International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:4, No:12, 2010

Implementation of a Reed-Solomon Code as an ECC in Yet Another Flash File System

Sungjoon Sim, Soongyu Kwon, Dongjae Song and Jong Tae Kim

***Abstract*—**Flash memory has become an important storage device in many embedded systems because of its high performance, low power consumption and shock resistance. Multi-level cell (MLC) is developed as an effective solution for reducing the cost and increasing the storage density in recent years. However, most of flash file system cannot handle the error correction sufficiently. To correct more errors for MLC, we implement Reed-Solomon (RS) code to YAFFS, what is widely used for flash-based file system. RS code has longer computing time but the correcting ability is much higher than that of Hamming code.

***Keywords*—**Reed-Solomon, NAND flash memory, YAFFS, Error Correcting Code, Flash File System

I. Introduction

FLASH memory has been increasingly used in many embedded systems since NAND flash memory's performance and capacity have been grown. Hard disk drive based storage devices can save mass data with low price, but it has disadvantages such as huge size, less durability for the shock, high power consumption and long response time. Flash memory has been appropriated for the small portable devices because it overcomes these disadvantages. But it has a low reliability problem.

Multi-level cell (MLC) [1] flash memories have been replaced single-level cell (SLC) flash memories in recent years. Because multiple bits are stored per memory cell in MLC flash memory, the probability of the error occurrence is high. That means the larger capacity what flash memory has, the more error the system can has. Furthermore, the reliability of MLC may become more important even if it assume that the reliability is same as that of SLC because of some points like erase endurance and soft error.

That is why the powerful error correcting code (ECC) is much needed than SLC flash memory. Widely used flash file systems such as Yet Another Flash File System (YAFFS) [2] and Jounalling Flash File System (JFFS2) [3] depend on the controller's ECC or generate a small ECC on the spare area for error correction [4]. Generally, single-bit error correcting codes such as Hamming codes, are used for SLC. Single-bit error correcting codes are no longer sufficient for MLC [5]. To handle this problem, ECCs based on non-binary symbol are needed. These codes are specialized to correct burst errors.

In this paper, the result that Reed-Solomon (RS) code is applied for ECC in YAFFS instead of Hamming code. RS code is very powerful error correction code to handle burst error since it is non-binary code. RS code can correct a symbol error, which includes *m* bits, in contrast to correction ability of Hamming code which can correct only one bit error in 256 bytes data block. Because YAFFS uses spare area to store tag and ECC parities, the number of parity for ECC would be limited. To fit the number of parities, we divided 256 bytes data block by 2. Two each RS codes, they are applied at each divided block, can handle one symbol error. Implemented RS code can correct up to 12 bit error in 256 byte, 24 bit error in a page.

The rest of the paper is organized as follows. Previous flash file systems are shown in Section 2. This section also describes their advantages and disadvantages. Section 3 presents our scheme of RS encoder and decoder. Simulation results and hardware tests are provided in Section 4. Section 5 is the conclusion.

II. Flash File Systems

A. Using Flash Translation Layer

The widely used existing file systems like FAT cannot be applied in flash memory because the architecture of flash memory. Some flash file systems include a layer called Flash Translation Layer (FTL) [6]. The main purpose of FTL is that existing file system can access flash memory as though flash memory is general block devices like hard disk drive. The Concept of page mapping is as shown in Fig. 1. When file system or user access to a stored data, the relevant address has to be arrangement. NAND flash memory has a fatal disadvantage what is bad block. Physical address has to tangle because of bad block. When file system or user access the sequential data, they look like be arranged well because FTL has address mapping table to solve this problem. And also FTL do garbage collection, wear-leveling and bad block management to use flash memory appropriately. NAND flash memory has limited the number of erase. Usually up to 100

Sungjoon Sim is with the Department of Electrical and Computer Engineering, Sungkyunkwan University, Suwon, Gyeonggi-do 440-746, Republic of Korea (phone: +82-31-290-7173; fax: +82-31-299-4613; e-mail: joonii63@ skku.edu).

Soongyu Kwon is with the Department of Mobile Systems Engineering, Sungkyunkwan University (e-mail: caesar01@skku.edu).

Dongjae Song is with the Department of Electrical and Computer Engineering, Sungkyunkwan University (e-mail: dongjae.song@gmail.com).

Jong Tae Kim is with the Department of Mobile Systems Engineering and the Department of Electrical and Computer Engineering, Sungkyunkwan University (e-mail: jtkim@ skku.edu).

thousand in SLC, 10 thousand in MLC is the limit. A page or data block has to be broken when a lot of erase occur at the particular page. FTL has wear-leveling algorithm to prevent this situation is happen. Wear-leveling can partition pages to the least used pages. Garbage collection is optimizing algorithm to use NAND flash memory. It prevent to waste resources by arranging useless pages and using pages.

No matter how flash file system is constructed using any mechanism, it does not matter because FTL switch the flash file system to general file system like FAT. This method has some advantages that easy to implement and suitable to general file system, however it has also disadvantages like its low performance because it is not optimized.

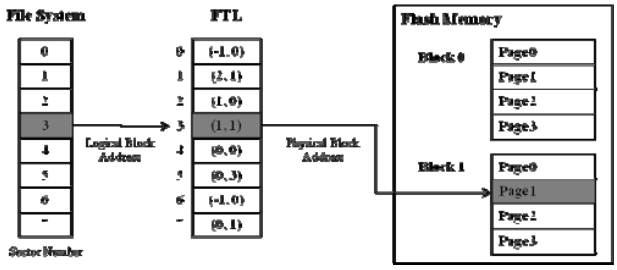

Fig. 1 Page mapping method of FTL

B. Using Log Structure Mechanism

To avoid these problems of FTL, combined flash file systems are developed. These file systems are using log-structured mechanism [7] because of the disadvantage that flash memory cannot be overwritten on the page which has a data already. Basically, data can be written on any empty page. In the situation that the page that any data is written, however, data can be written the page after erase data block what include the page instead data are written the page directly.

Because of a characteristic of flash memory, however, there is a weakness that flash memory cannot be used anymore if one particular block is erased many times to overwrite data. Therefore, log-structured file system is widely used to combined flash file systems because of the characteristic that file system erases data blocks at a time after collecting non-using blocks and writes on empty pages through garbage collection and wear-leveling. We can improve performance of flash file system from wear-leveling that is one of the properties of log-structure if we use log-structured file system. These combined flash file systems merely have a disadvantage that the early mount time is slow and the amount of memory use is a lot. JFFS, JFFS2 and YAFFS are representative example of combined flash file systems. Fig. 2 shows the difference between the file system with FTL and combined flash file system briefly. (a) shows the file system using FTL. The FTL is between general file systems, such as FAT, NTFS and EXT3, and flash device driver. (b) represents combined flash file system.

JFFS2 is a node based log-structured file system. A node including contents information is contained in every page in JFFS2. JFFS2 only keeps the minimum index in main memory. Because JFFS2 has to build the index each time the file system is mounted, its mount time is slow. YAFFS is also one of the combined flash file systems, with some advantages over JFFS2 such as a small taking RAM space and error correction mechanisms. However YAFFS also has not enough space for ECC parities because it relies on the spare area in each page for both error detection and correction. That cause limiting its correcting ability to detect and correct lots of errors in page or stand the loss if a complete page.

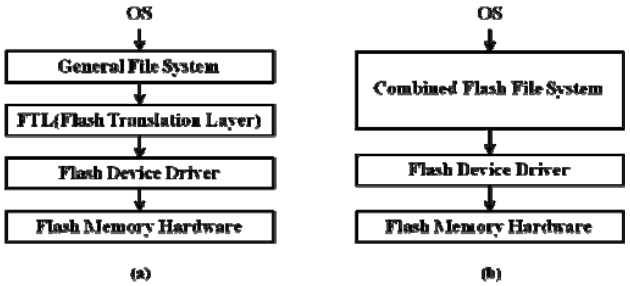

Fig. 2 difference between (a) file system using FTL and (b) combined flash file system

III. Our Scheme of RS Encoder/Decoder

YAFFS uses the spare area on each page to store ECC parities. This ECC can detect two bit errors and correct one bit error per 256 bytes. However its availability is limited by the size of spare area what must also be used to store other information tags. It is impossible to store a code can correct more than one bit error because of the limited size of the spare area. To solve this problem, we apply RS code to YAFFS instead of Hamming code [8].

RS code (n, k) using Galois Field (2^m) can have ($2t$) parity bits from follow equation (1).

$$RS(n,k) = (2^m - 1, 2^m - 1 - 2t) \quad (1)$$

The number of parities for calculating error position and error value is limited because of the flash memory's architecture even though we may correct many errors using RS code. Furthermore, the number of total symbols-sum of data symbols and parity symbols-should be less than 2^m to implement RS code, but the number of data symbols is same as 2^m. So we have to divide a data block by 2 to implement RS code. One is RS(255,253) using GF(2^8) and the other is RS(8,6) using GF(2^4).

First of all, Galois Field (2^m) is needed to implement RS code. Each Galois Field what is needed for presented RS code is generated by using linear feedback shift registers as shown in Fig. 3.

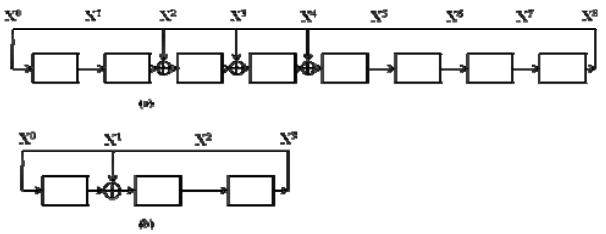

Fig. 3 Linear Feedback Shift Register to generate Galois Field (a) 2^8 (b) 2^4

The primitive polynomial of each Galois Field, which is based on generating method, is represented as following equation (2) and equation (3).

$$GF(2^4) = 1 + X + X^4 \quad (2)$$

$$GF(2^8) = 1 + X^2 + X^3 + X^4 + X^8 \quad (3)$$

We can calculate the syndrome, the position of errors and the value of errors using this GF(2^m).

The partitioning of two RSs in a page is as shown in Fig. 4. 253 bytes of data from 0 to 252 byte number are used for RS(255,253) and 3 bytes of data from 253 to 255 byte number are used for RS(8,6). It applies to next 256 bytes of data in same way.

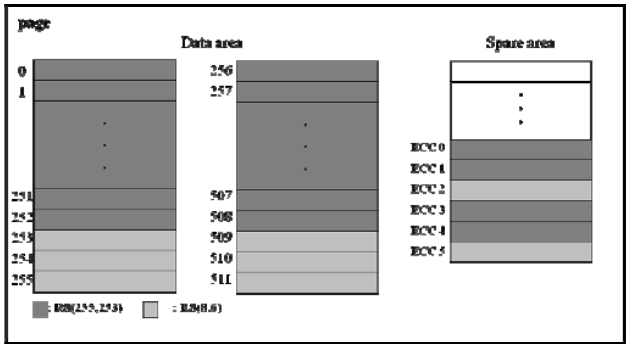

Fig. 4 partitioning two RSs in a page

A. RS Encoder

RS encoder is easy to implement because only two parity symbols are generated in each RS encoder. At first, 253 bytes of data enter the LFSR encoder which is as shown in Fig. 5 (a). Then two bytes of generated parity symbols are stored in ECC0 and ECC1 in spare area. The rest three bytes of data enter the encoder, which is as shown in Fig. 5 (b), then one byte of generated parity symbol is stored in ECC2. In the same way, from 256 to 511 byte number of data handle and parity symbols are stored in ECC3-5 because YAFFS code is built to handle 256 bytes of a page at once. In these calculations, the complex computations what are the product of Galois Fields and the addition of Galois Fields. The addition of Galois Fields is implemented by using exclusive or calculation and the product of Galois Fields is implemented by using the addition of two degrees of Galois fields.

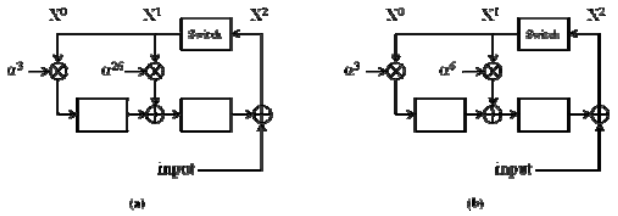

Fig. 5 LFSR encoder for (a) RS(255,253), (b) RS(8,6)

B. RS Decoder

Syndrome value is computed from the data when a page read occur. Two syndromes are come out from the computation in each RS codes in our scheme. That means we can correct one symbol if there one symbol error occurs. Since two RS codes which have different error correcting ability are implemented in a page, we can correct up to four symbols if each error exists in different area what the different RS is covered. Total 24 bit error can be corrected maximum in a page. Its performance is much better than that of Hamming code.

Error existence is checked by using computed syndrome value. There is no error if all syndrome values are zero. Data can be used to read operation if there is no error in the data, but error correction is next stage if there is error. Error correction stage consists of two parts, one is calculating the locations of errors and the other is calculating the values of errors. Only one computing process is needed to find the locations and values of errors because the correction ability of RS code is limited at one symbol. The complex computations such as Euclid methods or Forney's algorithm are not necessary for this reason.

The calculating process of error correction is as follows. At first, an error-locator polynomial can be defined as equation (4).

$$\sigma(X) = 1 + \sigma^1 X \quad (4)$$

The coefficient σ^1 is computed by XOR computation of two syndrome values. Second, every degree of Galois Field which is related to each RS code is substituted to X in the equation. Then the degree of error what the equation is equal to zero is found. Third, the location of the error is computed by calculating the inverse value of the degree. Forth, the value of error is computed by the calculation of divide any syndrome value by the location of error.

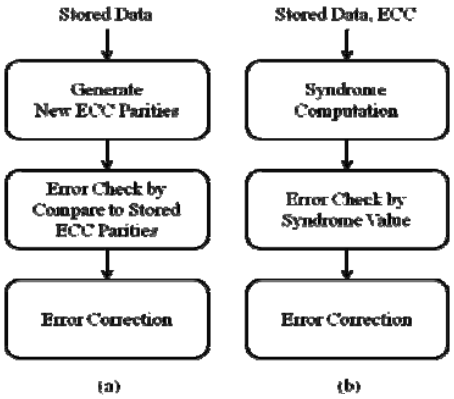

Fig. 6 The difference of process flow between (a) existing decoder and (b) proposed decoder

The difference between the existing decoder of YAFFS code and proposed decoder is as shown in Fig. 6 briefly. The basic concept of the existing decoder is comparing. To check the error exists or not, it generates new parities then compares to stored parities which is in spare area. The main idea of proposed RS decoder is a syndrome. It calculates syndromes to

check the error exists or not and to compute the location and value of the error.

IV. Simulation and Hardware Test Results

The implementation of RS code to YAFFS is constructed by C language as same as YAFFS base code is. We evaluate implemented RS code in two ways. One is computer simulation in the Ububtu 8.04 and the other is measuring read and write computation times to compare Hamming code and RS code in the hardware environment that the Android, including YAFFS, is ported as an operating system.

TABLE I
Time Overhead for Error Correction

Hamming	Encoding	Decoding
No error	30.14	11.56
Error Detected	30.42	22.76
RS	**Encoding**	**Decoding**
No error	360.50	1002.34
Error Detected	360.38	1006.14

*time unit is *u*sec.
*results from handling one page(512 bytes).

Table 1 represents the time overhead for error correction in the computer simulation. Every result is average of 1 million times of simulation. We can recognize that the decoding time of the Hamming code when error detected is nearly twice than that of Hamming code when no error detected, on the other hand the RS code has almost same time overhead between them.

Fig. 7 shows the simulation results of BERs for using Hamming and RS codes. In our simulation, every error is randomly located. RS code is applied to three cases. One is that every error separates from each other and another is 4-bit burst error. The other is 8-bit burst error. As a result, implemented RS code is the most efficient in the case of 8-bit burst error.

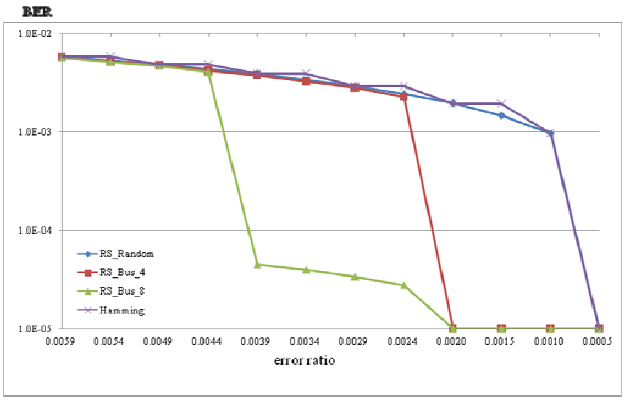

Fig. 7 BER performance for several situations

The environment of hardware test is as follows. The system's processor is S5PC100 from SAMSUNG which is based on ARM Cortex™-A8 and the flash memory is K9F2G08 which is 256MB NAND flash manufactured by SAMSUNG. Android 1.6 is ported as an operating system with linux kernel 2.6.27 and the root file system is YAFFS. We compare both read and write times in each file system that one is reference YAFFS and the other is modified YAFFS. Only ECC has changed from Hamming code to RS code in the modified YAFFS.

There is different time result in two different file systems. The time overhead that about 9% of read operation and 34% of write operation is obtained. The results represent the process time to read or write per 1MB data which is same as 2048 pages.

V. Conclusion

The MLC flash memories have been developed and are widely used for a storage device. However, error correction codes which are in flash file systems are not enough to overcome the disadvantages of MLC. To make error correction ability high, we proposed RS code for ECC to YAFFS instead of Hamming code. We can recognize that Hamming code is faster than RS code, however Hamming code can correct only one bit error in 256 bytes of data in contrast to proposed RS code can correct up to 12 bits error.

References

[1] R. Micheloni, R. Ravasio, A. Marelli, et. Al, "A 4Gb 2b/cell NAND Flash Memory with Embedded 5b BCH ECC for 36B/s System Read Throughput," *IEEE Inter. Solid-state Circuits Conf.*, Feb. 2006.
[2] ALEPH ONE LTD. "Yaffs: Yet another flash file system," http://www.yaffs.net.
[3] D. Woodhouse, "The jounalling flash file system," In *Ottawa Linux Symposium* (Ottawa, ON, Canada, July 2001).
[4] Y. Kang, E. L. Miller, "Adding Aggressive Error Correction to a High-Performance compressing Flash File System," *EMSOFT'09*, October 12-16, 2009, Grenoble, France.
[5] B. Chen, X. Zhang, and Z. Wang, "Error correction for multi-level NAND flash memory using Reed-Solomon codes," *Signal Processing Systems*, 2008. SiPS 2008. IEEE.
[6] A. Kawaguchi, S. Nishioka, and H. Motoda, "A flash-memory based file system," In *Proceedings of the Winter 1995 USENIX Technical Conference* (New Orleans, LA, Jan. 1995), USENIX, pp. 155-164.
[7] M. Rosenblum, J. K. Ousterhout, "The design and implementation of a log-structured file system," ACM *Transactions on Computer Systems 10*, 1 (Feb. 1992), 26-52.
[8] R. W. Hamming, *Coding and Information Theory*, second ed. Prentice-Hall, Englewood Cliffs, New Jersey, 1986.