is the number of errors. The tuples $e_{j_1}, e_{j_2}, \cdots, e_{j_v}$ and $\alpha^{j_1}, \alpha^{j_2}, \cdots, \alpha^{j_v}$ are the error values and the error locators, $e_j \in \{0, 1\}$ for the binary BCH code and $\alpha \in$ GF(2^m).

The syndrome is the r(x) value at each zero of the code:

$$\begin{aligned} s_1 &= r(\alpha) = e_{j_1}\alpha^{j_1} + \cdots + e_{j_v} x^{j_v} \\ s_2 &= r(\alpha^2) = e_{j_1}\alpha^{2j_1} + \cdots + e_{j_v} x^{2j_v} \\ &\cdots \quad \cdots \quad \cdots \quad \cdots \quad \cdots \quad \cdots \quad \cdots \quad \cdots \\ s_{2t} &= r(\alpha^{2t}) = e_{j_1}\alpha^{(2t)j_1} + \cdots + e_{j_v} x^{(2t)j_v} \end{aligned} \tag{2}$$

The error locator polynomial is defined as:

$$\sigma(x) = \prod_{\ell=1}^{v} \left(1 + \alpha^{j_\ell}\right) = 1 + \sigma_1 x + \sigma_2 x^2 + \cdots + \sigma_v x^v \tag{3}$$

With the roots of the equation being the inverse of the error locations. The relationship between the coefficients of and the syndrome [3-5]:

$$\begin{pmatrix} s_{v+1} \\ s_{v+2} \\ \cdots \\ s_{2v} \end{pmatrix} = \begin{pmatrix} s_1 & s_2 & \cdots & s_{v+1} \\ s_2 & s_3 & \cdots & s_{v+2} \\ \cdots & \cdots & \cdots & \cdots \\ s_v & s_{v+1} & \cdots & s_{2v-1} \end{pmatrix} \begin{pmatrix} \sigma_v \\ \sigma_{v-1} \\ \cdots \\ \sigma_1 \end{pmatrix} \tag{4}$$

Solving the key equation (4) is the most challenging problem in the process of BCH decoder. The popular methods for solving key equations are:

+ Berlekamp-Massey algorithm (BMA).

+ Euclid algorithm (EA).

The Berlekamp – Massey algorithm (BMA) is efficient for solving key equations in GF(2^m). BMA is considered the least hardware complexity algorithm, using an iterative procedure to construct a minimum-length linear shift feedback register structure, forming a syndrome sequence $s_1, s_2, \cdots, s_{2t}$.

The Euclidean Algorithm (EA) algorithm is widely used in the hardware implementation of BCH decoders based on a recursive procedure to find the greatest common divisor between two polynomials.

**Chien search and correct errors:*

To find the roots of $\sigma(x)$ a trial-and-error procedure is known as a Chien search. All non-zero elements β of GF(2^m) in series *1*, α, α^2, ... are tried and conditionally tested $\sigma(\beta^{-1}) = 0.$

Thus, algebraic methods for decoding BCH require solving high-order key equations over the Galois field. Iterative algorithms BMA, EA, and Chien search procedure have considerable processing delay when *n* and *t* are large, which limits the application of BCH code to low-latency information systems.

2.2. A direct method to decode BCH codes

In case the number of errors is not too large, Peterson's algorithm can be used to determine the error locator polynomial coefficient as follows:

When correcting two errors:

$$\sigma_0 = S_1, \quad \sigma_1 = S_1^2, \quad \sigma_2 = S_3 + S_1^3 \tag{5}$$

When correcting for three errors to avoid inversion, all coefficients are multiplied by $S_2 + S_1^3$:

$$\sigma_0 = S_3 + S_1^3, \quad \sigma_1 = S_1 S_3 + S_1^4, \quad \sigma_2 = S_1^2 S_3 + S_5, \quad \sigma_3 = S_1^3 S_3 + (S_3 + S_1^3)^2 + S_1 S_5 \tag{6}$$

Below is an in-depth analysis and design of a binary BCH decoder that corrects three errors. Algorithm 1 decodes the BCH code and corrects three errors as follows.

Algorithm 1.

Step 1: Calculate the syndrome $S = (S_1, S_3, S_5)$

Step 2: Find the error locator polynomial

- If $S_1 = S_3 = S_5 = 0$, there are no errors.

- If $D_2 = S_3 + S_1^3 = 0$ and $S_5 + S_1^5 = 0$, there is a single error.

- If $S_3 + S_1^3 \neq 0$ or $S_5 + S_1^5 \neq 0$ and $A = S_1^3 S_3 + (S_3 + S_1^3)^2 + S_1 S_5 = 0$, there are two errors.

- If $S_3 + S_1^3 \neq 0$, $S_5 + S_1^5 \neq 0$, and $A = S_1^3 S_3 + (S_3 + S_1^3)^2 + S_1 S_5 \neq 0$, there are three errors.

Step 3: Find the root of the error locator polynomial

- For single error, the error locator corresponds $n - \log S_1$.

- For two errors, error locator are inverses of roots of quadratic equations:

$$\sigma(x) = \sigma_0 + \sigma_1 x + \sigma_2 x^2 = 0,$$

where the coefficients are determined according to (5).

- For three errors, error locators are inverses of solutions of the equation:

$$\sigma(x) = \sigma_0 + \sigma_1 x + \sigma_2 x^2 + \sigma_3 x^3 = 0,$$

where the coefficients are determined according to (6).

To avoid confusion with other cases, when there are three distinct roots, the error locator polynomial has the form: $Ax^3 + Bx^2 + Cx + D,$ where $A = S_1^3 S_3 + (S_3 + S_1^3)^2 + S_1 S_5,\ B = S_1^2 S_3 + S_5,\ C = S_3 + S_1^3,\ D = S_1.$

The pipeline circuit that determines the coefficients is described in figure 1.

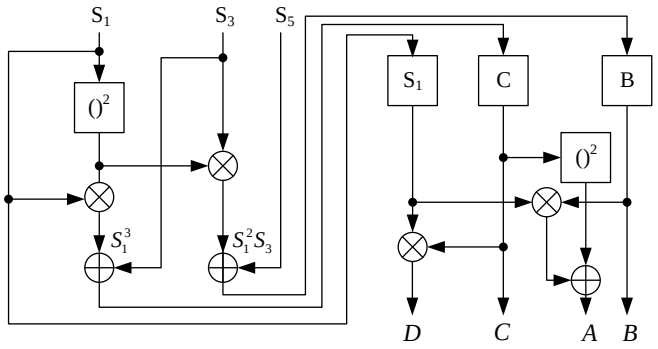

Figure 1. *Circuit for calculating the coefficient of error locator polynomial.*

- Step 4: Correct errors according to the error locations from the error locator.

3. DECODING THREE-ERROR BCH CODE BASED ON THE ROOT OF THE AFFINE POLYNOMIAL

3.1. Construct an affine polynomial that is a multiple of the error locator polynomial

In this section, only the case of three errors is considered (the key equation has three different roots). When $A \neq 0$ we can find multiple affine polynomials of the form:

$$A(z) = (Az^3 + Bz^2 + Cz + D)(Az + B) = A^2 z^4 + (AC + B^2) z^2 + (BC + AD) z + BD \quad (7)$$

The roots of the equation $A(z) = 0$ are the roots of the equation:

$$L(z) = A^2 z^4 + (AC + B^2) z^2 + (BC + AD) z = BD \quad (8)$$

In the following, we consider the method of finding roots of affine polynomials and linearized polynomials.

A polynomial $L(z),$ over $\mathrm{GF}(p^m),$ is said to be a linearized polynomial if

$$L(z) = \sum_{i=0}^{s} L_i z^{p^i}, \quad L_i \in GF(2^m). \quad (9)$$

Notation $\alpha^0, \alpha^1, ..., \alpha^{m-1}$ is the polynomial basis of the field $\mathrm{GF}(p^m)$, and we have the following proposition.

Proposition [2]

Let $L(z)$ be a linearized polynomial in $\mathrm{GF}(p^m)$, and the element z is represented $z=\sum_k Z_k\alpha^k,\ \ Z_k\in \mathrm{GF}(p)$ then:

$$L(z)=\sum_k Z_k L(\alpha^k) \tag{10}$$

When using the polynomial representation of the elements $L(\alpha^i)=\sum_{j=0}^{m-1} C_{i,j}\alpha^j$. Thus, the polynomial expansion coefficient $L(z)$ on the polynomial basis is calculated as follows:

$$\left[L_0,L_1,...,L_{n-1}\right]=\left[Z_0,Z_1,...,Z_{n-1}\right].C, \tag{11}$$

in which

$$C=\begin{bmatrix} C_{0,0} & C_{0,1} & C_{0,1} & . & . & . & C_{0,n-1} \\ C_{1,0} & C_{1,1} & C_{1,2} & . & . & . & C_{1,n-1} \\ . & . & . & . & . & . & . \\ C_{n-1,0} & C_{n-1,1} & C_{n-1,2} & . & . & . & C_{n-1,n-1} \end{bmatrix} \tag{12}$$

Example 1

Finding roots of the polynomial $f(z)=z^3+\alpha^{13}z^2+z+\alpha^3,$ with $\alpha^4+\alpha+1=0.$

The least multiple affine of $f(z)$ is the polynomial $A(z)=\alpha+\alpha^8z+\alpha^{12}z^2+z^4.$

Finding roots of the polynomial $A(z)=\alpha+\alpha^8z+\alpha^{12}z^2+z^4.$ Consider the linearized polynomial $L(z)=\alpha^8z+\alpha^{12}z^2+z^4.$

First, we calculate the values:

$$L(1)=\alpha^8+\alpha^{12}+1=1+\alpha+\alpha^3;$$
$$L(\alpha)=\alpha^9+\alpha^{14}+\alpha^4=0;$$
$$L(\alpha^2)=\alpha^{10}+\alpha^{16}+\alpha^8=0;$$
$$L(\alpha^3)=\alpha^{11}+\alpha^{18}+\alpha^{12}=1+\alpha^3.$$

From which, we can find the roots of the equation $L(z)=\alpha$ by solving the system of equations:

$$\left[Z_0,Z_1,Z_2,Z_3\right]\begin{bmatrix} 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}=\left[0100\right].$$

The above system of equations has roots $Z=\left[1011\right]=\alpha^{13};\ \ Z=\left[1111\right]=\alpha^{12}.$ These are the two roots over $\mathrm{GF}(2^4)$, in which α^{12} is the root of $f(z).$ It can be factorized in:

$$f(z)=z^3+\alpha^{13}z^2+z+\alpha^3=(z+\alpha^{12})(z^2+\alpha z+\alpha^6).$$

Carrying out finding the roots of the polynomial $z^2+\alpha z+\alpha^6$, we have two roots α^7, α^{14}.

3.2. Solve a system of linear equations over the field GF(2^m)

The author has presented an effective way to solve the system of linear equations of the form (11) in [3]. This method uses linear transformations on columns to get a triangular idempotent matrix.

Consider a system of linear equations in a finite field of the form:

$$\left[Z_0, Z_1, ..., Z_{m-1}\right]\begin{bmatrix} L_{0,0} & L_{0,1} & ... & L_{0,m-1} \\ L_{1,0} & L_{1,1} & ... & L_{1,m-1} \\ ... & ... & ... & ... \\ L_{m-1,0} & L_{m-1,1} & ... & L_{m-1,m-1} \end{bmatrix} = \left[0, 0, ..., 0\right]. \quad (13)$$

Using linear transformations on columns (exchanging, adding, subtracting columns with multiplicating by coefficients) and upward rotating rows can transform a matrix to a triangular idempotent form L.

Definition: A matrix L of dimensions m x m is in the reduced triangular idempotent form if the elements below the main diagonal are zero, and the elements on the main diagonal are zero or one. At the same time, if an element on the main diagonal is zero, then the elements on the same column will be zero. If an element on the main diagonal is one, then the elements on the same row will be zero.

For example, the matrix over GF(5) has a triangular idempotent form.

$$\begin{bmatrix} 0 & 2 & 4 & 0 & 3 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 3 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Note that $L^2 = L$ from this, it follows that Z is the root of the equation $ZL = 0$ if Z is a linear combination of the rows of the matrix $L - I$ where I is a unit matrix of order *m*.

The algorithm to reduce a binary matrix to a triangular idempotent form is as follows.

Algorithm 2 [2]

1. If the top element of the leftmost column is one, do nothing. If this element is zero, exchange that column with the leftmost column whose top element is one and whose main diagonal element on that column is zero. If no such column can be found, then exchange the leftmost column with the leftmost column whose top element is one and whose main diagonal element is one. If such a column still cannot be found (the top row is all zero), do nothing.

2. Zero the top element of each column except the leftmost column by subtracting columns from the leftmost column.

3. Rotate rows upward and rotate columns leftward.

In many cases, it is necessary to solve a system of linear equations of the form: $ZL = U$ we will transform it into the form: $[Z, -1]\begin{bmatrix} L \\ U \end{bmatrix} = 0.$ By transformations on columns (algorithm 2), the matrix L is transformed into a triangular idempotent form L, and the vector U is correspondingly transformed to the form $\tilde{U}$. Then we construct the matrix $L - I.$ Carrying out the test $\tilde{U}\left[\tilde{L} - I\right] = 0$ then $\tilde{U}$ is a root (all the products of each component of $\tilde{U}$ with the main diagonal components $L - I$ are zero). The remaining roots are linear combinations of the non-zero rows of the matrix $L - I$.

4. ANALYSIS OF THE COMPLEXITY OF THE PROPOSED BCH ENCODING ALGORITHM

To evaluate the complexity of the proposed algorithm, we pay attention to some of the following characteristics. Calculating the syndrome and the error correction circuit is quite simple, and for all algebraic methods of BCH decoding these steps are performed so that the comparison process can skip these steps. On the other hand, the complexity of addition and multiplication depends on the implementation technique. The complexity of addition is much lower than square and multiplication, and the complexity of inverse is much larger than multiplication. We omit the addition (XOR) operations for simplicity and consider the most commonly used implementation techniques [9-11]. Here we study steps with considerable complexity.

In figure 1, in the case of three errors, it is necessary to use two squares and four multiplications to determine the error locator polynomial. Moreover, the operations are pipelined, and the computation delay is:

$$T_{lp1} = T_{SQ} + 2T_{MUL} + T_{XOR}$$

where T_{SQ}, T_{MUL}, T_{XOR} is the latency of squarer, multiplier, XOR, respectively.

Note that to implement a bit-parallel Montgomery squarer needs only *m*/2 XOR gates. The time delay for this Montgomery squarer is equivalent to the delay of one XOR gate [11].

Meanwhile, if using the BMA algorithm with parallel architecture (considered to have lower complexity than EA and other algorithms), to correct *t* errors, it is necessary to use one inverse and 2*t* multiplications, for *t* = 3 requires six multiplications and one inverse. Assuming the most efficient multiplier (Mastrovito multiplier) is used, the delay of the multiplication $T_{MUL} \approx \tau_{AND} + \lceil \log_2 m \rceil \tau_{XOR}$, thus, processing delay of BMA algorithm is [12]:

$$T_{lp2} \approx t(2 + \lceil \log_2 m \rceil)\tau_{AND} + t\lceil \log_2 m \rceil \tau_{XOR} + tm\tau_{OR},$$

in which τ_{AND}, τ_{OR} and τ_{XOR} is the computation time of AND, OR, and XOR, respectively.

With the step of finding the root of the error locator polynomial, the comparison of the complexity and delay is as follows. First, to calculate the coefficients of the affine polynomial $A(z)$, it is necessary to use four multiplications and two square operations, which can be performed in parallel in period T_{MUL}. Note that with the method of solving a system of linear equations, the highest-complexity step is computing the values of

$L(\alpha^i),\ i=0,1,...,m-1$. In the case of t = 3, calculating each value $L(\alpha^i),\ i=0,1,...,m-1$ requires two sequential square operations (to calculate z^4) and two multiplications. Algorithm 2 to find roots of the system of equations has negligible complexity because using only column shifts and permutations, and the implementation delay is 3*m* clocks (independent of *t*). Thus, the delay in finding the error location according to the proposed method increases mainly because of solving the system of equations based on algorithm 2. This delay is paying the price for a significant reduction in complexity.

When using the Chien procedure to find the root of the error locator polynomial, it is necessary to compute the polynomial $\sigma(\alpha^i),\ i=1,...,n$ and check if it is zero. Each calculation $\sigma(\alpha^i)$ needs to use *t* multiplications. When all calculations are performed in parallel, the delay will be minimal. Then the processing delay can be estimated as follows [12]:

$$T_{Chien} \approx \lceil \log_2 m \rceil \tau_{XOR} + \lceil \log_2 m \rceil \tau_{AND}$$

When *m* is large, the bit-parallel computation $\sigma(\alpha^i)$ becomes too complex, so use a bit-serial multiplier to reduce the complexity. Then the Chien procedure needs *n.m* clocks to complete the error locations.

Table 1. *Comparison of complexity when correcting three errors of the proposed method and the traditional methods.*

	Square	**Multiplication**	**Inverse**
The proposed method of finding the error locator polynomial	2	4	0
Finding the error locator polynomial based on BMA	0	6	1
The proposed method of finding the error locations	2+2*m*	4+2*m*	0
Finding error locations based on Chien search	0	3*n*	0

Note that with a polynomial basis, the complexity of the multiplier is about $m^2/2$ XOR gates, while that of the squarer is about XOR 2*m* OR gates [10]. Hence, the square operation is much simpler than the multiplication. Thus, the proposed method has a significantly reduced complexity and no inverse (complexity of inverse is O(m^3)).

Table 2. *Comparison of processing delay when correcting three errors of the proposed method and the traditional method when performing parallelly bit processing.*

Processing delay	Finding the error polynomial	Finding error location	Total delay
The proposed method	$T_{lp1} = 2\tau_{AND} + 2(1+\lceil \log_2 m \rceil)\tau_{XOR}$	$T_e = 3\tau_{AND} + (2+3m+3\lceil \log_2 m \rceil)\tau_{XOR}$	$T_1 = 5\tau_{AND} + (4+3m+2\lceil \log_2 m \rceil)\tau_{XOR}$
The traditional method	$T_{lp2} = 3(2+\lceil \log_2 m \rceil)\tau_{AND} + 3\lceil \log_2 m \rceil \tau_{XOR} + 3m\tau_{OR}$	$T_{Chien} \approx \lceil \log_2 m \rceil \tau_{XOR} + \lceil \log_2 m \rceil \tau_{AND}$	$T_2 = (6+4\lceil \log_2 m \rceil)\tau_{AND} + (3m+4\lceil \log_2 m \rceil)\tau_{XOR}$

5. CONCLUSIONS

The paper proposes a method to decode BCH using the Peterson algorithm to directly calculate the error locator polynomial and find the error location by finding the roots of an affine polynomial, which is a multiple of the error locator polynomial. The analytical results show that the proposed method has a low complexity with a significantly reduced number of multiplication operations by replacing multiplication with square and without performing the inverse. And the total delay is also decreased considerably. Especially when the field size is larger, the method's efficiency is more remarkable. In addition, the proposed method for finding the root of the error locator polynomial can be extended to higher multiple error correction cases where the total delay gain compared to the traditional method will increase more. The proposed method allows the fabrication of decoders with low complexity and latency, which opens the possibility of application in low latency communication systems with few resources.

REFERENCES

[1]. Bijan Ansari, *"Finite field arithmetic and its application in cryptography*," Dissertation for the degree Doctor of Philosophy in Electrical Engineering, University of California, Los Angeles (2012).

[2]. Elwyn R. Berlekamp, *"Algebraic Coding Theory* (Revised Edition)*,"* World Scientific Publishing Co. Pte. Ltd. (2015).

[3]. F. J. MacWilliams, N. J. A. Sloane, *"The theory of error correction codes,"* Elselvier (1977).

[4]. Tood K. Moon, *"Error correction coding: Mathematical methods and algorithms,"* John & sons, Inc. (2005).

[5]. K. Deerganghao, *"Channel coding technique for wireless communications,"* Springer, (2015).

[6]. Fedorenko S. V., Trifonov P. V. *"Finding roots of polynomials over finite fields*," IEEE Transactions on Communications, Vol. 50, Issue 11, pp. 1709–1711, (2002).

[7]. J. Fredenberger, *"A configurable Bose Chauhuri Hocquenghem codec architecture for flash controller applications*," Journal of circuits, systems and computer, Vol. 23, No. 02, (2013).

[8]. J. Fredenberger, B. N. Bailon, M. Shafies, *"Reduced complexity hard and soft decoding of BCH codes with application in concatenated codes*," EIT circuit, devices and systems, pp. 284-296, (2021).

[9]. Johann Groβschadl, *"A low-power bit serial multiplier for finite fields GF(2^m)*," 34th IEEE International symposium on ciruits and system, vol. IV, pp. 37-40, (2001).

[10].H. Wu, *"Montrogomery multiplier and squarer in GF(2^m)"* Lecture notes in computer science, (2000).

[11].Lin Shu, Costello, Daniel J., "*Error correcting codes,"* Prentice-Hall, Inc. (2004).

[12].D. Strukov, "*The area and latency tradeoffs of binary bit-parallel BCH decoders for prospective nano electronic memories,"* ACSSC Papers, (2007).

TÓM TẮT

Bộ giải mã mã BCH độ trễ thấp sử dụng tính chất của đa thức affine trên trường hữu hạn

Bài báo đề xuất thiết kế bộ giải mã mã BCH có độ phức tạp và độ trễ thấp nhờ thực hiện tính toán song song và đơn giản hóa việc tìm vị trí lỗi nhờ tìm nghiệm của đa thức affine trên trường hữu hạn. Thiết kế đã đề xuất có thể thực thi trên các nền tảng phần cứng giá thành thấp đồng thời cho phép ứng dụng trong các hệ thống thông tin có độ trễ rất thấp.

Từ khoá: Mã hóa sửa lỗi; Trường hữu hạn; Đa thức affine; Mã BCH.