Frontiers of Information Technology & Electronic Engineering
www.jzus.zju.edu.cn; engineering.cae.cn; www.springerlink.com
ISSN 2095-9184 (print); ISSN 2095-9230 (online)
E-mail: jzus@zju.edu.cn

A BCH error correction scheme applied to FPGA with embedded memory*

Yang LIU[†‡1], Jie LI[†‡1], Han WANG[1], Debiao ZHANG[1], Kaiqiang FENG[1], Jinqiang LI[2]

[1]*National Key Laboratory for Electronic Measurement Technology, North University of China, Taiyuan 030051, China*

[2]*Shandong Aerospace Electronic Technology Institute, Yantai 264000, China*

[†]E-mail: lylyly357@163.com; lijie@nuc.edu.cn

Received July 5, 2020; Revision accepted Dec. 26, 2020; Crosschecked June 8, 2021

Abstract: Given the potential for bit flipping of data on a memory medium, a high-speed parallel Bose–Chaudhuri–Hocquenghem (BCH) error correction scheme with modular characteristics, combining logic implementation and a look-up table, is proposed. It is suitable for data error correction on a modern field programmable gate array full with on-chip embedded memories. We elaborate on the optimization method for each part of the system and analyze the realization process of this scheme in the case of the BCH code with an information bit length of 1024 bits and a code length of 1068 bits that corrects the 4-bit error.

Key words: Error correction algorithm; Bose–Chaudhuri–Hocquenghem (BCH) code; Field programmable gate array (FPGA); NAND flash

https://doi.org/10.1631/FITEE.2000323 **CLC number:** TN911

1 Introduction

The NAND flash requires sector erasure before rewriting. As the time of erasure increases, the oxide layer of individual memory cells is broken down, and thus the floating gate can no longer bind electrons. The written data is flipped on the storage medium (Kumar et al., 2012). Thus, the read data is inconsistent with the stored one from sending to receiving. Hence, it is crucial to adopt an error correction algorithm with strong error-correction capabilities to guarantee the data accuracy and system reliability (Kim J and Sung, 2012; Yang et al., 2012; Cho et al., 2014). For field programmable gate array (FPGA) based storage systems, it is necessary to effectively adopt the limited resources of FPGA, so the complexity of the error correction algorithm must be strictly limited. Compared with Bose–Chaudhuri–Hocquenghem (BCH) codes and Reed–Solomon (RS) codes, low-density parity-check (LDPC) has disadvantage of high complexity and the storage voltage needs to be sensed multiple times during reading. Previous studies have shown that the adoption of RS codes can obtain good results in correcting burst errors. However, it is evident that random errors occur at a higher probability than burst errors in the NAND flash (Neubauer et al., 2007; Kim D et al., 2018). In these cases, it is more effective to adopt a BCH algorithm with excellent error correction capabilities. The BCH code is a forward error correction code using the original data and redundant information added in advance. It is defined by the roots of a generator polynomial in a finite field, and errors can be corrected within a certain amount (Moon, 2005).

Every frame of data has to go through the BCH encoder or decoder. Therefore, the quality of BCH architecture design can drastically affect the

‡ Corresponding authors

* Project supported by the National Natural Science Foundation of China (No. 61973280) and the China Postdoctoral Science Foundation (No. 2019M661069)

ORCID: Yang LIU, https://orcid.org/0000-0001-8541-8104; Jie LI, https://orcid.org/0000-0002-6488-3696

performance of the entire storage system. For encoding operations, there are several parallel linear feedback shift register (LFSR) architectures which have been discussed by Pei and Zukowski (1992), Shieh et al. (2001), and Hu et al. (2017). In terms of decoding operations, a tree architecture that is easy to implement has been introduced (Bellorado and Kavcic, 2010), aiming to obtain key equations.

The structure of the BCH error correction system shown in Fig. 1 includes an encoder and a decoder. On one hand, LFSR encodes the original data, and then the output parity bits and the original data are written into the NAND flash. On the other hand, the decoder composed of the syndrome solution module determines the error-locator polynomial module, and the Chien search module completes the error correction of the readout data that may be corrupted. In this study, we elaborate on the encoding implementation method. In addition, based on the previous research on the cyclic redundancy check (CRC) calculation method (Shieh et al., 2001), through further derivation, formulas are used to describe the structure relationship between the parity code computation and the syndrome computation. Hence, resource sharing and time multiplexing technology is employed to reduce the chip area. Furthermore, by virtue of derivation, a direct tree-decision architecture for calculating the error-locator polynomial is proposed. This simplifies the iteration judgment process. Later, judging the values of relevant parameters of the intermediate process is conductive to concluding the number of errors. Hence, the parallel Chien search process ends after searching the corresponding number of roots to achieve a higher decoding speed.

In the determination of the error-locator polynomial module and the Chien search module, polynomial multiplication and division are the calculation processes that are operated most frequently and consume the most resources. Therefore, we propose to pre-calculate the two representations of the elements in the Galois field (GF) (vector form and power form of the primitive element). Then they are stored as the look-up table in the embedded memory on the FPGA, so that the polynomial multiplication and division described by the complex combinatorial logic are converted into simple power-sum operations.

The contributions of this paper are summarized mainly in the following aspects:

1. The encoding method in this study has better comprehensive performance in terms of hardware overhead, speed, and flexibility.

2. Time multiplexing technology is used in the error detection module to reduce hardware overhead by more than 95%.

3. In determining the error-locator polynomial module, a direct and clear iterative path is planned. This reduces the complexity of the algorithm. Compared with the inversionless Berlekamp Massey (IBM) algorithm, the number of polynomial multiplication operations is also smaller.

4. In the Chien search module, the early termination technology is used to speed up the decoding process.

5. The polynomial multiplication and division described by the complex combinatorial logic are converted into simple power-sum operations, so that the path delay is reduced by 61.5%.

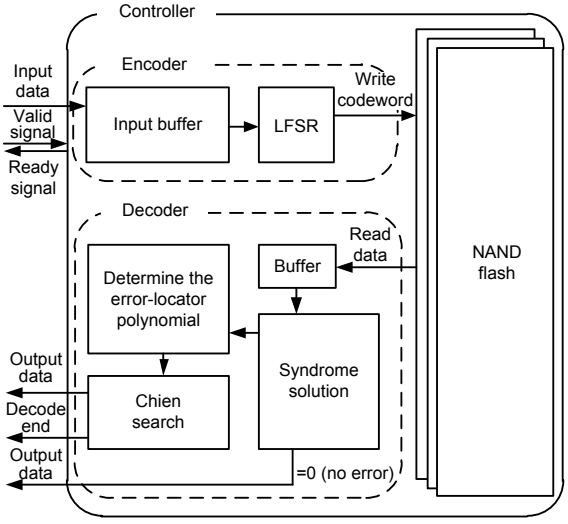

Fig. 1 Overall structure of the BCH error correction system

2 Code design and parallel encoding architecture

For a binary BCH code (n, k, t), k is the length of the primitive information $M(x)$, t is the number of errors that can be corrected, and n is the length of the entire codeword $C(x)$. The codeword is the combination of the information and the parity code $R(x)$. To make full use of the NAND flash during general

design, the redundant area of NAND flash will be considered to be fully used. The size of each page of the NAND flash MT29F512G08AUCBBH8-6IT is (16 384+1216) bytes. This requires that its code rate should be at least 16 384/(16 384+1216)=93.1%. In this study, we analyze the implementation process of the proposed error correction scheme with a BCH short code (1068, 1024, 4) as an example, and the code rate is 1024/1068=95.9%>93.1%. We assume the use of an encoding scheme with a degree of parallelism of 32, i.e., r=32. The BCH code has a total of 44 parity digits.

BCH code and CRC code are essentially cyclic codes (Pandian and Ray, 2015). In Eq. (1), the same systematic approach is taken for parity code computation in the GF when a generator polynomial $G(x)$ is specified:

$$R(x)=\left(x^{n-k}M(x)\right) \bmod G(x). \tag{1}$$

Shieh et al. (2001) presented an approach for parallel CRC computation. This method can also be applied to BCH encoding circuit, which is introduced next.

As shown in Eq. (2), $M(x)$ can be split into an accumulation of r subsequences represented by $M_i(x)$, for $0\leq i\leq r-1$, where m_i represents the bit of the standard binary form of the k-bit primitive information $M(x)$, that is, the coefficients of each order of the $M(x)$ polynomial form.

$$\begin{aligned} M(x)&=m_{k-1}x^{k-1}+m_{k-2}x^{k-2}+\cdots+m_1x+m_0\\ &=M_{r-1}(x)+M_{r-2}(x)+\cdots+M_1(x)+M_0(x), \end{aligned} \tag{2}$$

where

$$\begin{aligned} M_i(x)&=\sum_{j=0}^{k/r-1}m_{jr+i}x^{jr+i}\\ &=m_ix^i+m_{r+i}x^{r+i}+\cdots+m_{k-r+i}x^{k-r+i}. \end{aligned} \tag{3}$$

$x^{n-k}M(x)$ can be derived as

$$\begin{aligned} x^{n-k}M(x)&=x^{n-k}\left(M_0(x)+M_1(x)+\ldots+M_{r-1}(x)\right)\\ &=x^{n-k}\sum_{i=0}^{r-1}\sum_{j=0}^{k/r-1}m_{jr+i}x^{jr+i}\\ &=\sum_{i=0}^{r-1}\left(x^{n-k-(r-1)+i}\sum_{j=0}^{k/r-1}m_{jr+i}x^{(j+1)r-1}\right)\\ &=\sum_{i=0}^{r-1}x^{n-k-(r-1)+i}\bar{M}_i(x), \end{aligned} \tag{4}$$

where

$$\begin{aligned} \bar{M}_i(x)&=\sum_{j=0}^{k/r-1}m_{jr+i}x^{(j+1)r-1}\\ &=m_ix^{r-1}+m_{r+i}x^{2r-1}+m_{2r+i}x^{3r-1}+\cdots+m_{k-r+i}x^{k-1}. \end{aligned} \tag{5}$$

Fig. 2 describes the transformation process for the design of code (1068, 1024, 4). By adjusting the order of multiplier x, it converts $x^{n-k}M(x)$ to a sequence form by premultiplying $\bar{M}_i(x)$ by $x^{n-k-(r-1)+i}$. The dotted parts indicate that $M_i(x)$ is redivided into r, i.e., 32, parts with the same power polynomial representation, and there are $r-1$, i.e., 31, consecutive zero digits between two adjacent significant information bits.

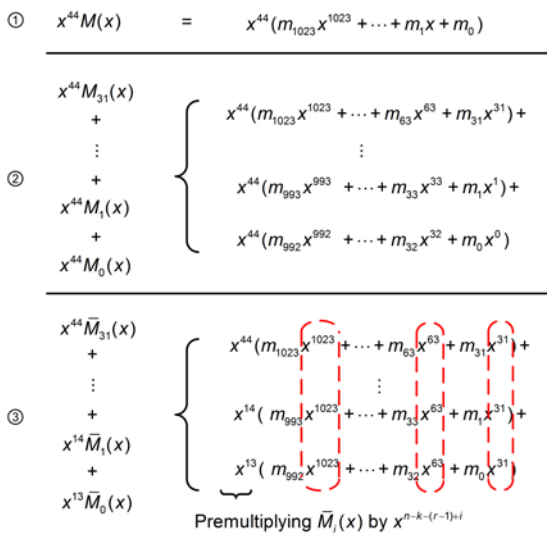

Fig. 2 Transformation process of Eq. (4) for code (1068, 1024, 4)

Premultiplying $\bar{M}_i(x)$ by $x^{n-k-(r-1)+i}$ is realized through shifting $\bar{M}_i(x)$ into the $(r-1-i)^{\text{th}}$ position of the circuit counting from the far right. Therefore, Fig. 3 conceptually depicts this encoding scheme in the form of LFSR. The symbol "⊕" refers to the modulo-2 addition operation (XOR) over the GF. The corresponding position connection relationship of the feedback is determined based on whether the coefficient g_i of $G(x)$ is 1 (Ayinala and Parhi, 2011; Jung et al., 2015). The content of D-FF b_i represents the parity-check digits after all the k-digit original information entered in parallel from the most significant bits.

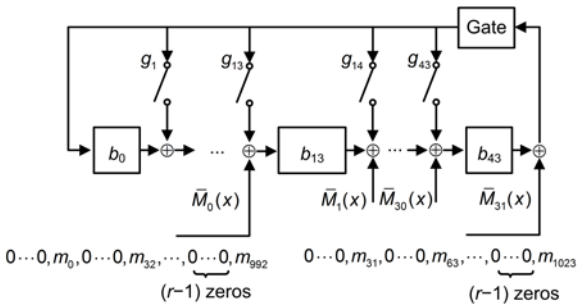

Fig. 3 Encoding scheme in the form of LFSR architecture

For the 32-bit parallel encoding method with the original information of 1024 bits, a total of 32 iterations needs to be executed. For $0\le t_e\le 31$, $\boldsymbol{I}(t_e)$ is the 32-bit information input at time t_e:

$$\boldsymbol{I}(t_e)=\left[\boldsymbol{0}\middle|m_{1024-32(t_e+1)}, m_{1024-32(t_e+1)+1}, \cdots, m_{1024-32(t_e+1)+31}\right]^{\mathrm{T}}_{1\times 44}. \tag{6}$$

First, the highest 32-bit $\boldsymbol{I}(0)$ (corresponding to the sequence m_{1023}, m_{1022}, …, m_{992}) comes into the circuit in parallel at the same time. After performing this process, the state $\boldsymbol{b}(1)$ of the register is denoted by Eq. (7) when the initial state is zero:

$$\boldsymbol{b}(1)=\boldsymbol{T}_g\boldsymbol{I}(0), \tag{7}$$

where $\boldsymbol{T}_g$ is the transformation matrix describing the connection relationship of the feedback loop in the circuit of Fig. 3, and is symbolized by $G(x)$.

$$\boldsymbol{T}_g=\begin{bmatrix}0&0&0&\cdots&0&1\\1&0&0&\cdots&0&g_1\\0&1&0&\cdots&0&g_2\\\vdots&\vdots&\vdots&&\vdots&\vdots\\0&0&0&\cdots&0&g_{42}\\0&0&0&\cdots&1&g_{43}\end{bmatrix}. \tag{8}$$

Second, the process of inputting subsequent 32-bit zeros in parallel to the encoder is equivalent to the register shift operation in the case without input. It manifests as the last state multiplied by a transformation matrix. A total of 31 continuous zero vectors with a length of 32 bits to be input is required before the next non-zero bit entry. The current state $\boldsymbol{R}(1)$ can be derived through combining matrix calculation as

$$\boldsymbol{R}(1)=\boldsymbol{T}_g^{31}(\boldsymbol{T}_g\boldsymbol{I}(0))=\boldsymbol{T}_g^{32}\boldsymbol{I}(0). \tag{9}$$

Finally, we perform successive iterations until the whole message bits $M(x)$ have been shifted in, and the update process is indicated in Eq. (10). Thus, a 32-bit parallel encoding is completed as shown in Fig. 4.

$$\boldsymbol{R}(t_e+1)=\boldsymbol{T}_g^{31}\left[\boldsymbol{T}_g\left(\boldsymbol{R}(t_e)\oplus\boldsymbol{I}(t_e)\right)\right]=\boldsymbol{T}_g^{32}\left(\boldsymbol{R}(t_e)\oplus\boldsymbol{I}(t_e)\right). \tag{10}$$

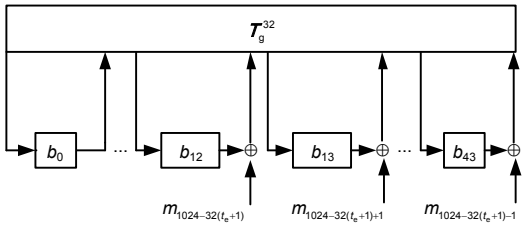

Fig. 4 Parallel encoding LFSR architecture

3 Parallel decoding architecture

3.1 Syndrome solution

The syndrome solution process consists of the following two steps: first, we calculate syndrome $S(x)$ to detect whether there is an error, and calculate all the $2t$ syndromes, s_1, s_2, …, s_{2t}. Let $V(x)$ be the received polynomial. In the first step, similar to the encoding operation, the syndrome $S(x)$ is the remainder obtained through dividing $V(x)$ by $G(x)$, and the only difference between them is that the syndrome generation does not require to multiply x^{n-k}. The conventional serial direct method is to create a new logical structure (Fig. 5), i.e., shifting $V(x)$ into the circuit from the leftmost part. If the parallel method proposed by Shieh et al. (2001) is used to calculate the syndrome, then its LFSR structure is as shown in Fig. 6. However, this will clearly bring extra hardware cost. How to apply the time multiplexing and resource sharing technique to reduce hardware consumption is presented in the following:

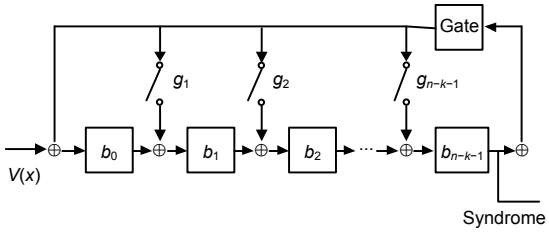

Fig. 5 LFSR architecture of the conventional serial direct method for the syndrome solution

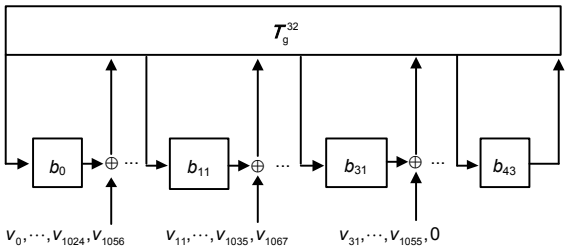

Fig. 6 LFSR architecture of the parallel method for the syndrome solution

A further derivation is provided in Eq. (11):

$$\begin{aligned}S(x)&=V(x)\bmod G(x)\\&=\left(\sum_{i=n-k}^{n-1}v_ix^i+\sum_{i=0}^{n-k-1}v_ix^i\right)\bmod G(x)\\&=\left(x^{n-k}\sum_{i=n-k}^{n-1}v_ix^{i-(n-k)}\right)\bmod G(x)+\left(\sum_{i=0}^{n-k-1}v_ix^i\right)\bmod G(x)\\&=\left(x^{n-k}V_m\right)\bmod G(x)+\sum_{i=0}^{n-k-1}v_ix^i,\end{aligned}\tag{11}$$

where V_m represents $\sum_{i=n-k}^{n-1}v_ix^{i-(n-k)}$. $(x^{n-k}V_m)\bmod G(x)$ is calculated in the same way as the encoding (It can be seen that it is consistent with the calculation form of Eq. (1)). For NAND flash, reading data operations and writing data operations generally cannot be synchronized. The logical structure used for encoding and syndrome generator is shared considering that these two operations are not performed simultaneously. So, the first k bits $x^{n-k}V_m$ of $V(x)$ can be shifted into the parallel encoding circuit, and the remaining $(n-k)$ bits $\sum_{i=0}^{n-k-1}v_ix^i$ are XORed with the contents of the LFSR after the completion of the last phases. Each bit of the obtained $(n-k)$-bit binary result "sc" represents the coefficient of each order of $S(x)$. It can be determined that no error occurs in the case that $S(x)$ is zero. Otherwise, the second step to calculate s_1, s_2, …, s_{2t} is performed using Eq. (12):

$$\begin{aligned}s_h&=S(\alpha^h)\\&=\mathrm{sc}_{n-k-1}\alpha^{h(n-k-1)}+\mathrm{sc}_{n-k-2}\alpha^{h(n-k-2)}+\cdots+\mathrm{sc}_1\alpha^h+\mathrm{sc}_0,\end{aligned}\tag{12}$$

where α denotes the primitive element, which is the root of the primitive polynomial, and $1\le h\le 2t$.

It is worth noting that when it is necessary to solve all the syndromes, the conventional method is to directly substitute α^h into the received polynomial $V(x)$ as described in Eq. (13). Each s_h is transformed into $(h(n-1)+1)$ bits. According to the needs of subsequent calculations, a p-parallel circuit is used to convert it into an m-bit vector form (taking the remainder of the primitive polynomial), so a total of $(h(n-1)+1)/p$ cycles is executed. If s_h is obtained using Eq. (12), it can be completed by executing only $[h(n-k-1)+1]/p$ cycles:

$$s_h=V(\alpha^h)=v_{n-1}\alpha^{h(n-1)}+v_{n-2}\alpha^{h(n-2)}+\cdots+v_1\alpha^h+v_0.\tag{13}$$

3.2 Determination of the error-locator polynomial

The error-locator polynomial $\sigma(x)$ shown in Eq. (14) corresponds to a correctable error pattern. The BM algorithm is the most classic method to find the error-locator polynomial coefficients. The iterative method of the BM algorithm (Massey, 1969; Berlekamp, 2015) is illustrated in the following: a complete iteration process consists of $2t$ loops in total, and each iteration introduces a correction item d_μ to modify $\sigma^{(\mu)}(x)$ to obtain $\sigma^{(\mu+1)}(x)$ according to Eqs. (15) and (16).

$$\sigma(x)=\sigma_0+\sigma_1x+\cdots+\sigma_lx^l,\tag{14}$$

$$d_\mu=s_{\mu+1}+\sigma_1^{(\mu)}s_\mu+\sigma_2^{(\mu)}s_{\mu-1}+\cdots+\sigma_{l_\mu}^{(\mu)}s_{\mu+1-l_\mu},\tag{15}$$

$$\sigma^{(\mu+1)}(x)=\sigma^{(\mu)}(x)+d_\mu d_\rho^{-1}\sigma^\rho(x)x^{(\mu-\rho)},\tag{16}$$

where l represents the order of the polynomial, and $\sigma^{(\mu)}(x)$ denotes the error-locator polynomial obtained at the μ^{th} iteration. The selection principle of ρ is making $\rho-l_\rho$ the largest, satisfying $\rho<\mu$ and $d_\rho\ne0$ (here, l_ρ represents the order of $\sigma^{(\rho)}(x)$). Clearly, using FPGA to implement such an iterative process is complicated.

A straightforward IBM tree-decision method has been proposed (Bellorado and Kavcic, 2010; Xu et al., 2013). Similar to the BM algorithm, during the μ^{th} iteration, IBM determines the update result of $\sigma^{(\mu+1)}(x)$ by judging whether d_μ is zero. However, when determining d_μ according to Eq. (15), it is necessary to know in advance $\sigma^{(\mu)}(x)$ obtained in the previous iteration, and as shown in Eq. (16), the calculation of $\sigma^{(\mu)}(x)$ may introduce polynomial division operations that are difficult to implement on hardware. In

response to this problem, for the IBM algorithm, if a factor $\beta-1$ is presented in the update process of d_μ or $\sigma^{(\mu)}(x)$, βd_μ and $\beta\sigma^{(\mu)}(x)$ are always found instead of d_μ and $\sigma^{(\mu)}(x)$, respectively. Then, the result obtained can be used in the next iteration. Although this approach changes the values of the coefficients of each order in the error-locator polynomial, it does not change the root of the error-locator polynomial, so it does not affect the use of the Chien search module introduced in Section 3.3 to obtain the result of the final error location. However, at the same time, this method increases the number of polynomial multiplication operations (This shortcoming will be analyzed in Section 4).

Based on the tree-decision idea of the IBM algorithm (but different from the IBM algorithm, which solves the problem of computing polynomial division on hardware at the cost of increasing the number of polynomial multiplication operations), we conduct further derivation and summary, and propose a method to solve the key equation that does not require inverse operation or use any conventional combinatorial logic multiplier of GF. So, we can quickly and accurately determine the number of errors and the error-locator polynomial coefficients.

d_μ is always zero in odd iterations so that we can apply the lookahead technique to bypass all odd iterations and lower the number of iteration cycles from $2t$ to t. Moreover, it can be found through a summary that when the number of error correction bits is relatively small, the choice of ρ at the corresponding number of iterations is fixed (Table 1).

Consequently, through this simplified algorithm, the error-locator polynomial can be directly obtained according to Eq. (16). We take the short code (1068, 1024, 4) over GF(2^{11}) as an example. Assume that there are some bit flips in $C(x)$, i.e., $S(x)=d_0\neq0$. As shown in Fig. 7, by judging whether the correction item d_μ is zero, we build a straightforward tree-decision architecture for $t=4$. e_x indicates that the number of errors exceeds the error correction capabilities under the corresponding circumstances. The error-locator polynomial can be directly solved with this architecture. This simplifies the complex iteration judgment process shown in Eq. (13). There is no need to traverse the previously solved $\sigma(x)$ to find parameter ρ that meets the conditions for each iteration. In this way, the decoding efficiency can be significantly improved and the algorithm complexity is reduced. Table 2 gives the final error-locator polynomial, and its order is the number of errors. The order of the derived error-locator polynomial is greater than 4. For example, when $d_2=0$, $d_4=0$, and $d_6\neq0$, the data is uncorrectable.

Table 1 Choice of ρ at the corresponding number of iterations

μ	ρ	μ	ρ
0	−1	4	2
2	0	6	4

Table 2 Final error-locator polynomial and the number of errors

Symbol	$\sigma^{(7)}(x)/\sigma^{(8)}(x)$	Number of errors
A	$1+s_1x$	1
B	$1+s_1x+d_4d_0^{-1}x^4$	4
C	$1+s_1x+d_6d_4^{-1}x^2+d_6d_4^{-1}s_1x^3+d_4d_0^{-1}x^4$	4
D	$1+s_1x+d_2d_0^{-1}x^2$	2
E	$1+s_1x+(d_2d_0^{-1}+d_4d_2^{-1})x^2++d_4d_2^{-1}d_0x^4$	3
F	$1+s_1x+(d_2d_0^{-1}+d_4d_2^{-1}+d_6d_4^{-1})x^2$ $+(d_4d_2^{-1}d_0+d_6d_4^{-1}s_1)x^3+d_6d_4^{-1}d_2d_0^{-1}x^4$	4

When calculating the error-locator polynomial $\sigma(x)$ or the correction term d_μ, it is necessary to perform polynomial multiplication, division, and addition operations over GF. The addition on hardware can be achieved efficiently through a simple XOR gate. However, traditional GF polynomial multiplication and division hardware implementations require a lot of logic resources.

GF has two properties. First, any non-zero vector element in GF can be expressed in the form of power of the primitive α. Second, the obtained result of the element in GF is still the element in the domain, no matter what operation is performed. In addition, embedded memory is available on modern FPGAs for use. So, we propose to pre-compute the two equivalent forms of the elements over GF(2^m), namely, power and vector. The IP core block memory generator to be configured for FPGA is ROM, and two parts of ROM are used to separately store these two forms of mapping relationship. These are called vectorROM and powerROM. vectorROM uses the address represented by the power of consecutive primitive α to index the corresponding vector. powerROM uses the address represented by the vector to index the corresponding

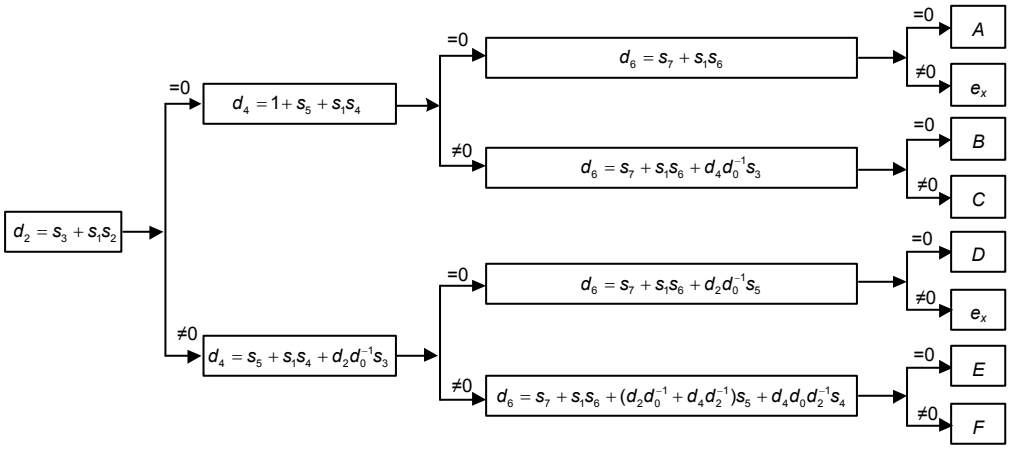

Fig. 7 A straightforward tree-decision architecture for t=4

power. The depth of these two ROMs is 2^m-1, the width is m, and the address width is also m.

Suppose that the forms of the power of the corresponding primitives of the two non-zero polynomial vectors $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$ are α^y and α^z, respectively. To obtain the result of multiplying $\boldsymbol{p}_1$ by $\boldsymbol{p}_2$, different from the traditional GF multiplier, it is necessary to consider vectors $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$ as the input addresses and read the content stored in the addresses by addressing powerROM to obtain the power values y and z of the corresponding form. According to the properties of GF, Eq. (17) can be obtained:

$$\boldsymbol{p}_1\boldsymbol{p}_2 = \alpha^y\alpha^z = \alpha^{(y+z)\bmod(2^m-1)}, \tag{17}$$

where

$$\alpha^y\alpha^{-z} = \begin{cases} \alpha^{(y-z)}, & y \geq z, \\ \alpha^{2^m-1+(y-z)}, & y < z. \end{cases} \tag{18}$$

Hence, it is necessary to calculate only the value of $(y+z) \bmod (2^m-1)$ to be used as the address of vectorROM, and vectorROM is addressed to transform the result back to the vector form. The calculation processes of the polynomial division and multiplication are similar. Suppose that to obtain the result, we perform the address first and read out y and z through the above process. According to the properties of GF, as shown in Eq. (18), the operation of the polynomial division can be divided into two circumstances, thereby transforming complicated polynomial multiplication and division operations into simple power-sum operations. This method calculates polynomial multiplication and division with two ROM resources of the size of $m\times(2^m-1)$ as a look-up table, which is called LUT-PMD.

The example short code (1068, 1024, 4) belongs to GF(2^{11}). Table 3 gives the power forms corresponding to a part of the vectors over GF(2^{11}). Assume that we multiply two polynomials $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$:

$$\begin{cases} \boldsymbol{p}_1 = [0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1], \\ \boldsymbol{p}_2 = [1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0]. \end{cases} \tag{19}$$

Table 3 Power form corresponding to vectors over GF(2^{11})

Power form	Power value	Vector form
$\alpha^0=\alpha^{15}$	00000000000	00000000001
α^1	00000000001	00000000010
⋮	⋮	⋮
α^{2044}	11111111100	10100000010
α^{2045}	11111111101	01000000001
α^{2046}	11111111110	10000000010

According to Table 3, multiplying $\boldsymbol{p}_1$ by $\boldsymbol{p}_2$ is equivalent to multiplying α^{2045} by α^{2046}, and the power values 11111111101 (2045) and 11111111110 (2046) stored in the powerROM with 01000000001 and 10000000010 are read as addresses, respectively. At this time, only calculation of (2045+2046) mod ($2^{11}-1$) is needed, and the result is 2044 (corresponding to the power value of 11111111100 in Table 3). Then, we read the vector 10100000010 with 11111111100 as the address from vectorROM, which is considered as the final calculation result. The division process is similar and is repeated here.

The sufficient memory resources in FPGA can be configured as an extremely wide word ROM. Therefore, highly parallel processing can satisfy the multiple calculations of polynomial multiplication or division at the same time. Because this is a direct addressing method, a lot of calculations are avoided. In comparison with the use of complex combinational logic circuit to calculate polynomial multiplications, this method of combining look-up tables presents advantages of low delay and a simple structure.

3.3 Chien search

The Chien search algorithm substitutes all the elements separately over GF(2^m) into polynomials to verify whether α^{-i} is a root of the error-locator polynomial (Zhang M et al., 2015). If $\sigma(\alpha^{-i})=0$ is fulfilled, then the i^{th} bit of $V(x)$ is an error location. We invert all the data in the wrong position to obtain the final correct data. Multiple Chien modules can be used to search the root, and simultaneously to improve the efficiency and reduce the number of cycles from n to n/p for a p-parallel circuit, as shown in Fig. 8. The circuit is verified with a parallelism of 32. The symbol "⊗" refers to the polynomial multiplication operation. The proposed LUT-PMD method can be used for calculation.

Because it is not necessary to correct the errors that occur in the parity digits, for the short code (1068, 1024, 4), we can further narrow the root from α^{980} to α^{2003}. Furthermore, we can judge how many bits the codeword has flipped according to the degree of the polynomial in Table 2. Thus, we can early terminate the Chien search module after completing the search for the specified interval or finding the corresponding number of error positions instead of traversing the entire GF as the end condition. This results in remarkable power saving and speed improvements. The early termination method and the root distance reduction method are shown in Fig. 9.

4 Results and discussion

The whole design of the parallel BCH correction system has been realized with Verilog Hardware Description Language (HDL) and synthesized using a synthesis tool. The circuit has been successfully verified on the Virtex-7 xc7vx690tffg1930-3 of the 7 series FPGA. The clock frequency is 100 MHz. The total power is 1.049 W. Table 4 illustrates the resource use of the design, where BRAM refers to block RAM. The BRAM in Xilinx® 7 series FPGA stores up to 36 000 bits of data. The use of BRAM resources does not take up additional logic resources and the BRAM has high speed, but the BRAM resource used can be only an integer multiple of its block size (even if only 1 bit of data is stored, one BRAM is required).

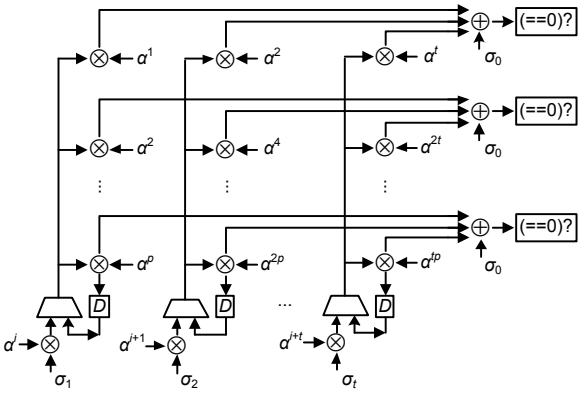

Fig. 8 Parallel Chien search circuit

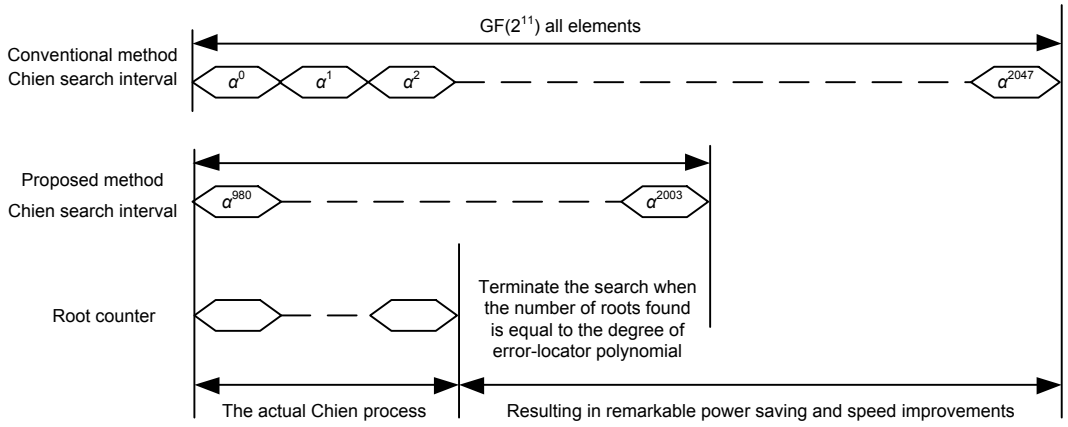

Fig. 9 Early termination method and root distance reduction method

Table 4 Resource use of the design

Resource	LUT	Flip flop	BRAM
Encoder	173	116	
Decoder	14 034	2078	272
Available one	433 200	866 400	1470

LUT: look-up table

The simulation environment is set to allow the codewords transmitted in the binary symmetric channel to be flipped with a crossover probability p_c. Algorithm 1 illustrates the simulation process (Unal et al., 2018). Fig. 10 shows the comparison of the frame error rate (FER) before and after the bit error is corrected by the system, limited as to the amount of data and the verification time. The smallest value of the cross flip probability is set to 0.0001. In fact, the bit-flip probability of NAND flash is much lower than this value by several orders of magnitude. At the same time, to ensure the accuracy of the result, FER is calculated when the cumulative number of error frames reaches 50. It can be seen from Fig. 10 that this system can significantly reduce the FER of the data.

Each part playing the main role is analyzed separately with the BCH short code (1068, 1024, 4) as an example.

For encoding, a method to reduce the complexity of the parallel structure by constructing the optimal transformation matrix was proposed (Hu et al., 2017). Zhang XM (2019) made improvements on that basis, and proposed that the number of gates required can be reduced by sharing the common terms of each register output formula. However, because the dimension of the transformation matrix is too high, for different generator polynomials, it takes hours to search for the vector of the optimal matrix using an exhaustive algorithm. The number of common terms is also uncertain. Table 5 gives a comparison of the additional work required by several methods (the same work is required when calculating the syndrome). Compared with Hu et al. (2017) and Zhang XM (2019)'s methods, the derivation of state updating equations is simpler based on the methodology in this study. The method can be easily expanded to other r-bit parallel input applications without added complexity. The optimal method should be the result of a trade-off in terms of area, speed, and flexibility. We provide the hardware requirements of five encoding methods shown in Table 6 when the control circuit is ignored. Fig. 11 shows the performance comparison of the

Algorithm 1 Simulation flow for generating FER

Input: 1024-bit information and crossover probability p_c
Output: frame error rate (FER) over each p_c
Initialization: Frame_counter=0
  for each crossover probability p_c **do**
    $p_c \in [0.0001, 0.0007]$
    Error_frame_counter=0
    while Error_frame_counter<50 **do**
      Generate randomly 1024-bit information
      Encode to obtain the codeword (a frame of data)
      Frame_counter=Frame_counter+1
      Add noise to the frame using p_c
      Decode (Frame)
      if Syndrome≠0 **then**
        Error_frame_counter=Error_frame_counter+1
      end if
      if Error_frame_counter==50 **then**
        FER=50/Frame_counter
      end if
    end while
  end for

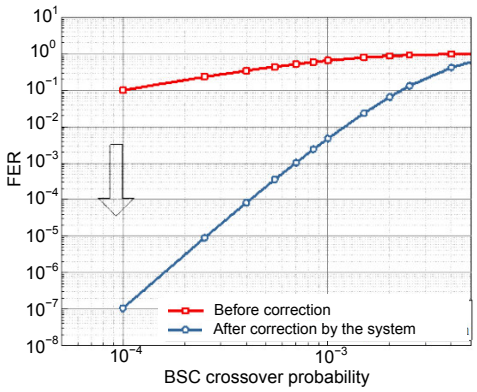

Fig. 10 Frame error rate (FER) vs. binary symmetric channel (BSC) crossover probability of error

Table 5 Comparison of additional work required for the five encoding methods

Method	Additional work
Serial, proposed, and conventional	No
Zhang XM (2019)'s	Write the corresponding exhaustive algorithm, and spend hours for searching for the vector of the optimal matrix
Hu et al. (2017)'s	Write the corresponding exhaustive algorithm, and spend hours for searching for the common terms of each register output formula and the vector of the optimal matrix

The same work is required when calculating the syndrome

five methods. The relative gate count in terms of 2-input NAND gate is NAND:XOR:D-FF=1:3:8. The hardware overhead is obtained after normalizing the serial implementation. It is noted that compared with the conventional parallel encoding method (Zhang XM and Parhi, 2005), the presented methodology reduces the hardware overhead by 39% under the same conditions. The four parallel encoding methods for comparison feed 32 bits in parallel. The hardware overhead of the two other parallel methods is also significantly reduced. However, compared with the method in this study, the hardware overhead improvements of these two methods are not obvious. For the BCH code (1068, 1024, 4), the hardware consumption of Hu et al. (2017)'s method is 9% more than that of this study, while Zhang XM (2019)'s method can reduce the hardware consumption by 6%. At the same time, when the path delay cannot be effectively restricted, Hu et al. (2017)'s method and Zhang XM (2019)'s method split the parallel update process originally completed in one clock cycle into two clock cycles. Therefore, 64 clock cycles are required for both methods, while the proposed method requires only 32 clock cycles. In summary, the proposed method is obviously more advantageous in terms of speed, flexibility, and path delay, taking into account the ability to reduce hardware overhead.

In the error detection module, the hardware requirements of five syndrome calculation methods are shown in Table 6. Fig. 12 shows the performance

Table 6 Comparison of hardware requirements for the five encoding methods

Method	Encode			Syndrome		
	D-FF	XOR	Gate count	D-FF	XOR	Gate count
Serial	44	39	469	44	20	412
Zhang XM (2019)'s	44	726	2530	44	784	2704
Hu et al. (2017)'s	44	852	2908	44	835	2857
Conventional	44	1358	4426	44	1190	3922
Proposed	44	774	2674	Multiplex existing encoding circuit+44 XOR		

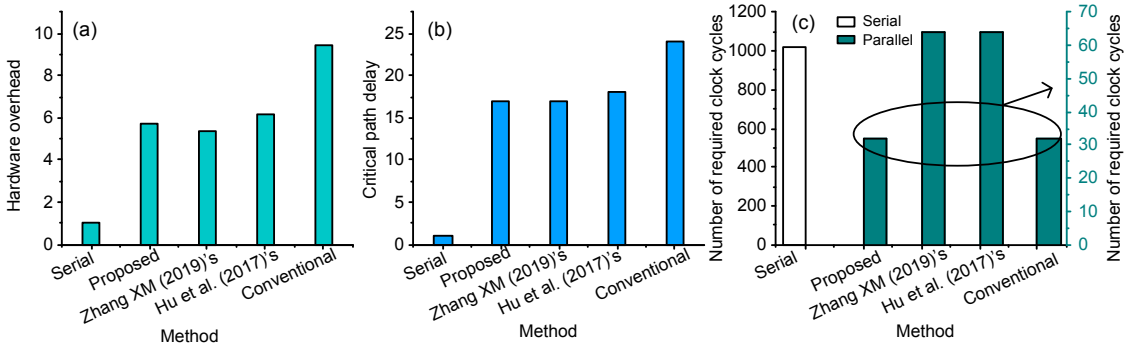

Fig. 11 Performance comparison of the five encoding methods: (a) hardware overhead; (b) critical path delay; (c) number of required clock cycles

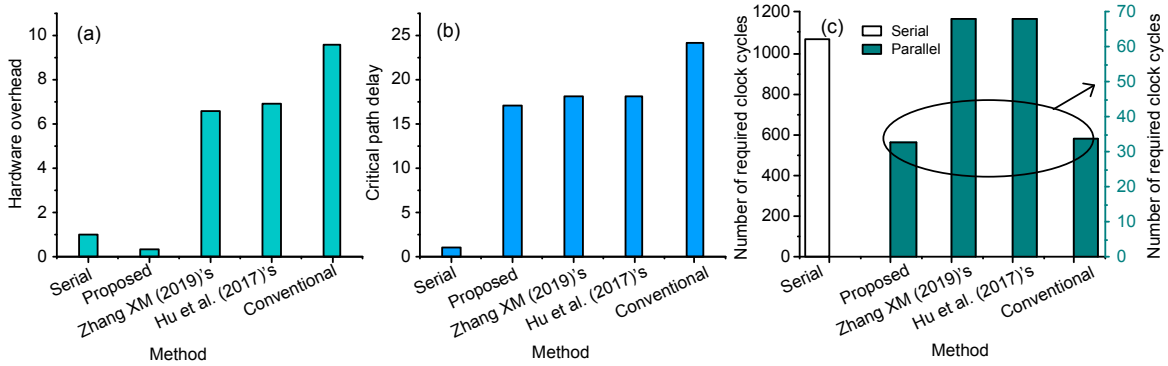

Fig. 12 Performance comparison of the five syndrome calculation methods: (a) hardware overhead; (b) critical path delay; (c) number of required clock cycles

comparison of the five methods. To adopt the time multiplexing encoding LFSR architecture proposed in this study, add just additional 44 XOR gates to complete the calculation of $S(x)$ and avoid the use a new LFSR. Therefore, compared to Zhang XM (2019)'s method, Hu et al. (2017)'s method, and the conventional method, the hardware overhead is decreased by 95.1%, 95.4%, and 96.6%, respectively. Similar to encoding, it has advantages in speed, flexibility, and path delay.

We provide a definite form for each step in Fig. 7 and Table 2. The polynomial can be simply and directly determined from the formula. This greatly speeds up the iterative process. The IBM algorithm (Bellorado and Kavcic, 2010; Xu et al., 2013) has the characteristic that it does not need to calculate polynomial division. The reason is that the wrong location is finally determined by the root-seeking, and the two sides of the equation $\sigma(\alpha^{-i})=0$ multiplied by the same factor will not change the result of the root. The polynomial division in the error-locator polynomial can be converted to polynomial multiplication. Take path *E* as an example, and transform it to find the root of Eq. (20):

$$d_0d_2+s_1d_0d_2x+(d_2d_2+d_4d_0)x^2+d_4d_0d_0x^3=0. \quad (20)$$

In such a case, it will inevitably increase the number of polynomial multiplication operations. The IBM algorithm needs to perform 20 polynomial multiplication operations on calculating the coefficients of the error-locator polynomial for the entire path *E*, while taking advantages of the proposed algorithm, compiling the look-up table requires only a total of 15 multiplication or division operations that correspond to power-sum operations. Table 7 provides a comparison of the total number of polynomial multiplication operations required for each path using these two algorithms.

In terms of Chien search, the hardware implementation is not improved, whereas the search time is significantly shortened compared with that of the conventional traversal search algorithm, and the effect of speeding up decoding is also realized. The specific speed increase depends on where the bit flip occurs.

Table 7 Number of polynomial multiplication operations required for each path

Algorithm	Number of operations					
	Path=*A*	*B*	*C*	*D*	*E*	*F*
Proposed	3	6	9	8	15	21
IBM	3	7	11	10	20	29

In this study, we propose a new hardware implementation method of polynomial multiplication and division, i.e., LUT-PMD. Compared with the traditional calculation of polynomial multiplication using combinational logic (Paar, 1996), the ratio of the critical path delay for FPGA integration of the two implementations is about 1:2.6, and the critical path delay is reduced by 61.5%. This indicates that LUT-PMD can run at a higher system clock frequency. However, since LUT-PMD requires two addressing processes, it takes at least three clock cycles to obtain the calculation result. If the combinational logic is used to calculate the polynomial multiplication, the calculation can be completed in one clock cycle. Therefore, if the polynomial multiplication is calculated only once, LUT-PMD has no obvious advantage in terms of calculation speed. However, a pipelined mode of operation can be used. That is, after the addressing of the multiplier is completed in the first clock, the addressing of the second group of multipliers is also completed, while assigning the address of vectorROM in the second clock. The overall calculation speed can be improved.

This study takes the BCH code (1068, 1024, 4) as an example to analyze the feasibility of the algorithm. The code rate is 95.88%, and the overall error correction can be completed for the memory with a bit error rate (BER) less than 3.74%. The main area of the NAND flash using single-level technology basically accounts for about 93%. The magnitude of the initial BER is 10^{-8} (BER will be affected by the environment and the number of times to read and write data). Therefore, it can fully meet the error correction needs of various types of NAND flash under harsh conditions.

When BCH codes with other parameters are adopted, new circuits can be designed through the method described in this paper. As in the encoding section, for different generator polynomials, the LFSR structure and the degree of parallelism can be flexibly adjusted. In the decoding part, the encoding circuit is multiplexed to calculate the syndrome. Next, to obtain a higher-order error-locator polynomial, on

the basis of $\sigma^{(7)}(x)$, the idea of tree-decision can be used to complete further derivation. It is also convenient to adjust the parallel search bit width and search interval in the Chien search module. Finally, we configure ROMs of different capacities as needed to complete the polynomial multiplication and division operations of the GF to which the BCH code belongs.

5 Conclusions

In this paper, we have optimized a high-speed parallel BCH error correction scheme applied to FPGA with embedded memory to strengthen the fault tolerance for storage systems. This novel and efficient code algorithm has been proven to work correctly, and it is also applicable to BCH codes with other design parameters.

Contributors

Yang LIU conceived the research and drafted the manuscript. Jie LI provided the theory analysis and the instruction. Debiao ZHANG and Kaiqiang FENG contributed to the analysis and manuscript preparation. Han WANG and Jinqiang LI performed the simulations. Yang LIU and Jie LI revised and finalized the paper.

Compliance with ethics guidelines

Yang LIU, Jie LI, Han WANG, Debiao ZHANG, Kaiqiang FENG, and Jinqiang LI declare that they have no conflict of interest.

References

Ayinala M, Parhi KK, 2011. High-speed parallel architectures for linear feedback shift registers. *IEEE Trans Signal Process*, 59(9):4459-4469. https://doi.org/10.1109/TSP.2011.2159495

Bellorado J, Kavcic A, 2010. Low-complexity soft-decoding algorithms for Reed–Solomon codes—part I: an algebraic soft-in hard-out chase decoder. *IEEE Trans Inform Theory*, 56(3):945-959. https://doi.org/10.1109/TIT.2009.2039073

Berlekamp E, 2015. Algebraic Coding Theory. World Scientific, London, UK.

Cho SG, Kim D, Choi J, et al., 2014. Block-wise concatenated BCH codes for NAND flash memories. *IEEE Trans Commun*, 62(4):1164-1177. https://doi.org/10.1109/TCOMM.2014.021514.130287

Hu GH, Sha J, Wang ZF, 2017. High-speed parallel LFSR architectures based on improved state-space transformations. *IEEE Trans Very Large Scale Integr Syst*, 25(3):1159-1163. https://doi.org/10.1109/TVLSI.2016.2608921

Jung J, Yoo H, Lee Y, et al., 2015. Efficient parallel architecture for linear feedback shift registers. *IEEE Trans Circ Syst II*, 62(11):1068-1072. https://doi.org/10.1109/TCSII.2015.2456294

Kim D, Narayanan KR, Ha J, 2018. Symmetric block-wise concatenated BCH codes for NAND flash memories. *IEEE Trans Commun*, 66(10):4365-4380. https://doi.org/10.1109/TCOMM.2018.2839606

Kim J, Sung W, 2012. Low-energy error correction of NAND flash memory through soft-decision decoding. *EURASIP J Adv Signal Process*, 2012(1):195. https://doi.org/10.1186/1687-6180-2012-195

Kumar HP, Sripati U, Shetty KR, 2012. High-speed and parallel approach for decoding of binary BCH codes with application to flash memory devices. *Int J Electron*, 99(5):683-693. https://doi.org/10.1080/00207217.2011.643498

Massey J, 1969. Shift-register synthesis and BCH decoding. *IEEE Trans Inform Theory*, 15(1):122-127. https://doi.org/10.1109/TIT.1969.1054260

Moon TK, 2005. Error Correction Coding: Mathematical Methods and Algorithms. John Wiley & Sons, Inc., Hoboken, USA.

Neubauer A, Freudenberger J, Kühn V, 2007. Coding Theory: Algorithms, Architectures, and Applications. John Wiley & Sons, Ltd., Chichester, UK.

Paar C, 1996. A new architecture for a parallel finite field multiplier with low complexity based on composite fields. *IEEE Trans Comput*, 45(7):856-861. https://doi.org/10.1109/12.508323

Pandian KKS, Ray KC, 2015. Five decade evolution of feedback shift register: algorithms, architectures and applications. *Int J Commun Netw Distr Syst*, 15(2-3):279. https://doi.org/10.1504/IJCNDS.2015.070974

Pei TB, Zukowski C, 1992. High-speed parallel CRC circuits in VLSI. *IEEE Trans Commun*, 40(4):563-657. https://doi.org/10.1109/26.141415

Shieh MD, Sheu MH, Chen CH, et al., 2001. A systematic approach for parallel CRC computations. *J Inform Sci Eng*, 17(3):445-461. https://doi.org/10.6688/JISE.2001.17.3.3

Unal B, Akoglu A, Ghaffari F, et al., 2018. Hardware implementation and performance analysis of resource efficient probabilistic hard decision LDPC decoders. *IEEE Trans Circ Syst I*, 65(9):3074-3084. https://doi.org/10.1109/TCSI.2018.2815008

Xu FX, Liu Y, Liu YQ, et al., 2013. Design and implementation of mode reconfigurable NAND flash error correcting system. *J Centr South Univ Sci Technol*, 44(5):1918-1925 (in Chinese).

Yang CG, Emre Y, Chakrabarti C, 2012. Product code schemes for error correction in MLC NAND flash memories. *IEEE Trans Very Large Scale Integr Syst*, 20(12):2302-2314. https://doi.org/10.1109/TVLSI.2011.2174389

Zhang M, Wu F, Xie CS, 2015. A novel optimization algorithm for Chien search of BCH codes in NAND flash memory devices. IEEE Int Conf on Networking, Architecture and Storage, p.106-111. https://doi.org/10.1109/NAS.2015.7255204

Zhang XM, 2019. A low-power parallel architecture for linear feedback shift registers. *IEEE Trans Circ Syst II*, 66(3):412-416. https://doi.org/10.1109/TCSII.2018.2860934

Zhang XM, Parhi KK, 2005. High-speed architectures for parallel long BCH encoders. *IEEE Trans Very Large Scale Integr Syst*, 13(7):872-877. https://doi.org/10.1109/TVLSI.2005.850125