On the Effect of Swap in Application Switching for Android Platform with Pattern Analysis of eMMC

Jae Gon Lee [1,2] and Jun Dong Cho [1]
[1] Sungkyunkwan University, Suwon, Korea
[2] Samsung Electronics Co. Ltd., Hwasung, Korea
Email: jg47.lee@samsung.com, goirisia@skku.edu, jdcho@skku.edu

Kyu Min Park, Byung Yo Lee, Hak Yong Lee, Kwang Won Park and Kwan Yong Jin
Samsung Electronics Co. Ltd., Hwasung, Korea
Email: kyumin.park@samsung.com, yo0831.lee@samsung.com, hakyong.lee@samsung.com, junhaa.park@samsung.com, kwanyong.jin@samsung.com,

***Abstract*—Currently, smartphone users often dissatisfy from the specifications provided by the manufacturer while using some services. Using swap operation is one way to solve this problem. In this paper, we evaluate the effect of swap in application switching for Android platform. By utilizing swap, the number of active applications is increased, and the application loading time is decreased, improving users experience performance. In addition, due to less time consumption, the swap also contributes to reduce the total energy consumption. Finally, from our experiments, in order to better support the swap, we realized that the sequential performance of storage with large I/O is important due to storage feature.**

***Index Terms*—swap, application switching, storage I/O pattern of swap, eMMC, android, smartphone**

I. Introduction

Recently, various smart phone products with competitive price expect the maximum performance adopting the octa-core application processor (AP) in mobile environment. Smartphone users search the information, share their daily life and enjoy a game through various applications that require more memories and storages.

The memory and storage components are allocated by hardware intellectual property (IP), and used by operating system (OS) in order to provide some service. Equipped memory is smaller than equipped storage because of memory hierarchy. Therefore, the memory restriction affects considerable user's dissatisfaction in the low price model. Thus, the efficient memory management is important in order to ensure the various features of a smartphone. Using swap is one way to solve this problem. Swap is a memory management technique that the storage is used as virtual memory space, thus the system have the virtually more memory space than physical memory. [1]

Android is based on Linux that supports swap feature. However, currently smart phone does not use swap in Android platform. Because embedded Multi-Media Card (eMMC) that is the storage for Android platform had low performance and lifetime limit, thus the system performance was degraded if we use swap. [2]

Nowadays, the advanced eMMC outperforms 7.7 times I/O performance than the previous eMMC, providing some performance improvement features like packed command.[3]-[5] Thus current storage eliminates the overhead of swap caused by low performance.

In this paper, in order to find the effect of swap for Android platform, we implement swap in a smartphone, GT-I9300, and then perform experiments for application switching. We observe the number of active application when application is loaded, and measure the reduction of time and system energy caused by difference of number of active application. Also, we analyze the swap pattern of eMMC using logic analyzer and blktrace tool.

This paper is organized as follows. Section II reviews the fundamental of swap and NAND flash memory, and activity/memory management in Android and eMMC. Section III performs experiment to identify the effect of swap point of view the system. Section IV shows the swap pattern analysis in eMMC. Finally, we conclude in Section V.

II. Related Work

A. Virtual Memory and Swap

Virtual memory overcomes the limited size of physical memory such that OS should be used efficiently. User requires more memory than the amount of physical memory. Virtual memory is a memory management technique that some parts of the processes data are stored in the storage instead of the physical memory, i.e. the

Manuscript received August 1, 2013; revised September 23, 2013.

©2014 Engineering and Technology Publishing

doi: 10.12720/ijoee.2.1.82-88

storage is used as virtual memory space. Especially, virtual memory is effective to support a multi-tasking environment that runs variety of programs that are loaded and executed in memory.

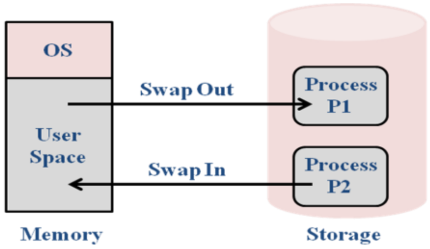

Figure 1. The concept memory swap.

Swap is a method how to implement a virtual memory when the processes attempt to allocate more memory than physical memory. And then OS begins to swap memory pages to and from the storage. High priority process starts to run and the amount of available memory drops below a certain level. OS stores some of memory pages of the lower priority processes onto the storage and increases the amount of available memory; it is called swap-out. If the process of the low priority gets to be performed, OS store the victim data that was stored in storage before to the memory; it is called swap-in as shown in Fig. 1. Replacement policies are First In First Out (FIFO), Least Recently Used (LRU), Least Frequently Used (LFU) and so on.

B. *Android Activity and Memory Management*

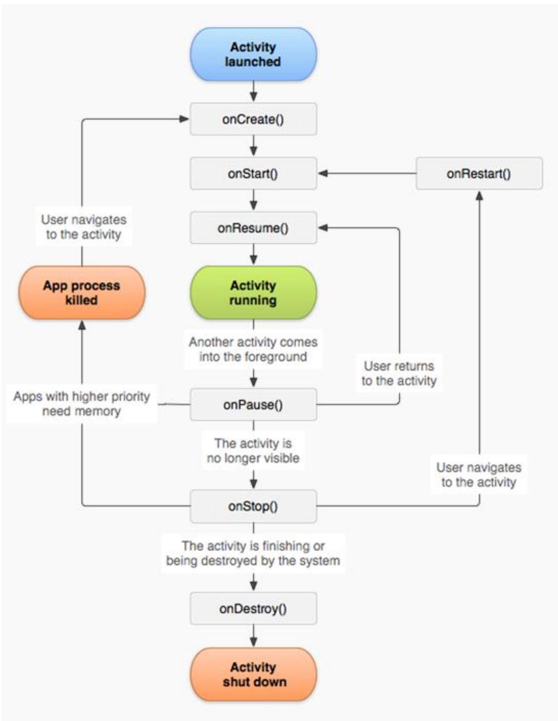

Figure 2. The activity life cycle in Android.

An activity is the main component in applications for interacting with users. It is a single user-interface (UI) on the screen and focus user's action. The application has different screens and each screen is implemented other activities. When the screen is changed, the new activity performed and existing activity is stored in a stack. Each activity is managed through a stack. Fig. 2 describes the life cycle of an Activity in Android.[6] Activity has 7 methods to change its state. If application is loaded for the first time or there was no data about application in the memory, activity initializes application through the method called onCreate(). But if data of the application remains in the memory, application is reloaded without onCreate() and number of block I/O is reduced from storage, so reloading time is shorter than first loading time.

Android increases available memory space using Out Of Memory Killer (OOMK) and Low Memory Killer (LMK) when memory is insufficient. OOMK is the process used in Linux in order to kill the fewest number of applications and increase the available memory space.[7] OOMK does not give any priority to process, so the important application might be removed.

LMK classifies process as six types according to the importance shown in Table I. Each type has the minimum value of available memory space in Android. In low memory situation LMK starts killing the process from low priority groups.

TABLE I. SIX TYPES OF PROCESSES ARE CLASSIFIED BY LMK

Process Type	Description
FOREGROUND_APP	This is the process running the current foreground app
VISIBLE_APP	This is a process only hosting activities that are visible to the user
SECONDARY_SERVICE	This is a process holding a secondary server
HIDDEN_APP	This is a process only hosting activities that are not visible
CONTENT_PROVIDER	This is a process with a content provider
EMPTY_APP	This is a process without anything currently running in it

C. *Nand Flash Memory and eMMC*

As shown in Fig. 3, flash memory cell consists of one transistor with a floating gate. Information is stored in floating gate. There are 2 kinds of flash memory, NAND and NOR but currently the dominating type is NAND Flash. So we consider only NAND flash memory in this paper. NAND flash memory is classified into two types, single-level-cell (SLC) and multi-level-cell (MLC). SLC stores only a single bit of data in flash cell and MLC stores data more than 2 bit in flash cell. 3 bit MLC was commercialized recently. [8]

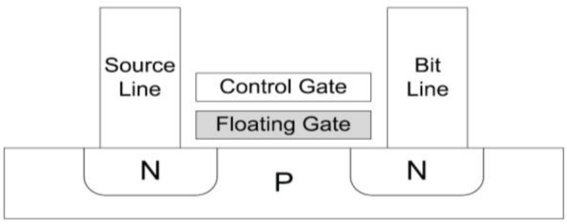

Figure 3. The fundamental cell in flash memory.

©2014 Engineering and Technology Publishing

NAND flash memory has 3 basic operations such as read, write, erase. Electrons are trapped onto the floating gate and these electrons modify the threshold voltage of control gate. Read operation distinguishes the difference of the threshold voltage between 0 and 1. Default state of NAND flash memory is logically equivalent to a binary 1 value. Write operation (also referred to as program) changes the status to 0 by programming the floating gate. Erase operation is bringing back the state of flash memory to its default state with value 1.

NAND flash memory comprises billions of flash cells that are organized in a hierarchical architecture like page and block as depicted in Fig. 4. Page is a basic unit to operate read and write. It has various size of 4KB ~ 16KB. A block consists of multiple flash pages and is in unit of erase operation.

Write operation of NAND flash memory has one limitation. It has to operate to certainly erase before write because overwriting is impossible. In other words, a flash cell has to be erased before it can be re-write. The number of write and erase is limited because of flash's characteristic. The available program/erase (P/E) cycles of 3 bit MLC 1000 times worse than SLC as shown in Table II. [9]

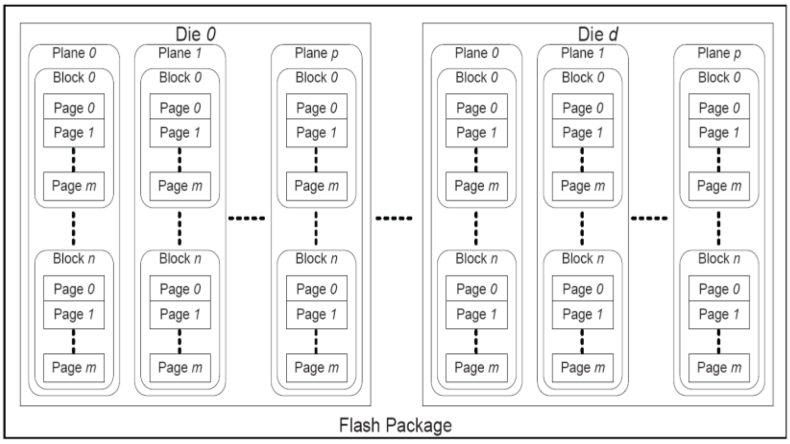

Figure 4. The architecture of a typical NAND flash memory.

TABLE II. ENDURANCE OF NAND FLASH MEMORY

	SLC	2 bit MLC	3 bit MLC
Bit per Cell	1	2	3
P/E Cycle (Endurance)	100,000	3,000	1,000

TABLE III. INTERFACE FEATURE OF EMMC

Bus speed modes	Frequency	Max Data Transfer
Backwards Compatibility with legacy MMC Card	0~26MHz	26MB/s
High speed SDR	0~52MHz	52MB/s
High speed DDR	0~52MHz	104MB/s
HS200	0~200MHz	200MB/s

There are many kinds of products based on NAND flash memory like solid-state drive (SSD), secure digital (SD) card and eMMC. Of these, the eMMC is used with the storage of the smart phone, tablet and etc. The latest specification of the eMMC supported form the JEDEC is released version 4.5. This product supports the max clock frequency of 200MB/s. Therefore, processing speed of command and response improves 4 times and processing time of data improves 2 times compared to previous 52MB/s as shown in Table III.

Additionally, some performance improvement features had been added like Context ID, Packed Command, and Trim & Discard. [9]

III. SWAP EFFECTIVENESS ANALYSIS DUE TO THE APP SWITCHING

If application starts loading, associative data is stored in the memory and available memory space is decreased that much. If some of new applications are executed continuously, the available memory space is insufficient and LMK terminates one or some of executed applications. Then related data of the terminated applications is deleted from the memory and attempts to load that application again, reloading time is same as first loading. If application is alive and related data is remained in the memory, (that situation is called active application) active application runs immediately so reloading time is shorter than first loading.

To use swap in Android platform increases available memory space because some of data of active application are written for swap-out to the storage. So the case of application termination due to the lack of memory decreases and accordingly the number of active applications can be increased. Users have their frequently used application in daily life and use them repeatedly. Increasing the number of active applications due to swap reduces the loading time of the application and contributes to improving users experience performance. So we experimented on the effect of swap with the repeated switching of 10 applications loading and analyzed in terms of system effectiveness.

©2014 Engineering and Technology Publishing

A. Experiment Condition

TABLE IV. SPECIFICATION OF GT-I9300 SMARTPHONE

Model	CPU	DRAM	Storage	OS
GT-I9300	Exynos 4412 1.4GHz	1GB	32GB	Jelly Bean 4.1.1

GT-I9300 that we used is commercial Android smartphone on Samsung and detailed specifications of it are follows in Table IV.

Storage was used the 3bit MLC of eMMC and was deleted 5% after full write in the user area for severe pre-condition. We built the kernel of GT-I9300 to use swap and experiment on the effect of swap using Android Debug Bridge (ADB). We created swap file allocated 1GB in eMMC and used the commercial game applications as follows in Table V.

TABLE V. THE GAME APPLICATIONS USING THE EXPERIMENT

Application Name	Execution size on DRAM[MB]	Loading time[sec]
Ben 10 : Xenodrome	39.15	7.12
Bomb The Zombies	13.89	4.84
Drag Racing	23.49	4.31
Cordy2	45.13	13.81
Lep's world	19.44	4.00
Tap sonic	19.54	5.81
Death Worm	43.51	5.41
PrettyPetSalon	8.79	2.06
Angry Birds Rio	25.75	8.44
Krazy Kart Racing	5.81	8.87

As shown in Fig. 5, memory density of GT-I9300 is 1GB but users can use the free space area of about 460MB that is only 45% of the total space. The Reserve area (Res.) was used by hardware IP and OS (or by Telecommunications Company or smartphone manufacturer) used the used area in order to provide some service. OS recognizes used area and free area so ratio of memory space between used area and free area are 45% and 55% by OS. Thus, we set the swappiness to 77 when the App. begins swap.

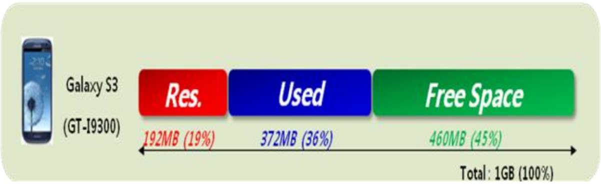

Figure 5. The memory usage in GT-I9300.

Figure 6. The application switching sequence.

Experimental scenario is loading ten game applications and switches each applications three times more as shown in Fig. 6. Repeating ten applications by in order of precedence is very severe condition and users use these applications arbitrarily so that doesn't represent the user scenario. Therefore, we classified the experiment as two conditions and evaluated four cases. Random order of ten applications without swap is default condition in GT-I9300. These four cases of scenario are organized as follows.

Case 1: Random order with swap
Case 2: Random order without swap
Case 3: In order of precedence with swap
Case 4: In order of precedence without swap

One of application is switched to HIDDEN_APP when loading is complete. In case of first loading was measured from onCreate() to onStop() by timer. In case of reloading was measured form onCreate() or onRestart() to onStop(). If data of the application remains in the memory, active application is reloaded through the method called onRestart() without onCreate. Power is measured voltage drop across a shunt resistor connected to battery by Keithley. Using this value and measured time, we calculate the amount of system energy.

B. Experiment Result

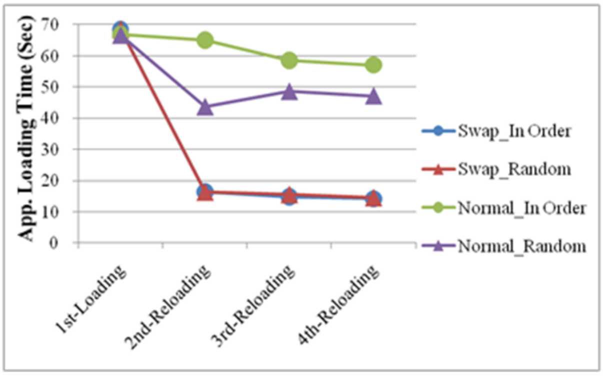

Figure 7. The loading time of 4 cases.

Fig. 7 shows the result of loading time about four cases scenario. In case of using swap, it had the similar result regardless of the order of ten applications. However, it had the difference of result according to the order of ten applications when system is not supported swap. In case 4, the previously executed application can be terminated for available memory space expansion. As a result 2nd-Reloading time was similar to 1st-Loading because all of ten applications had been killed before they have been reloaded. Krazy Cart Racing application that is 10th applications in scenario was not terminated, continued with remained active application in 3rd-Reloading and 4th-Reloading, so 10% time gain occurred. In case 2, the number of active applications increased as compared with case 4, there were four active applications existing on an average in 2nd/3rd/4th-Reloading.

Therefore, it had the gain of total loading time about 17%.

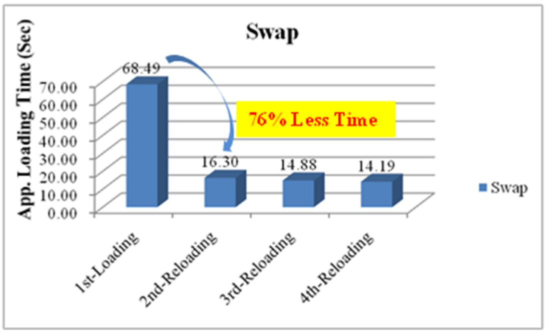

Figure 8. The loading time of case 1.

If swap is supported in system than swap-out occurs from first application loading. As a result, all of applications remain active and some of related data continue to remain in the memory without swap-out data. In this case of reloading, some data can be run directly from the memory and others are loaded from the eMMC so execution speed improvement occurs about 76% as shown in Fig. 8 and users feel fast to execute the application reloading. The effect of swap has the gain of total loading time about 54% as compare with case 4 and 45% as compare with case 2. However, the overhead due to the use of swap occur 1.53s in 1st-Loading.

Additional energy consumption increases 3% more due to swap in 1st-Loading as shown in Fig. 9. But comparing case 1 and case 4, reduction of application switching time affects decreasing energy consumption about 54% as shown in Fig. 10.

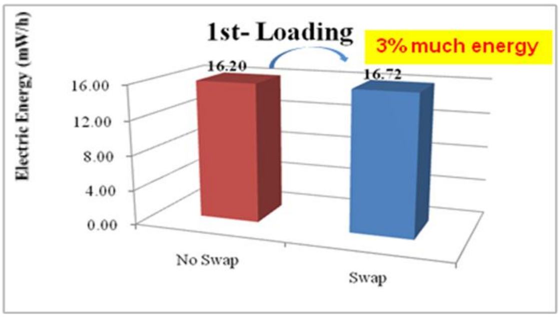

Figure 9. The comparison of energy consumption for first-Loading in swap and without swap.

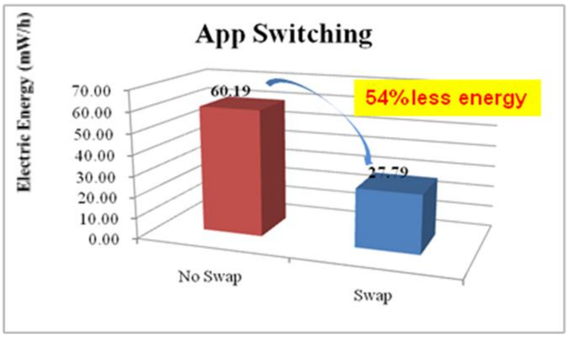

Figure 10. The comparison of energy consumption for application switching in swap and without swap.

IV. Swap Pattern Analysis in Storage

A. *Timing Analysis of eMMC in Appication Loading*

TABLE VI. The Accessed Time and Data Size in 1st-Loading

Swap	Total Time[s]	Write		Read	
		Time[s]	Data[MB]	Time[s]	Data[MB]
On	68.49	1.26	35.15	3.32	189.01
Off	66.96	0.17	2.15	3.31	186.91
Gap	1.53↑	1.09↑	33.00↑	0.01↑	2.10↑

We measured the I/O pattern from to eMMC using Logic Analyzer and blktrace tool. Table VI shows eMMC accessed time and data size classified by read and writes in 1st-Loading at ten applications in order of precedence.

71% out of overhead arising from using swap is a data write time for the swap-out. There are only 2.15MB of write operation for meta update when system is not supported swap. However, 33MB of write operation for swap-out occurs frequently when system is supported swap.

Table VII shows the result of eMMC accessed time and data size classified by read and write in application switching in order of precedence. 89.66MB of write operation in application switching occurs. 3 bit MLC is fatal endurance due to 12.5 times write operation in swap condition. Total data size for read is 906.24MB because all data is needed in loading application except Krazy Cart Racing application when system is not supported swap. However if system uses swap, swap-out data is needed in reloading application and total data size for read is 637.87MB so it takes as little as 274MB.

TABLE VII. The Accessed Time and Data Size in Application Switching

Swap	Total Time[s]	Write		Read	
		Time[s]	Data[MB]	Time[s]	Data[MB]
On	113.86	4.15	89.66	11.32	631.87
Off	247.73	2.11	7.13	16.42	906.24
Gap	133.87↓	2.04↑	82.53↑	5.1↓	274.37↓

©2014 Engineering and Technology Publishing

B. Swap Data Pattern Analysis

TABLE VIII. THE MAJOR I/O SIZE OF EMMC SWAP PATTERN

Write			**Read**		
IO [KB]	**Command Por[%]**	**B/W [MB/s]**	**IO [KB]**	**Command Por[%]**	**B/W [MB/s]**
4	18.70	5.84	4	11.20	19.95
8	8.31	9.11	8	0.45	37.19
16	3.90	14.64	16	1.24	45.80
32	3.12	20.00	32	22.74	60.77
64	0.52	27.75	64	16.29	74.40
128	0.52	46.12	128	33.71	78.48
Packed	11.43	45.33	256	1.70	79.95

Table VIII shows the command portion and bandwidth classified by major I/O size of eMMC swap pattern. GT-I9300 smartphone used 11% of packed command in this scenario. I/O size of packed command spread from 524KB to 3904KB and total data size for swap-out using packed command is 58.41MB. Swap-out data size out of total write was almost 94% and sequential pattern for Swap-out was measured more than 81% in Table IX. Packed command affected increasing sequential pattern of eMMC. Swap-in data size out of total read was almost 11% and sequential pattern for Swap-in was measured more than 55%. Thus, we identify that sequential performance of the eMMC is very important in particular, due to the large I/O size in application switching with swap scenario.

TABLE IX. THE DATA SIZE, BANDWIDTH AND SEQUENTIAL PORTION OF SWAP PATTERN

Swap Enable	**Write**			**Read**		
	Data [MB]	**B/W [MB/s]**	**Seq. [%]**	**Data [MB]**	**B/W [MB/s]**	**Seq. [%]**
Total data	89.66	21.60	37.71	631.87	55.82	14.17
Swap data	84.25	34.03	81.30	68.36	73.92	55.66

V. CONCLUSION & FUTURE WORKS

In this paper, we study about the effect of swap in Android platform. Firstly, we evaluate the effect of swap to increase number of active applications so as to reduce the application loading time and to improve user experience performance. Secondly, the system energy consumption was reduced. Thirdly, we investigate swap pattern of eMMC and effects of packed command for write operation.

For future works, we need to optimize the eMMC for swap. The eMMC supports user partitions that enhance mode. Thus, we need to implement the swap partition mapping to user partition of eMMC.[3] Enhanced mode of eMMC was changed from 3 bit MLC to SLC, thus the eMMC has 100 times margin of endurance of eMMC.

REFERENCES

[1] A. Silberschatz, P. B. Galvin, and G. Gagne, *Operating System Concepts*, 8th ed., John Wiley & Sons Inc., 2008, ch. 8-10.

[2] D. Bornstein, "Dalvic vm internals," presented at 2008 Google I/O Developer Conference, San Francisco, California, May 28-29 2008.

[3] eMMC Standard (4.5 Device), JESD84-B45 Specification, JEDEC Solid State Technology Association, 2011.

[4] H. Kim, N. Agrawal, and C. Ungureanu, "Revisiting storage for smartphones," *ACM Transactions on Storage* (TOS), vol. 8, no. 4, no. 14, 2012.

[5] H. Kim and D. Shin, "Optimizing storage performance of android smartphone," presented at 7th International Conference on Ubiquitous Information Management and Communication, Kota Kinabalu, Malaysia, January 17-19 2013.

[6] Activity life cycle. [Online]. Available: http://developer.android.com/reference/android/app/Activity.html

[7] M. Gorman, *Understanding the Linux Virtual Memory Manager*, Prentice Hall Professional Technical Reference, 2004, ch. 12-13.

[8] R. Bez, E. Camerlenghi, A. Modelli, and A. Visconti, "Introduction to flash memory," *IEEE*, vol. 91, no. 4, pp. 489-502, April 2003.

[9] M. C. Yang, Y. H. Chang, C. W. Tsao, and P. C. Huang, "New ERA: New efficient reliability-aware wear leveling for endurance enhancement of flash storage device," presented at 50th Annual Design Automation Conference, Austin, TX, USA, May 29-June 07 2013.

Jae Gon Lee was born in 24th of December 1980 at Pohang, Korea. He is pursuing his Master degree from Sungkyunkwan University (SKKU) and he is in charge of the eMMC planning and mobile application engineering. He has been with Samsung Electronics Co. Ltd., Memory division, Hwasung, Korea, since 2007.

Kyu Min Park was born in 2nd of July 1984 at Seoul, Korea. She majored in computer science from Soongsil University (SSU). She joined Samsung Electronics in 2007 and she is in charge of the eMMC planning and mobile application engineering.

Byung Yo Lee was born in 31th of August 1976 at Seoul, Korea. He majored in electronics engineering from Korea University. Since 2003, he is in charge of hardware development in SK Teletech such as Camera, LCD, Power and Audio. He joined Samsung Electronics in 2007, has worked as a senior engineer for eMMC products in the application engineering team.

Hak Yong Lee was received his Master's degree in the Department of Electronics and Computer Engineering in 2013 from Hanyang University and his Bachelor's degree in Information and Communication engineering from Sungkyunkwan University, Korea in 2003. Since 2003, he is an engineer of Samsung Electronics in Hwasung-City, Gyenggi-Do, Korea. His research interests include embedded systems, file system, and flash memory.

©2014 Engineering and Technology Publishing

Kwang Won Park was born in 17th of December 1974 at Seoul, Korea. He received the B.S. degree in Electronics Materials Engineering from Kwangwoon University in 2000. He has been with Samsung Electronics Co. Ltd., Hwasung, Korea, since 2000. He has worked as a associate engineer for SDR, DDR, Rambuse DRAM, NOR Flash products in the Memory PA Team (2000~2004) & has worked as a engineer for Mobile Memory, Flash Memory products in the Application Engineering Team(2005~).

Kwan Yong Jin was born in 30th of April 1966 at Wonju, Korea. He received the B.S. degree in electronics engineering from Inha University, Inchen Korea, in 1988. He has been with Samsung Electronics Co. Ltd., Korea, since 1990 and he has worked as a associate engineer for EDP DRAM, Mobile DRAM and Flash products in the application team.

Prof. Jun Dong Cho received Ph.D. degree from Northwestern University, Evanston, IL, 1993, in computer science. He was a Senior CAD Engineer at Samsung Electronics, Co., Ltd. He is now Professor of Dept. of Electronic Engineering, Sungkyunkwan University, Korea. He received the Best paper award at the 1993 Design Automation Conference. He has been an IEEE Senior Member since April 1996. His research interests are in the area of System on Chip Design Optimization and lower power designs.

©2014 Engineering and Technology Publishing