Mobile Netw Appl (2008) 13:547–554
DOI 10.1007/s11036-008-0089-6

# The Real-Time Compression Layer for Flash Memory in Mobile Multimedia Devices

**Chin-Hsing Chen · Chun-Ta Chen · Wen-Tzeng Huang**

Published online: 20 July 2008


**Abstract** Flash memory is used for storage in mobile multimedia and embedded systems such as mobile phones, digital cameras, and MP3 players because of its small size, light weight, nonvolatile operation, vibration resistance, high capacity, and low power consumption. Data compression is one effective method for increasing capacity and reducing data transfer, however real-time performance is necessary for mobile multimedia device applications. We propose a mechanism that uses contiguous packing and a read/write ping-pong buffer along with the X-match and run-length compression/decompression algorithm to create a real-time compression layer. Compared to the internal packing scheme and best-fit method of Yim et al., our mechanism increases the compression ratio, improves the write response time by 60%, and stabilizes the read response time to make it suitable for real-time applications.

**Keywords** embedded systems · flash memory · real-time compression layer · mobile multimedia device

C.-H. Chen
Department of Management Information Systems,
Central Taiwan University of Science and Technology,
Taichung 40601 Taiwan, Republic of China

C.-T. Chen
Department of Electrical Engineering,
National Taipei University of Technology,
Taipei 10608 Taiwan, Republic of China

W.-T. Huang
Department of Computer Science and Information Engineering,
Minghsin University of Science and Technology,
Hsin-Chu 30401 Taiwan, Republic of China

C.-H. Chen (✉)
No. 11, Buzih Lane, Beitun District,
Taichung City 40601 Taiwan, Republic of China
e-mail: chchen@ctust.edu.tw

## 1 Introduction

Portable electronic products such as personal digital assistants, cellular phones, digital still cameras, and MP3 players require nonvolatile memory storage. As flash memory can meet this requirement, it has become an essential component in mobile multimedia devices [2, 3, 10, 12]. However, there are two challenges in using flash memory: its write bandwidth and its capacity. A flash memory write operation is slow and must be preceded by an erase operation, and flash memory capacity is less than that of a hard disk. Yim et al. proposed a compression layer that uses an internal packing scheme (IPS) with the best-fit method ($IPS_{Best\text{-}Fit}$) to overcome these problems [16]. Flash memory supports input/output (I/O) operations [2, 6–9, 11] one page at a time. If compressed data are smaller than one page size they are written to NAND-type flash memory, and internal fragmentation space (IFS) will be generated, degrading the compression effectiveness [16]. A small IFS is therefore desirable to enhance the $IPS_{Best\text{-}Fit}$ compression ratio. However, a large buffer is necessary to search the entire list, and unless the list is kept ordered by size, this causes a great deal of IFS in each page. This compression layer cannot be used in mobile multimedia devices because the read response time is irregular [16]. Therefore, we propose a real-time compression layer architecture to overcome these disadvantages. Since our method can effectively reduce IFS and enhance the compression ratio, it will not only decrease the write response time by 60% but also improve the irregular read response time over that of $IPS_{Best\text{-}Fit}$.

The rest of this paper is organized as follows. Sections 2 and 3 describe related work and the background of this paper. Section 4 proposes the architecture and packaging method of our real-time compression layer. Section 5 presents the results and discussion, and Section 6 is our conclusion.

## 2 Related work

Yim et al. proposed a flash compression layer that used an IPS with the X-match and run-length (X–RL) algorithm because of its desirable properties such as a good compression ratio, small compression unit, and simple hardware implementation. They adopted a best-fit policy which puts the compressed pages into a group to minimize IFS. The prototype implementation and simulation studies showed that their proposed system offered more than 140% of the storage capacity of standard flash memory [16].

L.P. Chang et al. proposed a real-time garbage-collection mechanism which provided guaranteed performance for real-time systems. It also supported non-real-time tasks, so that the potential bandwidth of the storage system could be fully used. It included a wear-leveling method which was executed as a non-real-time service to resolve the flash memory endurance problem [4].

S. Park and S. Y. Ohm proposed new techniques for a real-time file allocation table (FAT) file system in mobile multimedia devices. They proposed two new techniques to solve the problems with the existing FAT file system. The sector reservation method effectively reduced the internal overhead, while the all-cluster pre-allocation method avoided periodic cluster allocation and reduced frequent FAT modifications in the file system. These methods worked well, and the response times were quite deterministic and uniform. This file system enabled stable recording of multimedia data streams in mobile multimedia devices [14].

## 3 Background

### 3.1 Characteristics of flash memory

There are two types of flash memory: NOR and NAND. NAND-type flash memory (NandFlash) is widely used in mobile multimedia devices. The standard characteristics of NandFlash have been discussed in the literature [2, 3, 6–8, 11]. NandFlash array organization is shown in Fig. 1 [6–8]. It has a unified bus for both addresses and data. Serial operation is used to access data though the command, address, and data bus. The access unit is generally one page of 512+16 B, and the erase unit is a block of 32 pages, or 16 KB. The typical minimum serial read time is 50 ns/B and the maximum random read time is 12 μs/B. This random read time is longer than the serial read time because it contains an initial latency time and $t_R$ as shown in Fig. 2 [6–8]. The initial latency is about 2 μs; $t_R$ is the data transfer time from a cell to the register, with a maximum time of 10 μs per transfer [6–8]. In NandFlash, a write operation represents the process of changing a bit from 1 to 0. Once a bit is changed from 0 to 1, it cannot be changed back by another write operation. Moreover, NandFlash cannot be updated in situ because the system must erase a block before writing new data to it. In addition, due to a limitation in endurance, a block that is frequently erased and rewritten will quickly become damaged [2–4, 12]. The characteristics of NandFlash are shown in Table 1.

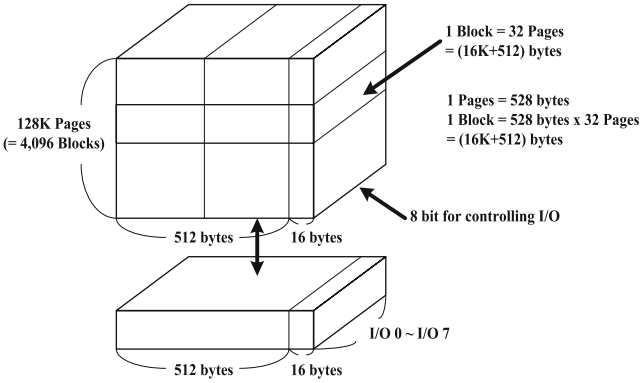



**Figure 1** Array organization of NandFlash [6–8]

### 3.2 Error checking and correction

The error checking and correction (ECC) code, which consists of 3 B for every 256 data bytes with a 1-bit correction capability, is generated by the hardware. We generate and store one ECC code to save space during write operations. During read operations, a new ECC code is generated and then checked with the original ECC code by an XOR operation. If the XOR operation gives a result of zero, the read operation was correct; if not, an error has occurred. The error detection process is shown in Fig. 3 [5].

### 3.3 Compression algorithm

Data compression can increase storage capacity and reduce transmission cost, and some compression algorithms have been applied to flash memory [1, 12, 13]. However, the appropriate algorithm must be selected for this particular application since the compression method must be lossless and operate in real time. Because of the lossless requirement, the compression stage must be complete before the system begins writing data to NandFlash. Lossless compression algorithms include statistical models such as Huffman coding, and dictionary-based models such as the Lempel–Zif–Welsh (LZW) or X–RL algorithms. The dictionary-based model is suitable for flash memory because of its lossless and real-time aspects [12]. Moreover, X–RL has an excellent compression ratio that is suitable to this application [16]. For example, it will set base samples *a*, *b*, and *c* in the initial state of the database, as shown in

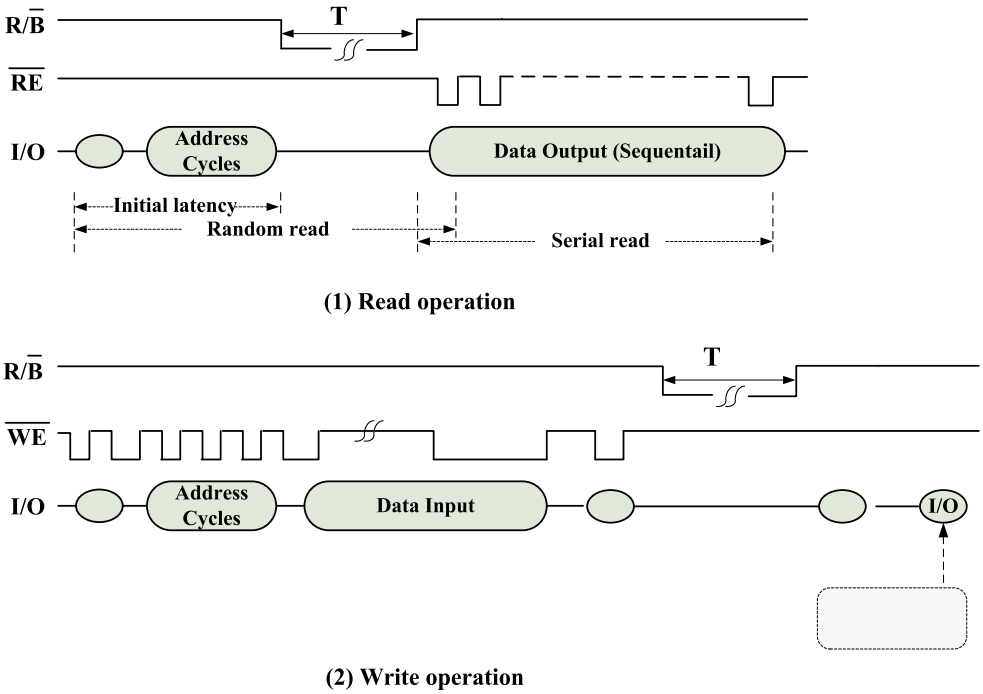



**Figure 2** Timing chart of the read/write operation [6–8]

Fig. 4. Then, it will progressively create new samples with $t$=11 and $t$=15, which are not included in the database. Hence, the new words *ab*, *ba*, *abc*, *ca*, *aba*, *abac*, *cab*, and *bac* will be added to the database. If the input data are *ababcababacabacab*..., the compression algorithm will encode the output string as *013237649* according to the dictionaries in the database. In some simulations and experiments, a database with 128 entries produces the optimum compression ratio. Typically, this method can enhance the storage capacity by 40% [16].

### 3.4 $IPS_{Best\text{-}Fit}$

Yim et al. proposed the $IPS_{Best\text{-}Fit}$ method to support a NandFlash compression layer. The block diagram of the flash compression layer with the internal packing scheme is shown in Fig. 5 [16]. It includes the compression/decompression and the write buffer units. The compression unit is an I/O unit of 512 B compressed by the X–RL algorithm. Since the compressed data are less than one page size, this will cause IFSs if the compressed data are directly written into NandFlash. Therefore, Yim's method used the compressed grouping with the best-fit method to package the compressed data into a write buffer, as shown in Fig. 6. For example, the 2-KB File *A* would be divided into *A1*, *A2*, *A3*, and *A4* according to the I/O unit size. After compression, these become *A1′*, *A2′*, *A3′*, and *A4′*, and will be grouped by the best-fit method. When the buffer is not large enough to store all the compressed data, it will write the smallest IFS buffer from the write buffer to the NandFlash. Yim et al. used a write buffer of 160 KB in their implementation. They also used the best-fit method

**Table 1** Characteristics of NandFlash [6–8]

| NandFlash | Characteristic |
|---|---|
| Page size | 512+16 B |
| Page program size | 512+16 B |
| Block erasure size | 16 K+512 B |
| Data transfer time from cell to register ($T_R$) | 10 μs (max.) |
| Random read time | 12 μs (max.) |
| Serial read time | 50 ns/B (min.) |
| Program time ($t_{PROG}$) | 200 μs (typ.) |
| Write time | 200 μs (typ.) |
| Erasure time | 2 ms (typ.) |

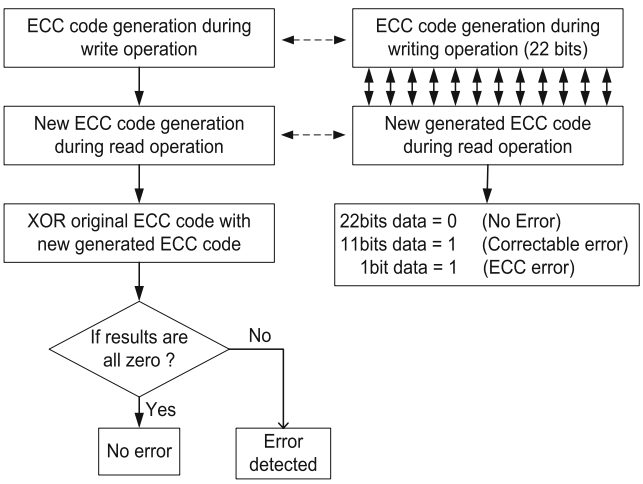



**Figure 3** Error detection process of ECC [5]

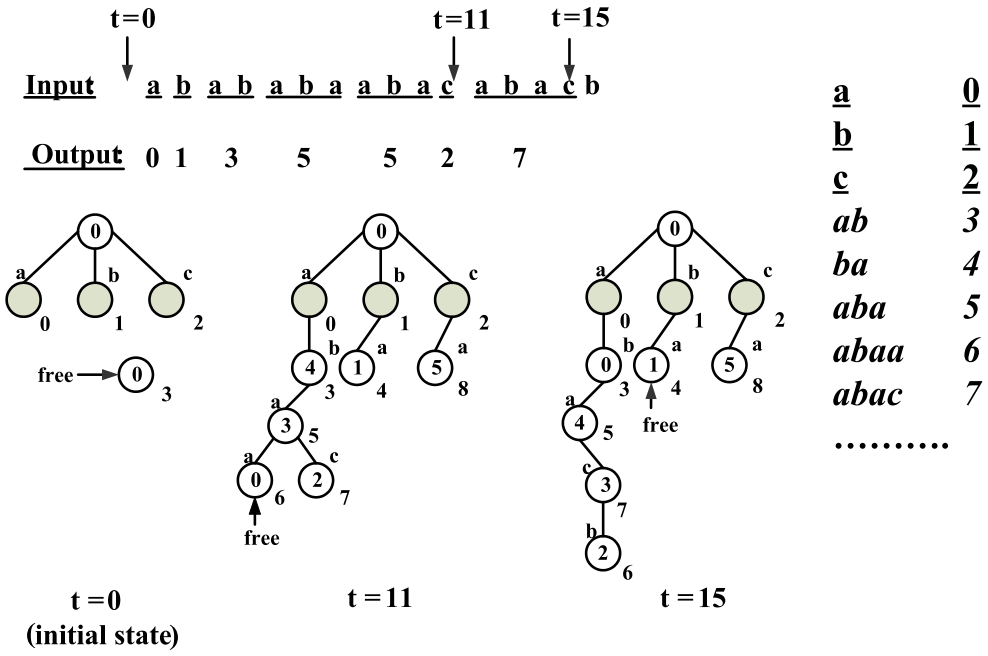



**Figure 4** Example of a dictionary-based compression algorithm

to package compressed data to minimize IFS since they believed that IFS would significantly degrade the data compression effectiveness. This method offered a flash memory storage capacity 40% greater than the original size and significantly expanded the write bandwidth [16].

We know that $IPS_{Best\text{-}Fit}$ will require a lot of time to package the compressed data, as it must search all items to find the best candidate. This will seriously increase the write response time, and will cause irregular read response times, given that the system may randomly read any compressed unit. For example, it requires four random read operations to read File *A* shown in Fig. 6. This irregular read response occurs during random reads, which includes the initial latency time and $t_R$. Furthermore, the IFS is the amount of each fragmentation page size $\left(\sum_{i=1}^{n} \text{Fragmentation\_size}_i\right)$ in $IPS_{Best\text{-}Fit}$, where $i$ is the first fragmentation page and $n$ is the last fragmentation page. We propose a real-time compression layer to overcome these problems.

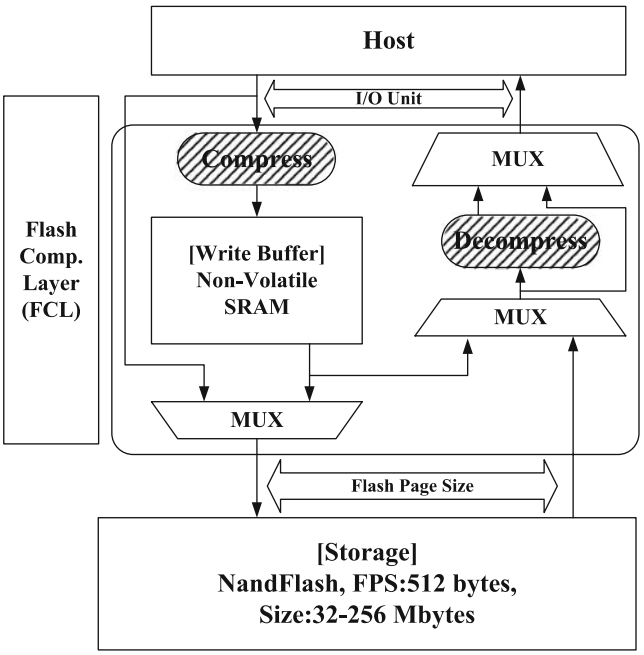


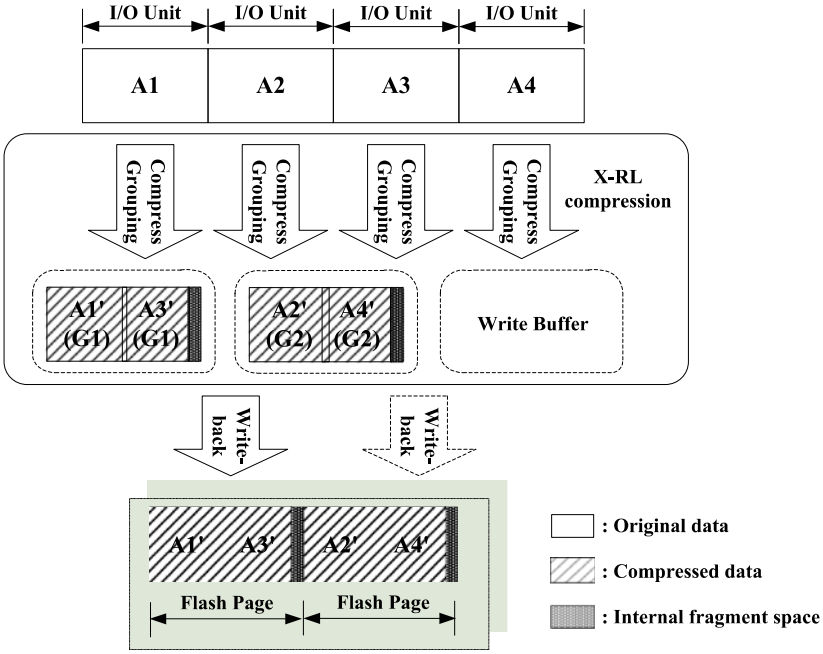

**Figure 5** $IPS_{Best\text{-}Fit}$ architecture [16]

## 4 Real-time compression layer

### 4.1 Architecture

We propose a real-time compression layer called $IPS_{Real\text{-}Time}$, designed for the characteristics of NandFlash, where reading and writing take place one page unit at a time. As each random read has an initial latency of about 12 μs, the random read speed is much less than a serial read (50 ns/B) as shown in Fig. 2, and random read times will not be uniform. Our method uses the X–RL compression/decompression algorithm with contiguous packing, and employs a read/write ping-pong buffer with $Buffer_1$ (512 B) and $Buffer_2$ (512 B) as shown in Fig. 7. When a file is written to NandFlash, the system will generate an ECC code that is simultaneously compressed by the X–RL algorithm. These compressed data are packed by the packing agent. When one write ping-pong buffer is full, the packaged compressed data are written to NandFlash. During a file read, the compressed data in the

**Figure 6** Example of $IPS_{Best\text{-}Fit}$




storage are decompressed by the X–RL algorithm, and a new ECC code is generated. The host obtains the data by reading the ping-pong buffer after checking the ECC code.

Our packing agent uses the contiguous method to pack the compressed data [15] where the data are arranged in the write buffer in order of I/O unit size. For example, the 2-KB File *A* (File *B*) is divided into *A1*, *A2*, *A3*, and *A4* (*B1*, *B2*, *B3*, and *B4*) by I/O unit size as shown in Fig. 8. They are then compressed by X–RL to become *A1′*, *A2′*, *A3′*, *A4′*, (*B1′*, *B2′*, *B3′*, and *B4′*). These compressed data packages are stored by the packing agent in the write ping-pong buffer. The packaged data will be written to NandFlash when one of the buffers is full. The contiguous method does not cause an IFS in each flash page as the best-fit method does. Furthermore, the compressed data to be written to NandFlash appear as full, shared, or partial pages, as shown in Fig. 9. A full page such as page 12 is completely used by one file, a shared page such as page 13 includes two or more files, and a partial page IFS such as page 14 is not completely used by a file. The system records three parameters: page number, offset, and the length of a file or file subpart. As an example, consider three compressed files stored in the write ping-pong buffer by the packing agent in pages 12–14: File *A* with two subparts, *A1* and *A2*, File *B*, and File *C* with two subparts, *C1* and *C2*. Parameters page 12, 0, and 512 are used to record subpart *A1*; page 13, 0, and 100 are used to record subpart *A2*; page 13, 101, and 212 are used to record File *B*; page 13, 313, and 199 are used to record subpart *C1*; and page 14, 313, and 500 are used to record subpart *C2*. These parameters are stored in the NandFlash spare area.

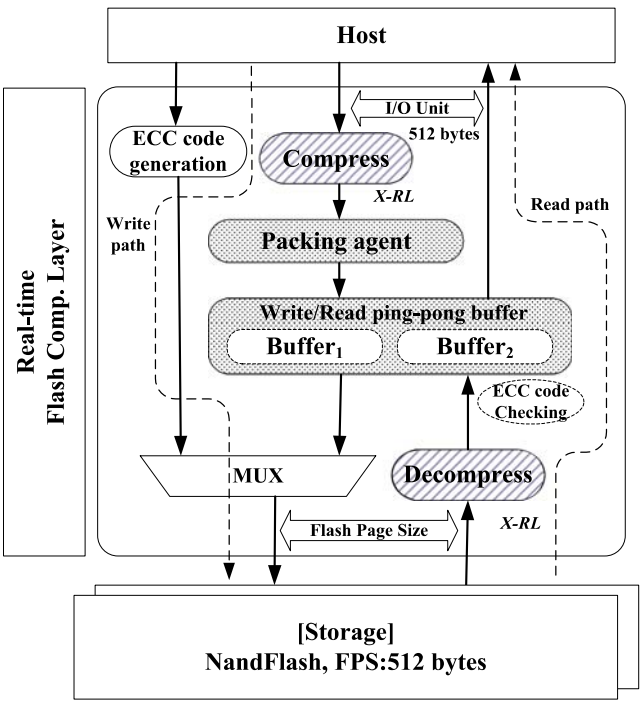



**Figure 7** $IPS_{Real\text{-}Time}$ architecture

The read/write ping-pong buffer can effectively reduce the write response time and stabilize the read response time, enhancing performance, much like a pipelining process. In the write ping-pong buffer, since the write speed of the X–RL compressor/decompressor is faster than the I/O bus speed, there is no overhead incurred [16]. Therefore, our method can use $Buffer_1$ to package data compressed by the packing agent, and $Buffer_2$ can be used simultaneously to

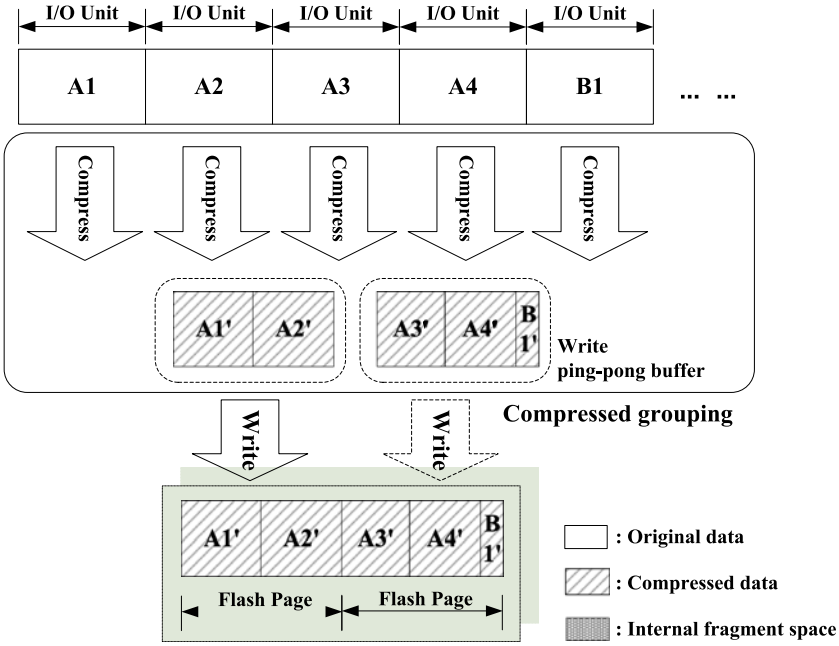



**Figure 8** Examples of $IPS_{Real\text{-}Time}$

write the packaged data into NandFlash. Conversely, the host can obtain data from the read ping-pong buffer after the decompression process and check the ECC code. At the same time that $Buffer_1$ presents the data to the host, $Buffer_2$ decompresses and checks the ECC code.

### 4.2 Definition of the response time

The write and read response times are defined as follows. Let $T_{compress}$ be the processing time of the X–RL algorithm, $T_{pack}$ be the packing agent time, and $T_{PROG}$ be the page programming time. The page write time $PT_{writing}$ can be expressed as:

$$PT_{writing} = T_{compress} + T_{pack} + T_{PROG}. \quad (1)$$

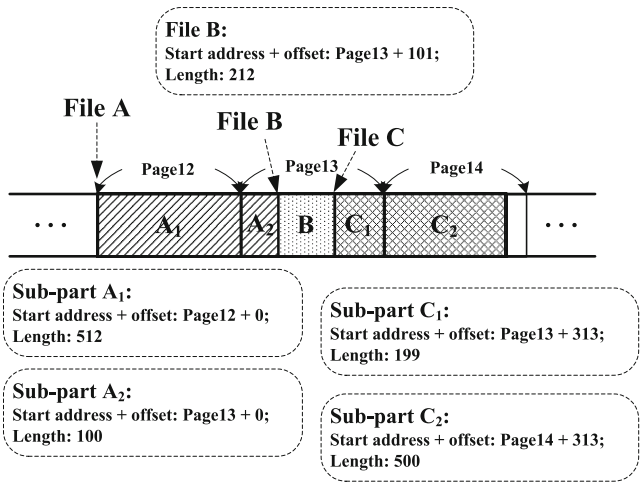



**Figure 9** Compressed data in the page conditions of NandFlash

Let $T_R$ be the time of the page read operation from the cell to the register, $T_{decompress}$ be the decompression time of X–RL, and $T_{ECC\_check}$ be the time for the ECC code check in the buffer. Then the page read time $PT_{reading}$ can be expressed as:

$$PT_{reading} = T_R + T_{decompress} + T_{ECC\_check}. \quad (2)$$

## 5 Results and discussion

The critical issues of the read/write response time and IFS comparison between $IPS_{Best\text{-}Fit}$ and $IPS_{Real\text{-}Time}$ have been discussed. The detailed evaluation parameters are shown in Table 2. A comparison of the write response time between Yim's method and ours is shown in Fig. 10. The normalized write response times of $IPS_{Real\text{-}Time}$ and $IPS_{Best\text{-}Fit}$ are 0.4 and 1, respectively, showing that $IPS_{Real\text{-}Time}$ can reduce the write response time by 60%. The reason for this is that $IPS_{Best\text{-}Fit}$ requires much more time for $T_{pack}$ because it uses the best-fit

**Table 2** Evaluation parameters

| System parameters | Values |
|---|---|
| Processor | 100 MHz |
| Random read time | 12 μs/B |
| Serial read time | 50 ns/B |
| Program time ($t_{PROG}$) | 200 μs |
| Buffer access time | 80 ns/B |
| $T_{compress}$, $T_{decompress}$ | 1 μs |
| $T_{ECC\_check}$ | 0.1 μs |
| I/O port | 8 bit |

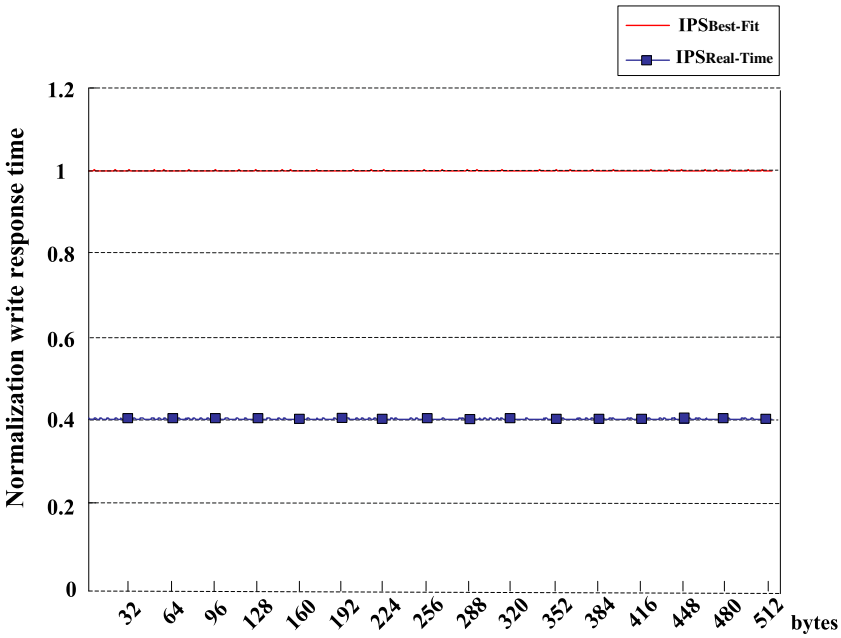



**Figure 10** Normalization write response time between $IPS_{Best\text{-}Fit}$ and $IPS_{Real\text{-}Time}$

method to obtain the minimum IFS. In contrast, given that $IPS_{Real\text{-}Time}$ uses a contiguous method to pack compressed data, it does not require much time to search each item.

A comparison of the read response time performance for $IPS_{Best\text{-}Fit}$ and $IPS_{Real\text{-}Time}$ is shown in Fig. 11. The irregular read response time peak occurs every 512 B in $IPS_{Best\text{-}Fit}$ because the best-fit method performs a random read for each compressed unit. The read ping-pong buffer in our method avoids this problem. Since the system needs to store the decompressed data into $Buffer_1$ and then check the ECC code, there is only a read-time peak in $IPS_{Real\text{-}Time}$ at the initial step. After this initial step, while the host reads data from $Buffer_1$, the system can finish preparing the next data to be stored into $Buffer_2$. The preparation of the next data in the second buffer will be complete before the host is finished reading data from the first, and this alternates for the next cycle. Therefore, $IPS_{Real\text{-}Time}$ ensures a stable read response time. The IFS for $IPS_{Best\text{-}Fit}$ is $\sum_{i=1}^{n} \text{Fragmentation\_size}_i$ and the IFS for $IPS_{Real\text{-}Time}$ is zero. Therefore, $IPS_{Real\text{-}Time}$ has a better compression ratio than $IPS_{Best\text{-}Fit}$.

Although Yim et al. wanted to obtain the minimum IFS using the best-fit method for their packing agent, this was at the cost of a large packing time which leads to an irregular read response time. Our proposed methodology is closely matched to the characteristics of NandFlash and deals with these issues effectively. Compared with $IPS_{Best\text{-}Fit}$, our method stabilizes the read response time, reduces the write response time, and enhances the compression ratio.

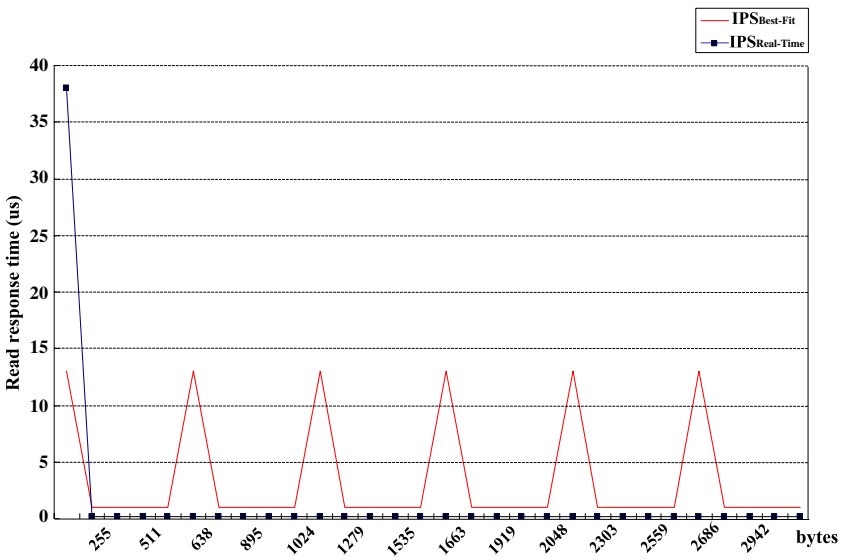



**Figure 11** Comparison of the read response time between $IPS_{Best\text{-}Fit}$ and $IPS_{Real\text{-}Time}$

## 6 Conclusion

A compression layer is necessary to increase the capacity and bandwidth of flash memory in mobile multimedia devices. In this paper, we have proposed a real-time compression layer for flash memory. Compared to the method of Yim et al., our method stabilizes the read response time and reduces the write response time by 60%. Furthermore, our method enhances the compression ratio because it fully uses each page without generating IFS. We have confirmed that our new method can support real-time applications in mobile multimedia devices.

## References


1. Bunton S, Borriello G (1992) Practical dictionary management for hardware data compression. Commun ACM 35(1):95–104 doi:10.1145/129617.129622
2. Bez R, Camerlenghi E, Modelli A, Visconti A (2003) Introduction to flash memory. Proc IEEE 91(4):489–502 (Apr) doi:10.1109/JPROC.2003.811702
3. Chang LP, Kuo TW (2005) Efficient management for large-scale flash-memory storage systems with resource conservation. ACM Transactions on Storage 1(4):381–418 (Nov) doi:10.1145/1111609.1111610
4. Chang LP, Kuo TW, Lo SW (2004) Real-time garbage collection for flash-memory storage systems of real-time embedded systems. ACM Transactions on Embedded Computing Systems (TECS) 3 (4):837–863
5. Samsung Corp. (2004) NAND Flash ECC algorithm, product planning and application engineering team. Available at http://www.samsung.com. Accessed June
6. Samsung Corp. (2001) K9F1208U0M-YCB0, K9F1208U0M-YIB0, K9F5608U0M-YCB0 flash memory. Data sheet. Available at http://www.samsung.com.
7. Samsung Corp. (2000) Flash/SmartMedia/file system. Memory Databook
8. Toshiba Corp. (2001) TC58256AFT 256-MBIT (32M*8 BITS) CMOS NAND E2PROM. Data Sheet
9. Dipert B, Levy, M (1994) Designing with FLASH MEMORY. Chapter 9: Flash memory files system (pp. 227–271). San Diego: Anna Books
10. Douglis F, C'eres R, Kaashoek F, Li K, Marsh B, Tauber JA (1994) Storage alternatives for mobile computers. Proceedings OSDI94, pp. 25–37
11. Flash Memory Databook, Intel (1995)
12. Huang WT, Chen CT, Chen YS, Chen CH (2005) A compression layer for NAND type flash memory systems. ICITA'2005 1:599–604 (July)
13. Kjelso M, Gooch M, Jones S (1996) Design and performance of a main memory hardware data compressor. In Proceedings of the 22nd Euromicro Conference, IEEE Computer Society Press, pp 422–430
14. Park S, Ohm SY (2006) New techniques for real-time FAT file system in mobile multimedia devices. IEEE Trans Consum Electron 52(1):1–9 (Feb) doi:10.1109/TCE.2006.1605017
15. Silberschatz A, Galvin P, Gagne G (2004) Operating system concepts. Wiley, New York
16. Yim KS, Bahn H, Koh K (2004) A flash compression layer for SmartMedia card systems. IEEE Trans Consum Electron 50 (1):192–197 (Feb) doi:10.1109/TCE.2004.1277861


**Chin-Hsing Chen** (chchen@ctust.edu.tw) is an associate professor of Department of Management Information Systems, Central Taiwan University of Science and Technology, Taiwan. She received her Ph.D. in Applied Mathematics from National Chung-Hsing University, Taiwan. Her research interests include the embedded system applications, flash memory algorithms, QoS in computer networks, and medical information systems.

**Chun-Ta Chen** (s3419002@ntut.edu.tw) received his B.S. and M.S. degree in Electronic Engineering from National Taipei University of Technology in 2002 and 2004. He is currently the Ph.D. candidate in the Department of Electronic Engineering, National Taipei University of Technology, Taipei, Taiwan. His research interests include flash memory system, algorithm, HW/SW co-design, computer architecture, system on chip, and embedded system.

**Wen-Tzeng Huang** (wthuang@must.edu.tw) received his Ph.D. degree from the Department of Computer and Information Science, National Chiao-Tung University, in 2001. He is currently an associate professor in the Department of Computer Science and Information Engineering, Minghsin University of Science and Technology, Taiwan. His research interests include the flash algorithms, telemedicine and telecare over wireless sensor network applications, and digital high-speed printed circuit board simulations and designs. He is member of IEICE.