International Journal of Multimedia and Ubiquitous Engineering
Vol.10, No.9 (2015), pp.309-318
http://dx.doi.org/10.14257/ijmue.2015.10.9.32

In-Block Level Redundancy Management for Flash Storage System

Seung-Ho Lim

Division of Computer and Electronic Systems Engineering
Hankuk University of Foreign Studies
slim@hufs.ac.kr

Abstract

Recently, the density of Flash memory has dramatically increased by moving to smaller geometries and storing more bits per cell with enhanced memory technologies. However, with the density increasing rapidly, the error rate of Flash memory is also increasing rapidly. To improve the reliability issue of upcoming flash memory, usually, redundancy techniques were adopted in both of architecture and operations for flash memory based systems. For outside the flash memory page, several reliability schemes were proposed which employ Redundant Array of Inexpensive Disks (RAID). In this paper, we propose in-block level redundancy scheme for providing reliability of flash memory while minimizing maintaining overhead of the redundancy. In the proposed scheme, the redundancy is generated in a flash memory block level. During the programming stage of a flash block, the redundancy is kept in Dram memory. If a block is exhausted with last-1 page to record incoming data, the in-keeping redundancy is flushed to last page of the block. By doing this, block level redundancy is maintained with minimal overhead.

Keywords: *Flash Memory, Block, Page, Parity, Bit Error Rate*

1. Introduction

Recently, with the help of low power consumption, low density, high capacity, and high IO bandwidth, NAND flash memory-based nonvolatile memory has become a main storage media for mobile embedded systems. Nowadays, NAND Flash memory-based storage devices such as Solid State Drive (SSD) [2], embedded Multimedia Cards (eMMC) [3], and USB memory sticks are widely used in variety of computing systems.

In flash memory, there is a floating gate (FG) insulated all around by an oxide layer. The FG is interposed between the CG and the MOSFET channel. Because the FG is electrically isolated by its insulating layer, electrons placed on it are trapped until they are removed by another application of electric field [1]. The density of Flash memory has increased by moving to smaller geometries and storing more bits per cell. In each cell in flash memory, the FG represents stored values, *i.e*., one or zero, according to the trapped electrons level. Therefore, if one bit is represented by one cell, there are two levels in the one cell, and if four bits are represented by one cell, there are four levels in the one cell, and so on. Currently, three-level cell (TLC) is main stream for NAND flash storage.

However, with the density increasing rapidly, the error rate of Flash memory is also increasing rapidly. The growth of number of states per cell raises interference between states since the quantized decision levels of the cell are getting close between adjacent states, which makes errors of detection. Moreover, the Program/Erase cycles (P/E cycles) have come down rapidly with the density increasing. The bit error rate is abruptly increased from some levels of P/E cycles, which in turn reveals the bit error rate is strongly correlated to the P/E cycles levels. In summary, the error rates occurred in the flash memory is getting worse as the density of flash memory is getting better.

ISSN: 1975-0080 IJMUE

To mitigate and improve the reliability issue of upcoming flash memory, usually, redundancy techniques were adopted in both of architecture and operations for flash memory based systems. Inside the flash memory page, Error Correction Code (ECC) is stored in the out-of-band region within a flash memory page, which we call page-level redundancy. Several ECC schemes were applied to flash memory chip as hardware components. Even ECC exists and it can cover one bit correction, or even several bit corrections according to its complexity, outside-page level Errors cannot be covered. For outside the flash memory page, several reliability schemes were proposed which employs Redundant Array of Inexpensive Disks (RAID) [7-10]. Outside the page level, the redundancies are generated with cluster of several pages. The pages can be grouped across the blocks or chips to spread out the redundancies. However this spread out of redundancy generation and distributed places of these cause additional complex and operational overhead in both memory and IO operations.

In this paper, we propose in-block level redundancy scheme for providing reliability of flash memory based storage system. In the proposed scheme, the redundancy is generated in flash memory block level and the generated redundancy is stored at the end page of a flash block. Usually, the programing operation of flash memory has a logging manner, which means that the write of pages in a flash block is from first page to last page. During the programming stage of a flash block, the redundancy is kept in Dram memory. The in-keeping redundancy is used to re-calculated up-to-date redundancy with incoming data. If a block is exhausted with last-1 page to record incoming data, the in-keeping redundancy of main memory is flushed to last page of the block.

The remainder of this paper is organized as follows. In Section 2, background and related work is described. The proposed in-block level redundancy scheme is explained in Section 3, and its performance evaluation is described in Section 4. Section 5 concludes this paper.

2. Background and Related Work

In this Section, the backgrounds and related work are illustrated. Most of the background is referred by our previous work [11].

2.1. NAND Flash Memory Basics

NAND flash memory is array of memory cells that consist of floating-gate transistors. There are three commands used in NAND flash memory; read, program (write), and erase. The read and program command are related with data transfer between host and Flash devices, whose data unit is Page. The erase command has no data transfer between host and Flash, and the erase is operated at the Block-based. In NAND flash memory, the size of one Page is 4KB and doubles as manufacturing process advances and a Block is composed of 64 or 128 Pages, typically. Due to the size mismatch between write and erase operation, write operation should consider efficient erase operation. Typically, read for one Page consumes about 125us, including Page read to internal chip register and bus transfer from chip register to host side. As a same manner, write for one Page consumes about 200us~400us.

The physical limitation of NAND flash memory is covered by special firmware called Flash Translation Layer (FTL) [4-6]. The NAND flash-based devices such as Solid State Disk (SSD) [2] and Multimedia Card (MMC) [3] embed FTL inside their systems as a form of firmware, so FTL runs on controller within the devices. FTL manages address mapping between file system and NAND flash memory, and does GC (Garbage Collection) [5]. File systems can treat Flash device just like usual storage media with the logical block address supported by FTL, and FTL hides internal mapping information and management schemes from host file systems.

FTL manages address mapping table between logical address of host part and physical address of Flash memory. Indeed, except the mapping management, FTL does many other roles, including wear leveling, garbage collection, bad block management, and request queuing and caching, and so on. However, the mapping management is most important job among many FTL's roles, and others are mostly dependent on the mapping management scheme. The FTL keeps track of the address mapping information between the logical address and the physical address. In this manner, the FTL prevents in-place updates of data and hides the latency of the erase operation. When the number of free pages is insufficient for write operations, free pages should be made by garbage collection (GC), where GC is the process that makes available free region by selecting one Block, moving data of valid Pages to other region, and erasing the Block. Thus, the selected victim Block should have minimum valid Pages for more efficient garbage collection.

2.2. Related Work

Typical approach against the short endurance and increasing errors is the use of well-known Error Correction Codes (ECC), or parity schemes like RAID technique used by legacy storage systems. Inside the flash memory page, Flash memory employs Error-Correcting Codes (ECC) to provide error correction for one bit or several bits per page during page read operation. The ECC data is stored in out-of-band region of flash memory page. For the ECC coding scheme, low-density parity check (LDPC) codes are considered as the choice for current flash memory controller due to their superior error correction capabilities.

While ECC can prevent data error inside a page, there is another approach for preventing data error outside a page with redundancy. RAID [7] enhances the reliability by using redundant data. In the RAID technique, the parity generation is relatively simple than ECC code algorithm, it generates more additional read and write requests, as well as space for redundancy data. In addition, since the parity data are generated across the flash memory chips, the coordination of each data write processing channel is required for efficient parity management, which degrades read and write latency. Among many RAID levels, RAID-4 and RAID-5 use an extra redundant parity block to hold redundant information of clustered blocks. The parity is generated by bitwise exclusive-or operations between blocks that belong to same cluster, thus, parity update usually requires two block reads and two block writes in traditional storage system. However, in flash-based storage devices, due to the flash memory characteristics, *i.e.*, out-of-place update, the approach of RAID techniques is different, which is mainly focus on reducing write requests for designing flash-based RAID scheme.

There are many previous works for RAID schemes for flash memory based embedded devices, which are mainly for SSD systems. Im [8] presents partial parity caching technique to reduce the number of read operations required to calculate parity, while Kim *et al* addresses the dynamic forms of a variable size stripe based on the arrival order of write requests without any parity cache [9]. In [10], they utilize built-in NVRAM to cache the parity data update for minimal write to flash memory.

3. In-Block Level Redundancy Scheme

In case of errors, another redundancy that exist out-of-the page are required to recover proper data. The out-of-the page redundancies would be generated like RAID technique, where parity page with regard to a group of pages clustered exists. The parity is calculated as a bitwise exclusive-or operation among pages of cluster, so it is also a complex computation job. The issue is that how to make up cluster of

pages. In our design, the parity group is cluster of the pages within same flash memory block, as a result, parity page is the last page of the flash memory block, which we call it in-block redundancy. The in-block redundancy makes it possible to make parity be generated and recovered independently for each block, each chip, and each core as well.

The In-block level redundancy scheme is described in Figure 1. In our scheme, a Flash memory block consists of one parity page and remaining pages are used for data. The last page is preserved for parity page. With this composition, the redundancy generation is as follows. When data is arrived from host system, its corresponding address is logical address, so, at first in Storage system, FTL translate logical address to physical address. Since FTL management is out of scope of this paper, we omit the specific description of the translation mechanism. Please refer to mapping management and recovery process for page-mapping FTL in [12]. After the FTL translation, the physical address of data is set. Then, in the in-block level redundancy scheme, the parity data is calculated for the incoming data. As shown in the Figure 1, the parity data is calculated and maintained in main memory, where parity data is calculated by bitwise exclusive operation of in-memory parity and incoming data. The calculated parity data is back to buffer in main memory, not flushed to NAND flash storage media. The incoming data is then written to NAND flash memory. If the physical address of the incoming data is above the last-1 page of the block, the parity data is not flushed to NAND flash memory, but just maintained in the memory.

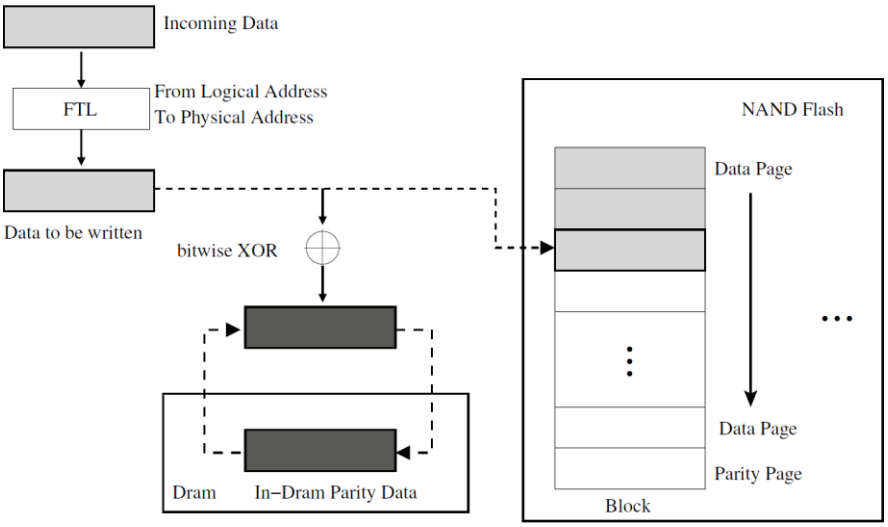

Figure 1. Parity Update During Incoming Data are Written in the Middle of Flash Memory Block

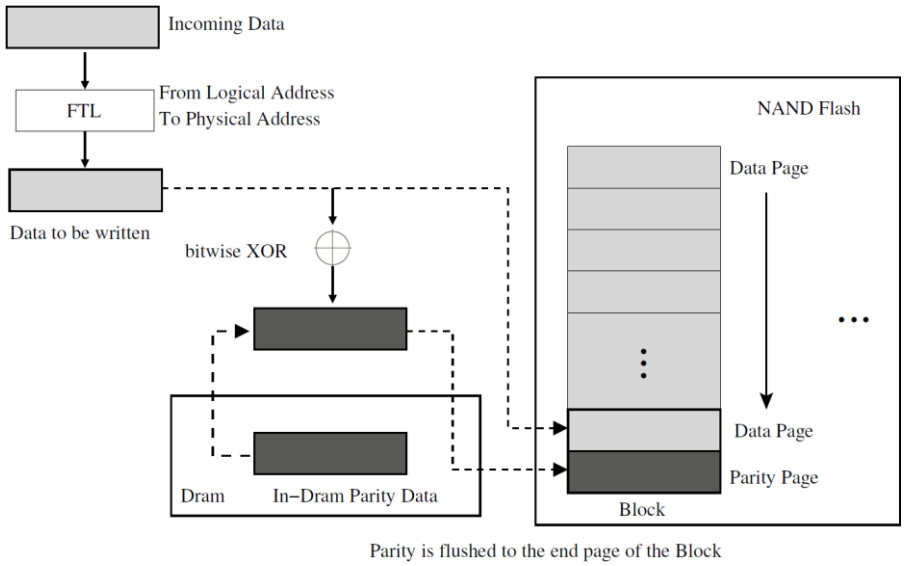

Figure 2. Parity Flush after Last-a Page is Written to the Flash Memory Block

Almost all FTL writes data to flash memory block as a log manner, which means data are written to flash memory block in a sequential order from first page to last, and there is no update for written area in a block until the block is recycled by GC. For this reason, it is possible to update parity with just incoming data without retrieval of previously written data, which is different from parity update of legacy HDD-based RAID system. Thus, the parity can be only updated by recalculating of incoming data and managed in Dram memory until all the pages in a block are consumed for incoming data.

When the physical address of incoming data is decided to last-1 page of the block, the calculated parity data is flushed on last page of the block after updating parity data with incoming data and writing incoming data, as shown in the Figure 2. After flushing the parity data, the parity buffer in main memory is cleaned and it is ready for another parity buffer for another block. The physical address of current incoming data is last-1 page means all the data pages in that block are consumed for data. In in-block level redundancy scheme, the last page of the block is used to store parity data in the block. In the parity page, each bit represents bitwise exclusive-or data of pages above all the data pages in the block. Even if the page has invalid data, the parity calculation is possible since data is not deleted physically.

Whether power off is sudden power off or not, if power off is occurred during the parity page maintained only in the main memory, not storing to NAND flash memory, the parity data should be recovered. The parity recovery process is held at the end of the recovery of FTL map. Although recovery of mapping table of FTL is also omitted in this paper, please refer to the previous work that presents recovery process of page-mapping FTL [12]. At the end of recovery for mapping table, FTL should find out the block at the moment of sudden power off by checking FTL's own mapping management to recover the mapping management of last used block of previous running time. Likewise, in our parity scheme, we only need to recover the parity data of the last block, since only one parity data is maintained in the main memory for the last used block, while others were stored in the last page in their blocks. The recover process of the last used block at the moment of power off is described in Figure 3. For the block that was used right before power-off, each data page is retrieved and parity data is calculated by bitwise exclusive-or operations. This process is circulated until valid data page is retrieved from the block.

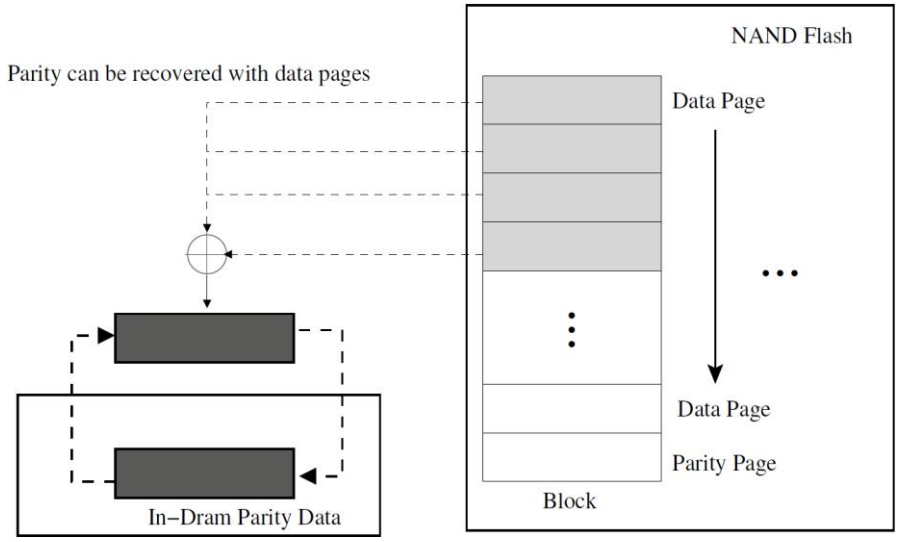

Figure 3. Recovery Process for Data that was not Flush to NAND Flash Memory Due to Power Loss

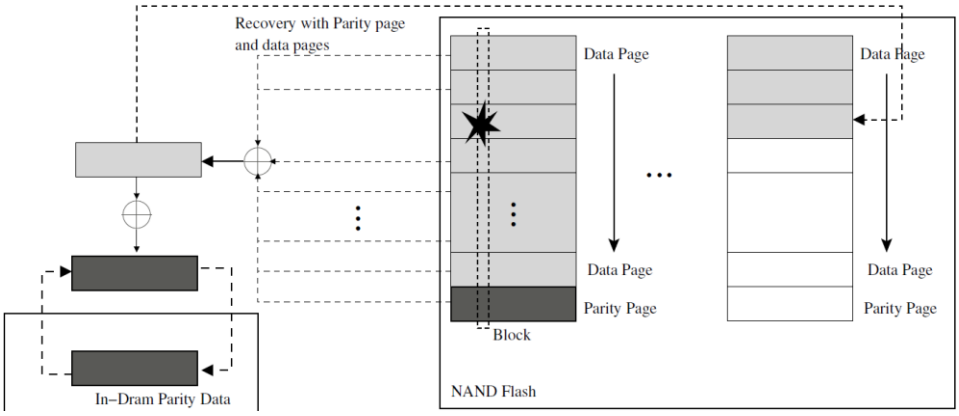

Figure 4 Data Recovery Process for Bit Errors of Data Page

Last, data recovery for broken data is described, which data recovery process is shown in the Figure 4. When host read some data, the read command is transferred from host to flash memory, which contains logical address. The logical address is translated to physical address by FTL, and FTL retrieves data from Flash memory with specific physical pages address. If there occur errors while reading data page of the physical address, which means some part of data in the data page were broken. However, we don't know which part is broken, and how much broken.

In in-block redundancy scheme, when the error is reported, data recovery process is done which is similar to the recovery process of parity data. For the block where data error occurs, each data page as well as parity page, except the error data page, is read and bitwise exclusive-or calculation is done. By doing this, broken page is recovered. After the recovery is done, this recovered data should be written to somewhere of Flash memory block. This is done as like usual incoming data write, and after the write of recovered data, the mapping table of FTL is also updated.

4. Implementation and Analysis

The proposed in-block redundancy scheme was developed with Linux-based Flash simulator, with Linux Memory Technology Device (MTD) device driver layer [14]. In the MTD device driver, there exist FTL layer as block device driver layer, NAND device

driver for command interfaces of FTL block layer and read chip device, and NANDsim simulator for NAND flash simulation. The developed in-block redundancy layer is implemented in FTL layer. Among several FTL management algorithm published, page-mapping FTL was adapted in our implementation, although others are applied similarly. The log-based page mapping and mapping table management of implemented FTL is described in detail in [12]. In the page-mapping FTL, a block is selected for incoming write operations. During pages are consumed for incoming writes within the block, in-block parity buffer is maintained in the memory within the FTL, while, the parity is calculated and updated with bitwise exclusive-or operations between incoming write data and parity data in the buffer. When all the data pages of the block is exhausted for data writes, the parity buffer is flushed to last page of the block via NAND device driver.

The strength of in-block redundancy is that it downs bit error rates with the redundancy data while having relatively less complexity for calculating and maintaining redundancy data. Since the reduced error rates for in-block redundancy is similar to those of others that have RAID redundancy scheme, the efficiency of bit error rate is estimated similar to other RAID technique. Every single uncorrected bit errors can be recovered by parity data if other bits with same column are read properly.

We have estimated the redundancy overhead by comparing with legacy system, which means that there is no redundancy scheme. There are three issues that can be discussed in in-block redundancy scheme; storage overhead, parity update overhead, and parity recovery overhead. The storage overhead for redundancy is essential part for all the redundancy schemes, not limited to in-block redundancy. In the in-block redundancy scheme, one page per block is used for parity data. The parity update overhead of in-block redundancy scheme is negligible since almost update of parity is done within main memory, not companied by flash IO operation until last data page, *i.e*., (last-1) page of the block is consumed. The only overhead for parity update is in-memory bitwise exclusive-or overhead. To minimize the exclusive-or overhead, we can consider memory-offloading for exclusive-or, or parallel processing between parity calculation and data page writes to Flash memory. The parallel processing is possible in in-block redundancy scheme with the help of parity recovery process. The parity recovery should be done by every power-on step in our scheme, whether other power los recovery is done or not, since the parity data for a block that was flush to parity page in the block before power loss is not preserved by any location. Thus, for every power-on step, whether it is proper power-on step or not, *i.e*., it is power-loss-recovery step, parity recovery process always takes up time. We have simulated average recovery time with the MTD and NANDSim simulation environment, and the estimated recovery coverage is limited by less than half of the block size which is limited by small scanning area in comparison with capacity of NAND flash memory. Although the recovery is accompanied by reading and bitwise exclusive-or operations, the time is extremely limited by the small recovery area.

5. Conclusion

Recently, the density of Flash memory has dramatically increased by moving to smaller geometries and storing more bits per cell with enhanced memory technologies. If one bit is represented by one cell, there are two levels in the one cell, and if four bits are represented by one cell, there are four levels in the one cell, and so on. Currently, three-level cell (TLC) is main stream for NAND flash storage. However, despite the evolution of capacity of NAND flash memory, there is also increasing crucial problem for it to be used as storage device. With the density increasing rapidly, the error rate of Flash memory is also increasing rapidly. The growth of number of states per cell raises interference between states since the quantized decision levels of the cell are getting close between adjacent states, which makes errors of detection.

To improve the reliability issue of upcoming flash memory, usually, redundancy techniques were adopted in both of architecture and operations for flash memory based systems. For outside the flash memory page, several reliability schemes were proposed which employs Redundant Array of Inexpensive Disks (RAID). Outside the page level, the redundancies are generated with cluster of several pages. The pages can be grouped across the blocks or chips to spread out the redundancies. However this spread out of redundancy generation and distributed places of these cause additional complex and operational overhead in both memory and IO operations.

In this paper, we propose in-block level redundancy scheme for providing reliability of flash memory while minimizing maintaining overhead of the redundancy. In the proposed scheme, the redundancy is generated in flash memory block level. During the programming stage of a flash block, the redundancy is kept in Dram memory. If a block is exhausted with last-1 page to record incoming data, the in-keeping redundancy is flushed to last page of the block. By doing this, block level redundancy is maintained with minimal overhead.

Acknowledgements

This work was supported by Hankuk University of Foreign Studies Research Fund. The preliminary version of this work was appeared in CES-CUBE 2015.

Reference

[1] "Flash Memory", http://en.wikipedia.org/wiki/Flash_memory.
[2] Webopedia, "What is solid state disk? - A Word Definition from the Webopedia Computer Dictionary", ITBusinessEdge, **(2012)**.
[3] EDEC, "Embedded Multimedia Card Electrical Standard", September **(2013)**.
[4] A. Ban, "Flash file system optimized for page-mode flash technologies. U.S. Patent 5,937,425", Filed, October 16, **(1997)**.
[5] Intel Corporation, "Understanding the flash translation layer (FTL) specification", http://developer.intel.com/
[6] J. Kim, J. M. Kim, S. H. Noh, S. L. Min and Y. Cho, "A space efficient flash translation layer for CompactFlash systems", IEEE Transactions on Consumer Electronics, vol. 48, no. 2, May **(2002)**, pp. 366-375.
[7] P. M. Chen, E. K. Lee, G. A. Gibson, R. H. Katz and D. A. Patterson, "RAID: High-Performance, Reliable Secondary Storage", ACM Computing Surveys, vol. 26, no. 2, **(1994)**, pp. 145-185.
[8] S. Im and D. Shin. "Flash-Aware RAID Techniques for Dependable and High-Performance Flash Memory SSD", IEEE Transactions on Computers, vol. 60, no. 1, January **(2011)**, pp. 80 92.
[9] J. Kim, J. Lee, J. Choi, D. Lee and S. Noh, "Enhancing SSD Reliability through Efficient RAID Support", in Proceedings of 3th Asia-Pacific Workshop on Systems, July, **(2012)**.
[10] Y. Qin, D, Feng, J. Liu, W. Tong, Y. Hu and Z. Zhu, "A Parity Scheme to Enhance Reliability for SSDs", in Proceedings of 7th International Conference on Networking, Architecture, and Storage, **(2012)**.
[11] Y. S. Lee, L. Barolli and S. H. Lim, "Mapping granularity and performance tradeoffs for solid state drive", The Journal of Supercomputing, vol. 65, no. 2, **(2013)**, pp. 507-523.
[12] S. H. Lim, "Implementation of metadata logging and power loss recovery for page-mapping FTL", IEICE Electronics Express, vol. 10, no. 11, May **(2013)**, pp.1-6.
[13] S. H. Lim, "Virtually Separable Block Management in Flash Storage System", International Journal of Multimedia and Ubiquitous Engineering", vol.9, no.9, September **(2014)**, pp. 299-310.
[14] Memory Technology Devices, "Memory Technology Device Overview", Retrieved 1, http://www.linux-mtd.infradead.org/, September **(2012)**.
[15] S. H. Lim, "A Light-Weight Redundancy Technique for Limited Embedded Flash Storage Device", 5th International Conference on Circuits, Control, Communication, Electricity, Electronics, Energy, System, Signal and Simulation, June **(2015)**.

Authors

Seung-Ho Lim, received BS, MS, and PhD degrees in the Division of Electrical Engineering from the Korea Advanced Institute of Science and Technology (KAIST) in 2001, 2003, and 2008, respectively. He worked in the memory division of Samsung Electronics Co. Ltd from 2008 to 2010, where he was involved in developing a high performance SSD (Solid State Disk) for server storage systems. He is currently a professor in the Division of Computer and Electronic Systems Engineering at Hankuk University of Foreign Studies. His research interests include operating systems, embedded systems, and flash storage systems.