Design and Implementation of ECC Module Based on BCH Code in SSD

Yifei Niu, Songyan Liu, Yanlin Chen, Xiaowen Wang and Huan Liu

***Abstract*—**Error Correction Code (ECC) is an effective method to ensure the correctness of data stored in Solid State Disk (SSD). At present, Bose, Chaudhuri, Hocquenghem (BCH) code is the most widely used in ECC. However, how to improve performance of encoding and decoding has always been a problem. A new pipeline operation is proposed by this paper to improve the performance of ECC module based on BCH code in SSD. Pipeline operation is used in I/O transmission, and the ECC process submerges in pipeline. The 64-bit parallel architecture is adopted to complete the encoding. The 3-stage pipeline structure is adopted in decoding. Therefore, the efficiency of encoding and decoding is improved and the latency is reduced. ECC module is able to support multiple error correction capabilities. The capability is configurable to 24-bit, 40-bit, and 56-bit for 512Bytes. The data throughput of the ECC module can reach 7.68Gbps.

***Keywords*—**BCH code, Decoding, ECC, Encoding, SSD.

I. Introduction

AS the data processing speed of the Central Processing Unit (CPU) is getting more and more fast, the performance gap between the traditional Hard Disk Drive (HDD) and CPU is also increasing [1]. SSD based on NAND Flash has characteristics of fast reading and writing speed as well as low power consumption compared with HDD, which replaces HDD gradually [2]. Flash bit in SSD may flip due to program/erase (P/E) cycles, read disturb, program disturb, data retention, etc., so that the data stored in flash will be incorrect [3]. Similar to magnetic storage and optical storage, flash also requires error control technology to ensure the integrity and reliability of data [4]. To further increase storage density, MLC (Multi-Level Cell) and TLC (Triple-Level Cell) are becoming mainstream [5].

The earliest ECC code used in NAND Flash is Hamming code. Its principle and implementation are simple, but only single bit error can be corrected. As the Hamming code is not enough to process a higher bit error rate, the industry began to introduce RS (Reed-Solomn) code as ECC code. The RS code has strong ability to correct burst errors, whereas NAND Flash is more likely to occur random independent errors. However, the errors within NAND flash are properly solved with the BCH code [6]. BCH code is the most common and wide ECC code for NAND Flash [7].

BCH code is an important type of cyclic code. Previous papers accomplished plentiful research on NAND Flash. Arul K. Subbiah [8] presents a novel method to reduce the area of the BCH multimode encoder based on a re-encoding scheme. Ping Chen [9] implements a high performance low complexity ECC module circuit for SSD controllers by BCH code. Sheyang Ning [10] proposes an advanced bit flip scheme to correct major program errors. Byeonggil Park [11] presents a novel folding technique for BCH decoders, the regularly structured GF multiplier which is efficiently folded to reduce the complexity and the critical delay. In previous papers, almost all of the authors implement by verilog language or other applications, and the parallel bits are 8, 16 or 32 bit. This article implements ECC module based on FPGA (Field Programmable Gata Array) and tests in practical SSD. The parallel bits are 64. The reading and writing pipeline are adopted in I/O transmission, and ECC is completed in I/O transmission, so as to further improve the performance of the system.

The paper is organized as follows. Section Ⅱ presents BCH encoding and decoding design within ECC module. Section III explains the hardware implementation. Section ⅣdeDepict the test results. Finally, section V is the conclusion.

II. BCH Encoding and Decoding

A. BCH Encoding

For a BCH (n, k, t) code, where the code length, message length and maximum error correction number are denoted by n, k, and t, the BCH code is calculated by equation (1) and (2) in Galois Filed(GF):

$$c(x) = x^{n-k}m(x) + r(x) \tag{1}$$

$$r(x) = x^{n-k}m(x) \bmod g(x) \tag{2}$$

Where $m(x) = m_{k-1}x^{k-1} + m_{k-2}x^{k-2} + ... + m_1 x + m_0$ stand message polynomial, $c(x)$ is code polynomial, $g(x)$ means generator polynomial and $r(x)$ is the remainder polynomial of BCH code.

Yifei Niu is with School of Electronic Engineering, Heilongjiang University, Harbin 150000, Heilongjiang, China (e-mail: 2171304@s.hlju.edu.cn).

Songyan Liu is with School of Electronic Engineering, Heilongjiang University, Harbin 150000, Heilongjiang, China (corresponding author; e-mail: liusongyan@ hlju.edu.cn).

Yanlin Chen is with School of Electronic Engineering, Heilongjiang University, Harbin 150000, Heilongjiang, China.

Xiaowen Wang is with School of Electronic Engineering, Heilongjiang University, Harbin 150000, Heilongjiang, China.

Huan Liu is with School of Electronic Engineering, Heilongjiang University, Harbin 150000, Heilongjiang, China.

Since the BCH code is cycle code, the Linear Feedback Shift Register (LFSR) can be used to implement the BCH encoder. The circuit of serial LFSR is as Fig. 1, where $g_0 = 1$, $g_{n-k} = 1$; $g_i = 1(1 \le i \le n-k-1)$ presents the line is connect, $g_i = 0$ means the line is disconnected. Initially, the switch K$_1$ is closed, K$_2$ is off, and when the LFSR outputs the check bit, K$_1$ is switched off and K$_2$ is closed. The information bits are input from the right edge, which is equivalent to the dividend multiplied by x^{n-k} and then divided by $g(x)$. The hardware implementation of traditional serial encoder is relatively simple, but the clock cycle of encoding is long, which cannot meet the application requirements of NAND Flash, so that we designed the encoder with parallel structure.

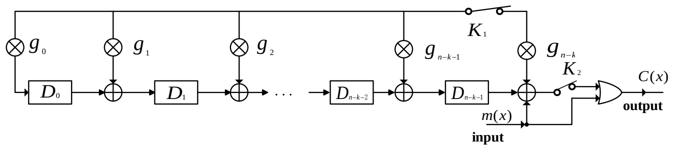

Fig. 1. Circuit of serial LFSR

In this paper, the unit of checking data is sector, the system provides several different ECC abilities, including 24bit,40bit and 56 bit. The length of checksum for different ECC ability is shown in Table 1.

Table 1 Length of checksum corresponding to different ECC ability

Error Correction (bits)	Galois field GF(2^m)	Length of checksum (Byte)	Coding efficiency
24	GF(2^{13})	39	0.93
40	GF(2^{13})	65	0.89
56	GF(2^{13})	91	0.85

B. BCH Decoding

For the transmission code $c(x)$, it is defined that the received code is $R(x)$. If the original code is interfered during transmission, an error pattern of the following equation (3) will be generated:

$$e(x) = e_0 + e_1 x + ... + e_{n-1} x^{n-1} \tag{3}$$

Therefore, the receiving polynomial $R(x)$ as equation(4) can be represented by the transmission code and error pattern.

$$R(x) = e(x) + c(x) \tag{4}$$

The decoding of BCH code is as follows:

(1) Calculating syndrome polynomial $S_1, S_2, ..., S_{2t}$ by $R(x)$, then judging whether the received code has an error according to the value of syndrome. The circuit of calculating syndrome is as Fig. 2;

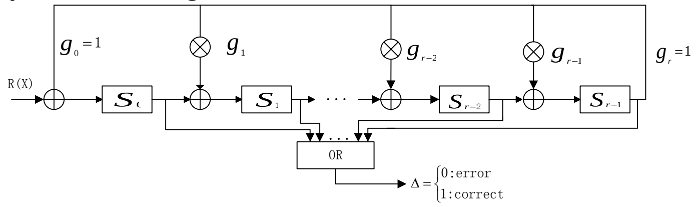

Fig. 2. The circuit of calculating syndromes

(2) Calculating the coefficient of error location polynomial $\sigma(x)$ according to syndrome;

(3) Calculating the roots of error location polynomial $\sigma(x)$ and then getting the error locations and correct errors.

Syndrome polynomial is calculated by equation (5):

$$S = (S_1, S_2, ..., S_{2t}) = R \times H^T \tag{5}$$

It is provided that $e(x)$ has v errors at positions j_1, j_2, …, j_v, where $0 \le j_1 < j_2 < ... < j_v < n$, and then we can get $e(x)$ as equation (6):

$$e(x) = x^{j_1} + x^{j_2} + ... + x^{j_v} \tag{6}$$

For calculating the syndrome, it is to substitute $2t$ roots $\alpha^i (i = 1, 2, ..., 2t)$ of $g(x)$ into $R(x)$, and then getting equation (7):

$$S_i = R(\alpha^i) = R_0 + R_1 \alpha^i + ... + R_{n-1} (\alpha^i)^{n-1} \tag{7}$$

Since $R(\alpha^i) = e(\alpha^i) + c(\alpha^i)$, $c(\alpha^i) = 0$,we can get equation (8):

$$S_i = e(\alpha^i) = (\alpha^{j_1})^i + (\alpha^{j_2})^i + ... + (\alpha^{j_v})^i \tag{8}$$

For $1 \le l \le v$,we define $\beta_l = \alpha^{jl}$, and equation (8) can be abbreviated as equation(9):

$$S_i = \beta_1^i + \beta_2^i + ... + \beta_v^i \tag{9}$$

Define a polynomial $\sigma(x)$ of degree v on $GF(2^m)$:

$$\begin{aligned} \sigma(x) &= (1+\beta_1 x)(1+\beta_2 x)....(1+\beta_v x) \\ &= \sigma_0 + \sigma_1 x + ... + \sigma_v x \end{aligned} \tag{10}$$

Where σ_i is the coefficient of error location polynomial, if we know the value of β_l, we can get the error location in polynomial $R(x)$, and β^{-1}, β^{-2},…, β^{-v} are the roots of $\sigma(x)$.

Since the coefficient of syndrome polynomial S_1, S_2, …, S_{2t} and the error position polynomial $\sigma(x)$ are linked by the number of error positions β_1, β_2,…, β^{-v}, the following equation (11) can be obtained by associating the polynomial $\sigma(x)$ and $R(x)$.

$$\begin{aligned} & S_1 + \sigma_1 = 0 \\ & S_2 + \sigma_1 S_1 + 2\sigma_2 = 0 \\ & ... \\ & S_v + \sigma_1 S_{v-1} + \sigma_2 S_{v-2} + ... + \sigma_{v-1} S_1 + v\sigma_v = 0 \\ & ... \end{aligned} \tag{11}$$

Equation (11) is also named Newton identity. We can find the identity that matches Newton identity and the time of numbers are less, that is $\sigma(x)$. The $\sigma(x)$ can be calculated by σ_i, and we can determine the error location by calculating the root of $\sigma(x)$.

It can be implemented by iteratively through an efficient algebraic hard decision decoding algorithm, which is usually called Berlekamp-Massey (BM) algorithm [12]. The process of BM algorithm is as follows:

Firstly, determining the appropriate $\sigma^{(1)}(x)$ to satisfy the first equation of Newton identity. Then, provided that σ_2 is equal to σ_1 ,we need to judge if this matches the second equation of Newton identity. If it is satisfied, we can get $\sigma^{(2)}(x)=\sigma^{(1)}(x)$. Otherwise, a correction value is added in $\sigma^{(1)}(x)$ to make the second equation be established. $\sigma^{(2t)}(x)$ is obtained after $2t$ iterations, and then $\sigma(x)=\sigma^{(2t)}(x)$ can be determined. The reciprocal of roots for error location polynomial are error locations. From the mathematical perspective, the problem of finding the error location has been solved, but seeking root is not so easy to achieve in engineering.

Qian Wentian proposed a method of calculating roots in 1964, which was called Chien search, so that lots of problems in engineering were solved. The idea of Chien algorithm is: There is a primitive element a in $GF(2^m)$, and the reciprocal of $1, a, a^2, ..., a^{n-1}$ are substituted into $\sigma(x)$, we can get equation (12):

$$\sigma(a^{-i})=\sigma_0+\sigma_1 a^{-i}+\sigma_2(a^{-2i})+...+\sigma_t(a^{-ti}) \quad (12)$$

Where a^{-i} is the root of $\sigma(x)$ if $\sigma(a^{-i})=0$, and then whether the i -th bit in the received code has an error will be known. If there is a bit error in the i -th bit, and the code of this position is corrected, the result of the decoding is $R(x)+e(x)$. If it is not zero, it indicates that there is no error in the corresponding position.

III. Hardware Implementation

The ECC module of this paper is based on SSD. The SSD is mainly composed of a flash array and a controller. The flash array is divided into 3 logical channels, each of which contains a flash array of 8 NAND Flash chips, a cortex-M1 processor and an NFI (NAND Flash Interface) unit. The structure of hardware is shown in Fig. 3, where NAC is NAND Array Controller, DMAC is Directly Memory Address Controller.

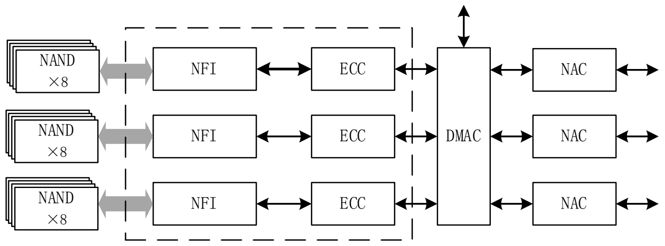

Fig. 3. Structure of hardware

The flash memory chip uses MTL NAND Flash of Micron's MT29F256G08CEEAB, which has a capacity of 32G per chip and contains 2 wafers. Each wafer contains 2 planes. Each plane contains 1024 blocks, and each block contains 512 pages. The page has a data space of 16KB and a reserved space of 1216 bytes. Each NAND Flash is 8-bit width, and ECC operates 8 NAND Flash in parallel, each is called a physical channel.

In this paper, the ECC module includes encoding and decoding. Encoding and decoding are completed in a byte stream. When writing data, original data will be written to NFI and LFSR respectively. LFSR will calculate check code, then add binary to the behind of original data, and finally write it to NAND Flash. When reading data from NAND Flash, it will be read to page buffer and LFSR respectively, then it will be detected whether an error has occurred by ECC error detection register. If an error occurs, the syndromes are popped from LFSR, and the error locations and numbers are calculated by ECC core in the end. Next, ECC correction reads and corrects the errors from page buffer and returns the status to the ECC correction FIFO. There are 8 page buffers, and one of the page buffers is for a physical channel. The structure of ECC for one logical channel is shown as Fig. 4.

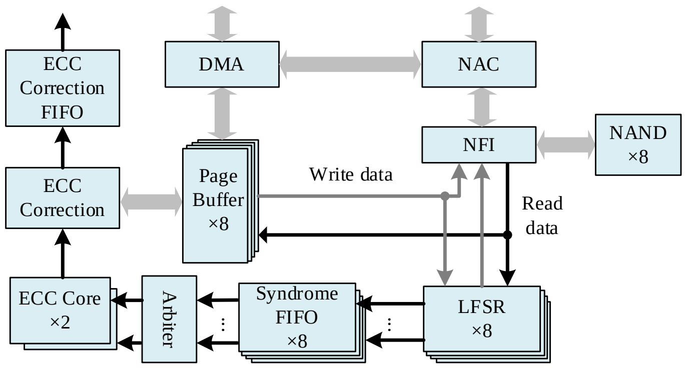

Fig. 4. The structure of ECC for one logical channel

A. Write Pipeline

The unit of writing data is page, and 8 physical channels can be written in parallel at the same time. Therefore, the MC (Master Controller) waits for message in data buffer to reach the size of 8 pages, and then performs the write operation.

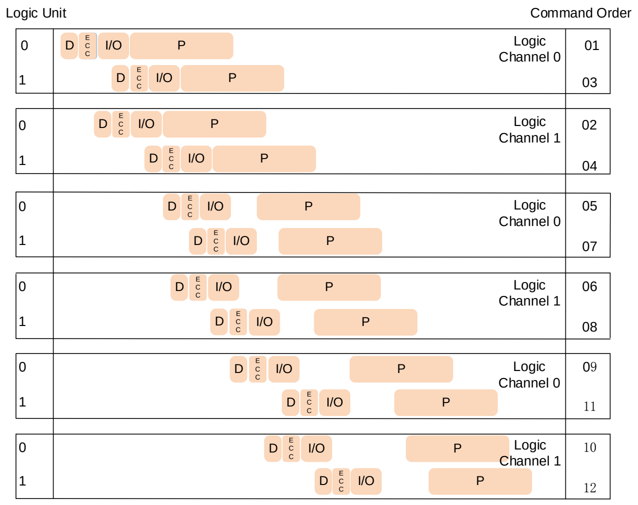

Fig. 5. Write Pipeline

Internal programming operation can be performed simultaneously between different logic cells, but I/O buses of chip data transmission are shared. Therefore, after executing the I/O transmission of a command and starting the programming operation, NAC does not wait for the end of programming. However, it starts preparing to receive the next command, and ECC is completed in I/O transmission. When the same logical unit receives the second write command, it waits for the completion of previous operation and checks whether the previous operation was successful. To further optimize performance, we used a write operation with a cache. Fig. 5 shows the results of the pipeline of the write operation, it takes two channels for example. Where D represents the process of DMA transmission from the message processing unit to page buffer, and I/O represents the process of data transmission in the ONFI bus, ECC presents the ECC process, P represent the programming process of internal flash unit.

From Fig. 5, we can see if we use internal cache buffer of flash chip, the I/O transmission and ECC would be hidden in the programming. In the main stage of writing, the internal programming of chip takes the longest time and becomes a performance bottleneck. The time of DMA, I/O and ECC is short, which can be hidden in the pipeline. Finally, the overall performance of writing will depend on the bandwidth of internal programming, which can be expressed as equation (13)：

$$P_{write} = N_{\log icchns} \times N_{LUNs} \times \frac{C_{page} \times N_{physchns}}{T_{program}} (MB/s) \quad (13)$$

Where $N_{\log ic\ chns}$ represents the number of logical channels, N_{LUNs} represents the number of logical units in each chip, C_{page} presents the size of valid value in a page, $N_{phys\ chns}$ represents the number of physical channels in each logical channel, $T_{program}$ represents the time spent in internal page programming in milliseconds.

B. Read Pipeline

Unlike the internal programming process of writing operation, which is the most time-consuming and becomes a performance bottleneck. The internal read and DMA process in the read operation are fast, and the I/O transfer process of chip becomes a performance bottleneck. In other words, the overall read performance will depend on the chip's I/O bandwidth. Fig. 6 illustrates the pipeline process of reading operation in two logical channels.

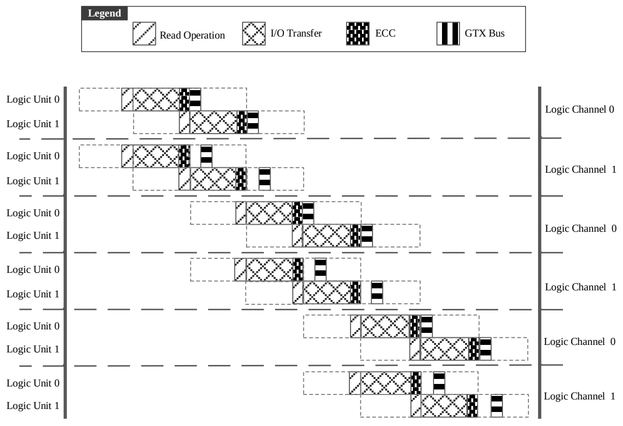

Fig. 6 Read Pipeline

Since two logic units can perform internal read operation simultaneously, this process is hidden during the I/O transmission. For DMA and ECC, the time they cost is less than I/O transfer, so they are hidden in I/O transfer.

However, there is still a factor to be considered for the reading operation. Each sector in the flash chip also stores redundancy check information such as ECC and CRC, and the actual amount of data obtained is less than the amount of data passing through the I/O of the flash chip. Ultimately, the overall read performance will be expressed as equation (14):

$$P_{read} = N_{\log ic\ chns} \times N_{phys\ chns} \times B_{I/O} \times \frac{L_{data}}{L_{data} + L_{OOB} + L_{CRC} + L_{ECC}} (MB/s) \quad (14)$$

Where $N_{\log ic\ chns}$ represents the number of logical channels and $N_{phys\ chns}$ denotes the number of physical channels in each logical channel. $B_{I/O}$ represents the bandwidth of chip bus, the length of data, out-of-band information, CRC and ECC check code in each sector are L_{OOB}, L_{CRC}, and L_{ECC}, respectively.

C. LFSR and Syndrome FIFOs

There are 8 LFSR modules, ECC code and pre-syndromes are generated in them, each module is responsible for one physical channel. The width of a LFSR is 8 bits, and 8 physical channel is parallel. The principle of LFSR has been introduced in previous section.

When programming data, NFI will wait for LFSRs to be idle after all payload data are shifted into LFSRs, and then it will shift out the ECC code from LFSRs and store them to NAND Flash. Similarly, while reading data, NFI will shift all data into LFSRs, including ECC code. After that, when the ECC Core module needs the pre-syndromes to calculate the error numbers and the error locations, NFI will wait for LFSRs to be idle and then shift out the pre-syndromes from LFSRs.

There are 8 syndrome FIFOs, the generated pre-syndromes by LFSR are pushed into them when the ECC correction ability is open and some LFSR finds errors in the receiving data and the read-back page is valid. Each of syndrome FIFOs is 28-bit width and 32-level depth. One FIFO only for a particular LFSR, and a

FIFO is not available for any other LFSRs even it's empty. It will take at least 32 clock cycles to push all pre-syndromes into FIFO, or more clock cycles if any FIFO is not empty at the moment. Then LFSR can process next sector data.

D. Arbiter and ECC Core

In this article, there are 2 ECC cores, however, we have 8 Syndrome FIFOs, so that an arbiter is needed to determine which data will be pushed into ECC core. When arbiter allows ECC core to read data from any of syndrome FIFOs, it also records the request sequence of 2 ECC cores. There are two levels as arbitration mechanism in order to hold the order of incoming operations: If requests arrive at different time, the policy is to determine according to which request comes first; If requests arrive at the same time, it adopts the policy of fixed priority.

The ECC Core will pop pre-syndromes from syndrome FIFOs, and then pre-syndromes will be expanded to syndromes. In an ECC core, there're 3 stages of pipeline, precisely, syndrome expand, Berlekemp and Chien search. The structure of ECC core is depicted in Fig. 7. Finally, calculating error numbers and error locations can be done by any of ECC cores, and error status will be returned to the next module.

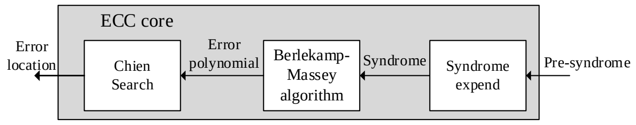

Fig. 7. The structure of ECC core

E. ECC Correction and ECC Correction FIFO

The error information from 2 ECC cores will be pushed into ECC correction. Then ECC correction reads error data from page buffer and corrects them. The error correction sequence is consistent with the order allowed by arbiter. After processing all error data of one sector, ECC correction module will push an entry to the ECC correction FIFO.

The ECC Correction FIFO is 16-level depth, and each FIFO entry reports correction status of one sector for one physical channel after it finishes ECC process of a sector. And the messages of ECC Correction FIFO can be read by software. The FIFO contains command pool index, lane number and sector number, ECC warning flag, error bit quantity, as well as finish flag.

F. ECC data pipeline

There're several stages of pipeline for the total process of ECC module. Firstly, when the 8 syndrome FIFOs are all empty, syndromes from 8 LFSR modules will be pushed into 8 syndrome FIFOs. If the syndromes of previous sector in 8 channels have been entered into ECC cores, next syndrome data will be shift to syndrome FIFOs. This rule reduces the complexity of design at the cost of efficiency.

Secondly, the arbiter will grant an ECC core to read the data from one of 8 syndrome FIFOs, when any of ECC cores is available and some of 8 syndrome FIFOs are not empty. When other data of channels have flowed into ECC cores, some syndromes in some channels will remain in the FIFOs. Because we have 8 syndrome FIFOs, among which there are only 2 ECC cores.

Thirdly, the ECC core will calculate error numbers and error locations. An ECC core may receive syndrome data of a few channels in a short time. However, they may not be able to receive syndromes from syndrome FIFOs when they are all busy in correcting.

IV. Test

For the design proposed in this paper, it has calculated in theory and tested on the SSD in practice. The theoretical and test value of reading and writing performance in different error correction capabilities are shown in Fig. 8 and Fig. 9.

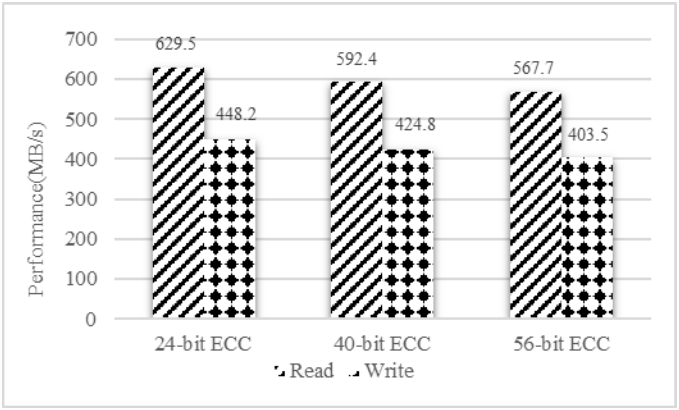

Fig. 8. Theoretical value for reading and writing performance in different error correction capabilities

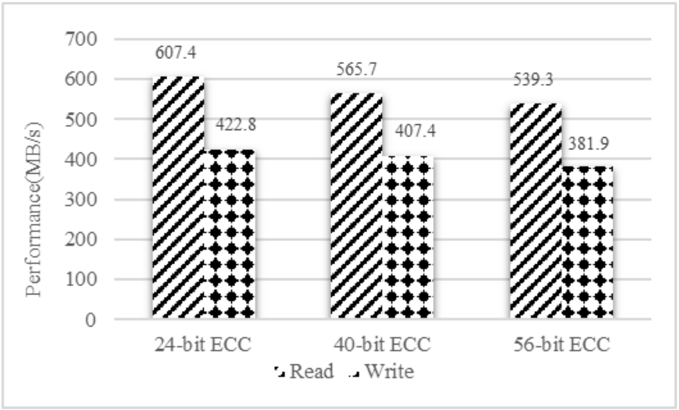

Fig. 9. Test value for reading and writing performance in different error correction capabilities

As shown in Fig. 8 and Fig. 9, reading and writing speed will decrease along with the increase of error correction bits, and the test value is lower than the theoretical one due to system power consumption, etc. From the result analysis, the performance bottleneck of writing operation is the bandwidth of programming operation. The performance bottleneck of reading data is the bandwidth of data transmission of the chip bus, and the ECC operation is submerged in pipeline, which is not the performance bottleneck of the system. Here is a 40-bit error correction capability used for test in SSD, the test results are compared with other papers as shown in Table 2.

Table 2 Performance Comparison

Performance	This Paper	Paper[9]	Paper[12]

Correctable Errors	40bit/512B	72bit/1KB	245bit/4KB
Parallelism	64	32	32
Data Throughput	7.68Gbps	3.2Gbps	6.37Gbps
I/O Write Bandwidth(MB/s)	407.4	N.A	N.A
I/O Read Bandwidth(MB/s)	565.7	N.A	N.A
Decoding Latency(us)	1.05	2.92	8.19

Paper [9] applys a 32-bit parallel architecture in encoding and 2-stage pipeline to accomplish the decoder with strong error correction capability and long decoding latency. For paper [12], it proposes a parity-check block as an additional decoding step, and a novel memory based syndrome updating method effectively improves the energy efficiency as well as the decoding latency. Despite the decoding latency is short, the error correction capability is not strong in such a method. As shown in Table 2, by increasing the degree of parallelism and using pipeline scheduling adequately, we can effectively increase data throughput and reduce decoding latency.

V. Conclusion

This paper improves the reading and writing performance of the system by designing the pipeline of the ECC module and increasing the degree of parallelism. The ECC module can support different error correction bits, having been tested on the physical SSD, whose test results have reached the anticipated effect. It supports 7.68Gbps with 64-bit parallel encoding and 3-stage pipeline decoding structure. Planar-level parallelism is not used in this design because of the more demanding constraints, which will result in the increasement about the complexity of system, overhead of pipeline scheduling and the power consumption of system. For writing operation, if the chip is shared by I/O bus and DMA between the planes, the time of I/O, ECC and DMA will not be submerged adequately by the programming operation. Therefore, we will improve the pipeline scheduling mode to achieve the plane level in order to further improve system performance in the further work. Considering there are only two ECC cores, the decoding efficiency may be impacted, we will also perfect this part to further improve the performance of the system.

References

[1] M. Lin, R. Chen, J. Xiong, "Efficient Sequential Data Migration Scheme Considering Dying Data for HDD/SSD Hybrid Storage Systems," *IEEE Access*, vol. 5, pp. 23366–23368, Nov. 2017.

[2] Jung. Kyu. Park, Yunjung. Seo, Jaeho. Kim, "Solid State Cache Management Scheme for Improving I/O Performance of Hard Disk Drive," *2018 IEEE Int. Conf. on Consumer Electronics, Las Vegas, NV, USA*, pp. 1-4.

[3] Yougoo. Lee, "Hardware optimizations of hard-decision ECC decoders for MLC NAND flash memories," *2015 Int. SoC Design Conf.*, Gyungju, South Korea, pp. 133-134.

[4] J. Hsieh, K. Hung, H. Li, "A Hardware.Efficient BCH Encoder Design," *2016 Int. Conf. on Consumer Electronics-Taiwan*, pp. 1-2.

[5] Bo Mao, Suzhen Wu, and Lide Duan, "Improving the SSD Performance by Exploiting Request Characteristics and Internal Parallelism," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, Feb. 2018.

[6] Chien-Chung Ho, Yu-Ping Liu, Yuan-Hao Chang, and Tei-Wei Kuo, "Antiwear Leveling Design for SSDs With Hybrid ECC Capability," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, pp. 488-501, Feb. 2017.

[7] Sheyang Ning, Tomoko Ogura Iwasaki, and Shuhei Tanakamaru, "Reset-Check-Reverse-Flag Scheme on NRAM With 50% Bit Error Rate or 35% Parity Overhead and 16% Decoding Latency Reductions for Read-Intensive Storage Class Memory," *IEEE Journal of Solid-State Circuits*, vol. 51, pp. 1938-1951, Aug. 2016.

[8] Arul K. Subbiah, Prof. Tokunbo Ogunfunmi, "Area-effcient re-encoding scheme for NAND Flash Memory with multimode BCH Error correction" *2018 IEEE Int. Symposium on Circuits and Systems*, Florence, Italy, pp. 1-5.

[9] Ping Chen, Chun Zhang, Hanjun Jiang, and Zhihua Wang, "High Performance Low Complexity BCH Error Correction Circuit For SSD Controllers," *2015 IEEE Int. Conf. on Electron Devices and Solid-State Circuits*, Singapore, pp. 217–220.

[10] Sheyang Ning, "Advanced Bit Flip Concatenates BCH Code Demonstrates 0.93% Correctable BER and Faster Decoding on (36 864, 32 768) Emerging Memories," *IEEE Transactions on Circuits and Systems I: Regular Papers,* vol. 65, pp. 4404-4412, Dec. 2018.

[11] Byeonggil Park, Seungyong An, Jongsun Park, and Youngjoo Lee. "Novel Folded-KES Architecture for High-Speed and Area-Efficient BCH Decoders," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 64, pp. 535-539, May, 2017.

[12] Jaehwan Jung, In-Cheol Park, and Youngjoo Lee," A 2.4pJ/bit, 6.37Gb/s SPC-enhanced BC-BCH Decoder in 65nm CMOS for NAND Flash Storage Systems," *2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC),* Jeju, South Korea, pp. 329-330.

Yifei Niu was born in January, 1995. She received the bachelor degree in Electronic Information Science and Technology from Heilongjiang University of China. Currently, she is a graduate student at School of Electronic Engineering, Heilongjiang University, China. Her major research interest in embedded system.

Songyan Liu was born in February, 1969. He received the PhD degree in Microelectronic Science and Technology from Harbin Institute of Technology, China. He has worked as an engineer in well-known companies at home and abroad. Currently, he is a professor at School of Electronic Engineering, Heilongjiang University, China. He has published many papers in related journals.

Yanlin Chen was born in April, 1994. Currently, she is a graduate student at School of Electronic Engineering, Heilongjiang University, China. Her major research interest in embedded system.

Xiaowen Wang was born in September, 1995. Currently, she is a graduate student at School of Electronic Engineering, Heilongjiang University, China. His major research interest in embedded system.

Huan Liu was born in June, 1991. Currently, she is a graduate student at School of Electronic Engineering, Heilongjiang University, China. His major research interest in embedded system.