System Software for Flash Memory: A Survey

Tae-Sun Chung[1], Dong-Joo Park[2], Sangwon Park[3], Dong-Ho Lee[4], Sang-Won Lee[5], and Ha-Joo Song[6]

[1] College of Information Technoloty, Ajou University, Korea
tschung@ajou.ac.kr
[2] School of Computing, College of Information Science, Soongsil University, Korea
[3] Computer Science & Information Communication Engineering Division, Hankuk University of Foreign Studies, Korea
[4] Department of Computer Science and Engineering, Hanyang University, Korea
[5] School of Information and Communications Engineering, Sungkyunkwan University, Korea
[6] Division of Electronic, Computer and Telecommunication, Pukyong National University, Korea

Abstract. Recently, flash memory is widely adopted in embedded applications since it has several strong points: non-volatility, fast access speed, shock resistance, and low power consumption. However, due to its hardware characteristic, namely "erase before write", it requires a software layer called FTL (Flash Translation Layer). This paper surveys the state-of-the-art FTL software for flash memory. This paper also describes problem definitions, several algorithms proposed to solve them, and related research issues. In addition, this paper provides performance results based on our implementation of each of FTL algorithms.
Keywords: Flash memory, Embedded System, File System

1 Introduction

Flash memory has inherently strong points compared to traditional hard disk: non-volatility, fast access speed, shock resistance, and low power consumption. Therefore, it has been widely adopted in embedded applications such as USB flash memory, CF card memory, mobile devices, and so on. However, due to its hardware characteristics, flash memory-based applications require special software operations while reading (writing) data from (to) flash memory.

One of basic hardware characteristics of flash memory is that it has an erase-before-write architecture [4]. That is, in order to update a location on a flash memory, it has to be first erased before the new data can be written to it.

Moreover, the erase unit (block) is larger than the read or write unit(sector) [4] resulting in the major performance degradation of the overall flash memory system.

Therefore, the system software called FTL (Flash Translation Layer) should be introduced [1, 2, 5, 8, 10, 11]. The basic scheme for FTL is as follows. By using the logical to physical address mapping table, if a physical address location mapped to a logical address is previously written, the input data is written to

an empty physical location to which no data have ever been previously written and then the mapping table is updated due to newly changed logical/physical address mapping. This protects one block from being erased per overwrite.

In applying the FTL algorithm to real embedded applications, there are two major considerations: the storage performance and the SRAM memory requirement. With respect to the storage performance, as flash memory has special hardware characteristics as mentioned above, the overall system performance is mainly affected by the write performance. In particular, since the erase cost is much more expensive than the write or read cost, it is very important to minimize the erase operations. Additionally, the SRAM memory required to keep the mapping information is so valuable resource in real embedded applications that if an FTL algorithm requires large SRAM memory, it will increase the product cost of embedded applications, leading to losing the price competitiveness.

In this paper, we survey the-state-of-the-art FTL algorithms. Gal et al. [6] have also provided algorithms and data structures for flash memories. Compared to the work, our work focuses on FTL algorithms and does not discuss file system issues [9, 13, 7]. we describe the problem definition, the FTL algorithms proposed to solve it, and the related research issues. In addition, we provide performance results based on our implementation of each of FTL algorithms.

This paper is organized as follows. Section 2 describes problem definition. Section 3 shows how the previous FTL algorithms can be classified, and Section 4 presents performance results. Finally, Section 5 concludes the paper.

2 Problem Definition & FTL Functionalities

2.1 Problem Definition

First, we define operation units in the flash memory system as follows.

Definition 1. *A sector is the smallest amount of data which is read or written at a time. That is, a sector is the unit of read or write operations.*

Definition 2. *A block is the unit of the erase operation on flash memory. The size of a block is multiples of the size of a sector.*

Figure 1 shows the software architecture of the flash file system. We will focus on the FTL layer in Figure 1. The file system layer issues a series of read or write commands with a logical sector number each, to read data from, or write data to, specific addresses of flash memory . The logical sector number is converted to a real physical sector number of flash memory by some mapping algorithm in the FTL layer.

Thus, the problem definition of FTL is as follows. We assume that flash memory is composed of n physical sectors and the file system - the upper layer - considers a flash memory as a block-i/o device that is consisted of m logical sectors. Since a logical sector has to be mapped at least one physical sector, the number m is less than or equal to n.

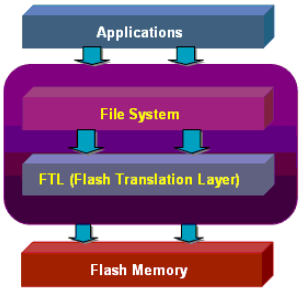


Fig. 1. Software architecture of the flash memory system

Definition 3. *Flash memory is composed of a number of blocks and each block is composed of multiple sectors. Flash memory has the following characteristics: If the physical sector location on flash memory was previously written, it has to be erased in the unit of block before the new data can be overwritten. The FTL algorithm is to produce the physical sector number in flash memory from the logical sector number given by the file system.*

2.2 FTL Functionalities

An FTL algorithm is supposed to provide the following functionalities.

- Logical to physical address mapping: The main functionality of an FTL algorithm is to convert logical addresses from the file system to physical addresses of flash memory.
- Power-off recovery: Even though a sudden power-off occurs during FTL operations, the data structures of the FTL system should be recovered and its consistency has to be maintained.
- Wear-leveling: FTL should include the wear-leveling function to wear down memory blocks as evenly as possible.

3 A Taxonomy for FTL Algorithms

In this section, we suggest a taxonomy of FTL algorithms according to features such as addressing mapping, mapping information management, and the size of the SRAM table.

3.1 Addressing Mapping

Sector Mapping A naive and intuitive FTL algorithm is the sector mapping [1]. In sector mapping, every logical sector is mapped to a corresponding physical sector. Therefore, if there are m logical sectors seen by the file system, the row size of logical to physical mapping table is m.

For example, Figure 2 shows an example of sector mapping. In the example, we assume that a block is composed of four pages and so there are totally 16 physical pages, where each page is organized into the sector and spare areas. If we also assume that there are 16 logical sectors, the row size of the mapping table is 16. When the file system issues a command - "write some data to LSN (Logical Sector Number) 9", the FTL algorithm writes the data to PSN (Physical Sector Number) 3 according to the mapping table in case the PSN 3 has not been written before.

But, in other case, the FTL algorithm looks for an empty physical sector, writes data to it, and adjusts the mapping table. If there is no empty sector, the FTL algorithm will select a victim block from flash memory, copy back the valid data to the spare free block, and update the mapping table. Finally, it will erase the victim block, which will become the spare block.

In order to rebuild the mapping table after power outage, the FTL algorithm either stores the mapping table to flash memory or records the logical sector number in the spare area on each writes to the sector area.

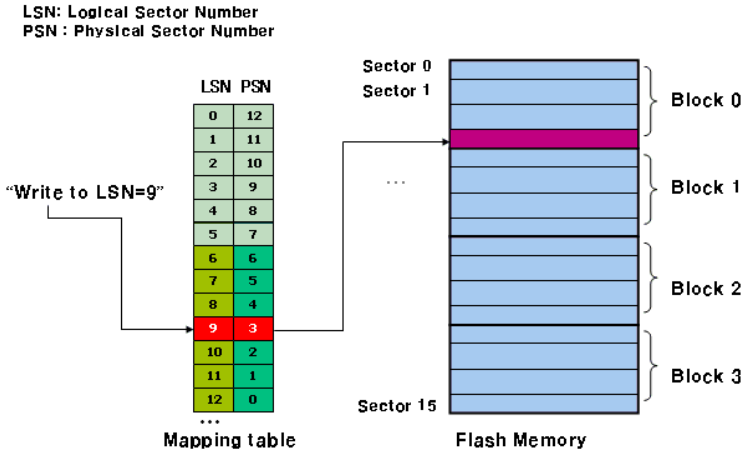


Fig. 2. Sector mapping

Block Mapping The sector mapping algorithm requires such a large memory space (SRAM) that it is hardly feasible for small embedded systems. For this reason, the block mapping-based FTL algorithms [2, 5, 10] are proposed. The basic idea of the block mapping algorithms is that the logical sector offset within a logical block is identical to the physical sector offset within the physical block.

In the block mapping scheme, if there are m logical blocks seen by the file system, the row size of logical to physical mapping table is m. Figure 3 shows an example of the block mapping algorithm. Assuming that there are four logical blocks, the row size of the mapping table is four. If the file system issues the command "write some data to LSN 9", the FTL algorithm calculates the logical

block number 2(=9/4) and the sector offset 1(=9%4), and then finds physical block number 1 using the mapping table. Since in the block mapping algorithm, the physical sector offset equals the logical sector offset, the physical sector location can be easily determined.

Hence the block mapping algorithm requires a small size of mapping information. However, if the file system issues writes with the same LSNs many times, the FTL algorithm has to perform the copy and erase operations frequently, leading to severe performance degradation.

For rebuilding the mapping table, the FTL algorithm records the logical block number in the spare area of the first page of the physical block.

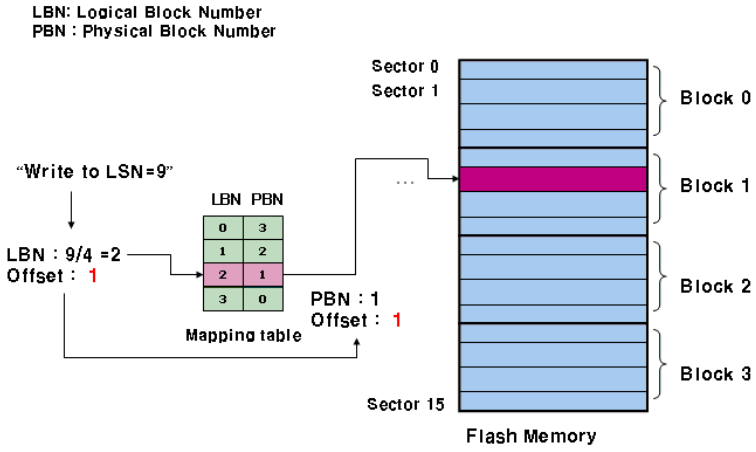


Fig. 3. Block mapping

Hybrid Mapping Since both sector and block mapping have some disadvantages as mentioned in the previous two subsections, hybrid mapping approaches are introduced [8, 11]. A hybrid technique, as its name suggests, first uses a block mapping technique to get the corresponding physical block , and then, uses a sector mapping technique to find an available empty sector within the physical block.

Figure 4 shows an example of the hybrid technique. When the file system issues the command "write some data to LSN 9", the FTL algorithm calculates the logical block number 2(=9/4) for the LSN, and then, finds the physical block number 1 from the mapping table. After getting the physical block number, the FTL algorithm allocates an empty sector for the update. In the example, since the first sector of the pysical block 1 is empty, the data is written to the first sector location. In this case, since the two logical and physical sector offsets(i.e., 2 and 1, respectively) differ from each other, the logical sector number 9 should be written to the spare area in page 1 of the physical block 1. For rebuilding

the mapping table, not only this information but also the logical block numbers have to be recorded in the spare areas of the physical blocks.

When reading data from flash memory, the FTL algorithm first finds the physical block number from the mapping table using the given LSN, and then, by reading the logical sector numbers from the spare areas of the physical block, it can get the most recent value for requested data.

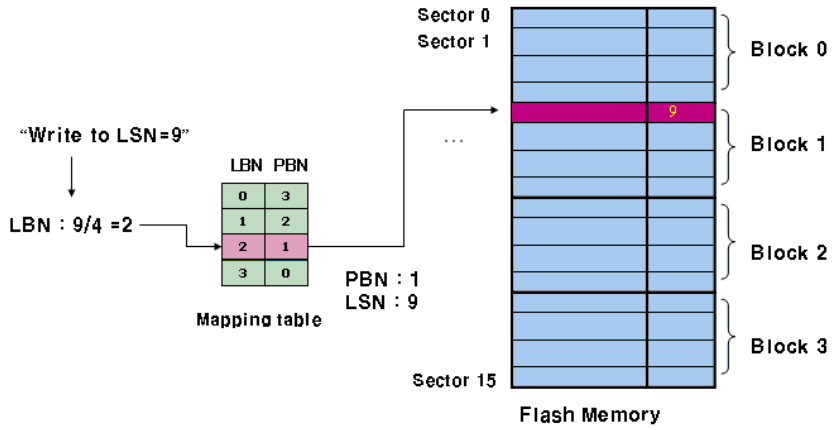


Fig. 4. Hybrid mapping

Comparison We compare the performance of FTL algorithms in two perspectives: file system- issued read/write performance and memory requirement for storing mapping information.

The read/write performance of an FTL algorithm can be measured by the number of flash I/O operations (read, write, and erase), since the read/write performance is I/O-bounded. We assume that the mapping table of an FTL algorithm is maintained in SRAM, and the access cost of the mapping table is zero, then the read and write costs can be computed by the following two equations, respectively.

$$C_{read} = xT_r \tag{1}$$

$$C_{write} = p(T_r + T_w) + (1-p)(T_r + (T_e + T_w) + T_c) \tag{2}$$

C_{read} and C_{write} above are the costs of read and write issued from the file system layer, respectively, and T_r, T_w, and T_e are the costs of read, write, and erase processed in the flash memory layer. T_c is the cost of copies needed to move sectors within a block to other free block before erasing and to copy back after erasing. p is the probability that a write command does not incur any erase

operation. We assume that the input logical sector within the logical block is mapped to one physical sector within a physical block.

In the sector and block mapping techniques, the variable x in the equation 1 has 1, since the sector to be read can be found directly from the mapping table. However, in the hybrid technique, the value of the variable x is in the range of $1 \leq x \leq n$, where n is the number of sectors within a block. This is because the requested data can be read only after scanning the logical sector numbers stored in the spare areas of a physical block. Thus, the hybrid mapping scheme has higher read cost compared to the sector and block mapping techniques.

In case of the write cost, we assume that a read operation is required before actual a write to see if the input data can be written to in-place sector location. Thus, T_r has to be added in the equation 2. Since T_e and T_c are higher cost operations compared to T_r and T_w, the variable p is a key point in computing the write cost. In the sector mapping, the probability of requiring an erase operation per write is relatively small, while in block mapping, it is relatively high.

Another comparison criteria is the memory requirement for storing mapping information. Table 1 shows such memory requirements for the three address mapping techniques. Here, we assume a flash memory device of size 128MB, where flash memory is composed of 8192 blocks and each block is composed of 32 sectors [4]. In sector mapping, three bytes are needed to represent the whole sector numbers, while in block mapping, only two bytes are needed to represent the whole block numbers. An hybrid mapping requires three bytes for block mapping and for sector mapping within a block, respectively. From the Table 1 block mapping requires the smallest SRAM memory as expected.

	Bytes for addressing	Total
Sector mapping	3 Bytes	3B * 8192* 32 = 768KB
Block mapping	2 Bytes	2B *8192= 16KB
Hybrid mapping	(2+1) Bytes	2B*8192+1B*32*8192 = 272KB

Table 1. Memory requirement for mapping information

3.2 Managing Address Mapping Information

The most important meta information in the FTL algorithms is the address mapping information. In order to be able to rebuild the address mapping table during a power-on process, the address mapping information should not be lost in the sudden power-offs, and therefore it has to be persistently kept somewhere in flash memory. We can classify the techniques for storing the mapping information on flash memory into two categories: map block method and per block method.

Map Block Method A map block method stores the address mapping information into some dedicated blocks of flash memory called map blocks. If we use one map block, erase operations on the map block happens so frequently. Hence we use several map blocks in order to lessen such frequent erases.

In the case that the mapping information may be changed due to writes issued by the file system, the above recording job will be done. When performing the recording job, if there is no unused sector in the map blocks pool, erase operations on all the map blocks have to be executed to make some free map blocks. The mapping table can be cached in SRAM for fast mapping lookups. In this case, the mapping table has to be rebuild in SRAM by reading the latest sector of the latest map block from flash memory during a power-on process.

Per Block Method The address mapping information can be stored to each physical block of flash memory. Differently from the map block method, logical block numbers are stored in the spare area of the first sector of each physical block. In addition, in order to to keep the mapping from a logical sector number to the sectors in a physical block, the logical sector numbers are recorded in each sector in the block. When rebuilding the mapping table due to power-off, both the logical block numbers and logical sector numbers in the spare areas of flash memory are used.

3.3 RAM Table

The size of RAM is very important in designing the FTL algorithm because it is a key factor in overall system cost. The smaller the RAM size, the lower the system cost. If a system has enough RAM, the performance can be improved. The FTL algorithms have their own RAM structures and we can classify the FTL algorithms according to their RAM structures. RAM is used to store following information of the FTL algorithms.

- Logical to physical mapping information: The major usage of RAM is to store the logical to physical mapping information. By accessing the RAM information, the physical flash memory location for reading or writing data can be found efficiently.
- Free memory space information: Once free memory space information in flash memory is stored in RAM, an FTL algorithm can manage the memory space without further flash memory accesses.
- Information for wear-leveling: The wear-leveling information can be stored in RAM. For example, the erase count of flash memory blocks may be stored in RAM.

4 Experimental Evaluation

For the simulation, we got various access patterns that the FAT file system [3] issues to the block device driver when it gets a file write request.

We will report performance results over two representatives of access patterns in embedded applications: Symbian [12] and Digicam. The first one is the workload of 1M byte file copy operation in Symbian operating system and the second one is the workload of digital cameras.

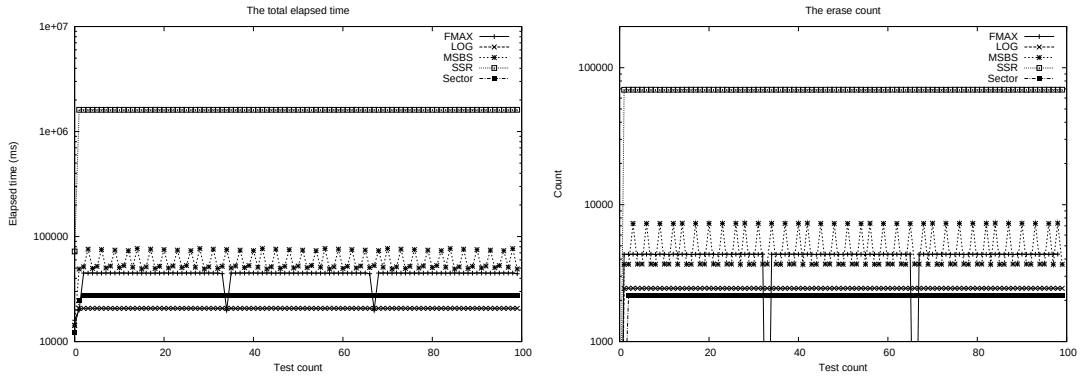
Figure 5-(a) shows the total elapsed time for the Digicam pattern. The x axis is the test count and the y axis is the total elapsed time in millisecond. At first, flash memory is empty, and flash memory is occupied as the iteration count increases. The result shows that the Log scheme shows the best performance. It is interesting that the Log scheme shows the better performance than the sector mapping which requires a lot of SRAM resources for mapping. We think the reason is that the workload of the Digicam is mostly composed of sequential write operations and the Log scheme operates almost ideally in the sequential write patterns.

Since the sector scheme uses a LSN-to-PSN mapping table, it has to update the mapping table each time overwrite operations are performed. This means it has to write the change in the mapping table to flash memory whenever a logical sector is overwritten, which is not happen so many times in Log scheme that uses a LBN-to-PBN mappings. In the experiment, we configured that there are 32 sectors in a block. Therefore, LSN-to-PSN mapping table have to be updated approximately 30 times often than the LBN-to-PBN mapping. Therefore, even though the sector mapping scheme incurs less erase operations (Figure 5-(b)), it has longer overall execution time than the Log scheme.

The SSR and Mitsubishi techniques show the poor performance compared to other techniques. We think the main reason is that the one logical block is mapped to only one physical block in the SSR and Mitsubishi techniques. In particular, when erasing a block in the SSR technique, since many valid sectors exist in the erased block, many copy operations are necessary and the probability that the erased block will be erased again in the future is very high. The Log scheme shows the better performance than FMAX. It is because, we think, that merging algorithm of FMAX is poor than that of the Log scheme in our workload.

Figure 5-(b) shows the erase count. The result is similar to the result of the total elapsed time. This is because the erase count is the most dominant factor in the overall system performance. [4] says that the running time ratio of read (1 page), write (1 page), and erase (1 block) is 1:7:63 approximately. We can see that the sector mapping requires the smallest erase counts. In sector mapping scheme, a logical sector can be mapped to any free physical sector. Most of the blocks are either full of invalid blocks or full of valid blocks. Therefore merge operations between blocks with valid sectors are uncommon, causing less erase operations.

Figure 6-(a) and Figure 6-(b) show the performance result in the Symbian workload. In the Symbian workload, the sector mapping shows the best performance. This result comes from the fact that the workload of Symbian, in contrast to Digicam, has many random write operations. Thus, in the Log scheme, the erase operation occurs frequently compared to the sector mapping.



(a) The total elapsed time (b) Erase counts

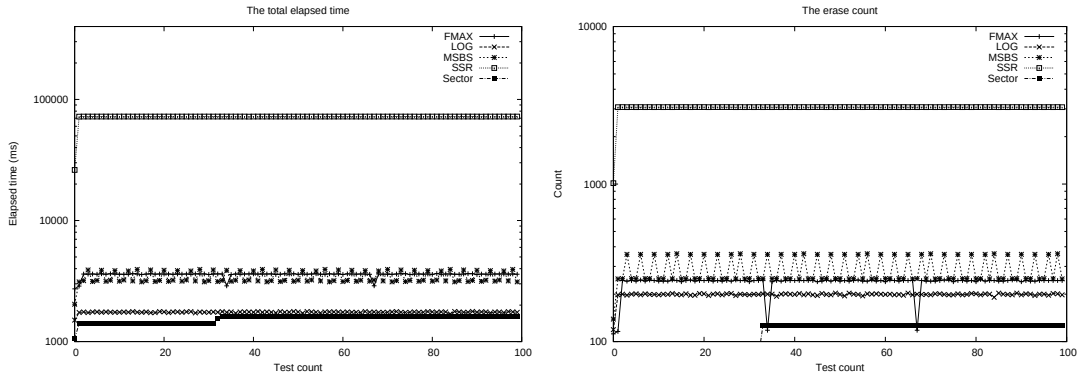
Fig. 5. Digicam: The total elapsed time and erase counts

5 Conclusion

In this paper, we have surveyed the state of the art FTL algorithms. First, we classified the FTL algorithms into three categories according to the address mapping method: sector, block, and hybid. In most cases, the sector mapping method shows the best performance but it requires much memory resources. The block mapping technique requires the smallest memory resources but its performance is the worst. Thus, the hybrid technique is proposed. In addition, the meta information storage techniques are classified into two categories: per block method and map block method. Finally, various FTL techniques differ in using RAM tables. We implemented various FTL algorithms such as FMAX, sector mapping, Log scheme, SSR, and Mitsubishi, and showed the performance results. The Log scheme shows good performance in the sequential access patterns. We can see that the workload of flash memory system has great impact on the overall performance of a flash memory system. For a further study, we want to design an FTL algorithm which has best performance while requiring smallest resources by exploiting the access patterns of file systems.

Acknowledgment

This work was supported in part by MIC & IITA through IT Leading R&D Support Project (2006-S-040-01) and was supported partly by the Ministry of Information and Communication, Korea under the ITRC support program supervised by the Institute of Information Technology Assessment, IITA-2005-(C1090-0501-0019), and also supported partly by Seoul R&BD Program(10660).



(a) The total elapsed time (b) Erase counts

Fig. 6. Symbian: The total elapsed time and erase counts

References

1. Amir Ban. Flash file system, 1995. United States Patent, no. 5,404,485.
2. Amir Ban. Flash file system optimized for page-mode flash technologies, 1999. United States Patent, no. 5,937,425.
3. Microsoft Corporation. Fat32 file system specification. Technical report, Microsoft Corporation, 2000.
4. Samsung Electronics. Nand flash memory & smartmedia data book, 2004.
5. Petro Estakhri and Berhanu Iman. Moving sequential sectors within a block of information in a flash memory mass storage architecture, 1999. United States Patent, no. 5,930,815.
6. Eran Gal and Sivan Toledo. Algorithms and data structures for flash memories. *ACM Computing Surveys*, 37(2), 2005.
7. A. Kawaguchi, S. Nishioka, and H. Motoda. Flash Memory based File System. In *USENIX 1995 Winter Technical Conference*, 1995.
8. Jesung Kim, Jong Min Kim, Sam H. Noh, Sang Lyul Min, and Yookun Cho. A space-efficient flash translation layer for compactflash systems. *IEEE Transactions on Consumer Electronics*, 48(2), 2002.
9. M. Resenblum and J. Ousterhout. The Design and Implementation of a Log-structured File System. *ACM Transactions on Computer Systems*, 10(1), 1992.
10. Takayuki Shinohara. Flash memory card with block memory address arrangement, 1999. United States Patent, no. 5,905,993.
11. Bum soo Kim and Gui young Lee. Method of driving remapping in flash memory and flash memory architecture suitable therefore, 2002. United States Patent, no. 6,381,176.
12. Symbian. http://www.symbian.com, 2003.
13. M. Wu and W. Zwaenepoel. eNVy: A Non-Volatile, Main Memory Storage System. In *International Conference on Architectural Support for Programming Languages and Operating Systems*, 1994.