國 立 中 正 大 學

資訊工程學系研究所

碩 士 論 文

應用於快閃記憶體之高效能且低複雜度之平面化設計技術

A Low-Complexity High-Performance Wear-Leveling Algorithm for Flash Memory System Design

研 究 生：薛甯謐

指導教授：鍾菁哲 博士

中華民國 一零一 年 七 月

摘要

在此論文中 ，我們針對快閃記憶體中平面化技術提出討論，並重建 Greedy 垃圾收集方法及靜態平面化演算法作為與本論文所提出之高效能且低複雜度之平面化設計技術（循序垃圾收集技術, SGC)的比較對象，用來延長快閃記憶體的壽命及平衡其使用率。

論文中包含目前現有的快閃記憶體的種類及優缺點，並介紹快閃記憶體中需要哪些重要元件來維持讀寫速度及壽命。此外會介紹目前現有的平面化技術有哪些類別，以及為什麼需要平面化技術，針對現有的各種平面化技術的優缺點來做分析，並討論對於現在的快閃記憶體這些技術是否適用。

本論文所提出之平面化技術稱為循序垃圾收集技術(SGC) ，所提出的循序垃圾收集技術(SGC)比起現有的設計平面化能更加平均化且有較低的設計複雜度，因此快閃記憶體的壽命可以隨之有效的延長；此外本論文提出之循序垃圾收集技術不需要額外調整觸發平面化技術的門檻。本論文所提出的循序垃圾收集技術可以輕易地使用軟體的方式去實現或是使用硬體的方式去實現。

論文中將比較 Greedy 垃圾收集方法及靜態平面化演算法及循序垃圾收集技術這些種方法，並針對抹除次數中的最大值，標準差，平均抹除次數，以及抹除時的額外開銷這些數據來比較。

最後為了證明循序垃圾收集技術是可以輕易地在軟體及硬體環境上實現，使用ＦＰＧＡ開發板來驗證方法是否可以在硬體上實現。

Abstract

In this thesis, we discuss the techniques of wear leveling in the flash memory system and also rebuild algorithm of Greedy garbage collection and static wear leveling for comparison. Thus a low-complexity high-performance wear leveling is proposed and it can extend lifetime of the flash memory and balance the utilization of each block.

This thesis analyzes the advantages and disadvantages in current wear leveling techniques. Besides, we introduce the important issues to maintain read/program performance and the lifetime in the flash memory. In addition, we will introduce the current wear leveling methods, and discuss the complexity of these approaches.

The proposed wear leveling algorithm is called sequential garbage collection (SGC), SGC outperforms existing designs in terms of wear evenness and low design complexity. The lifetime of the entire flash memory can be greatly lengthened by the proposed SGC. In addition, the proposed SGC doesn't require any tuning threshold parameter. The low-complexity low-cost SGC makes it easy to be implemented by firmware-based or hardware-based approaches.

This thesis compares Greedy garbage collection, static wear leveling and sequential garbage collection (SGC) in terms of maximum block erase count, the standard deviation of the block erase count, average erase count and overhead of wear leveling.

Finally, to prove the sequential garbage collection (SGC) is easy to be implemented by firmware-based or hardware-based approaches. We use the FPGA board to implement the propose algorithm and verify the performance of the proposed algorithm.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Flash memory overview

In recent years, flash memory becomes an important storage system for embedded systems, just like smart phones, PDAs, note books. Flash memory has many advantages, such as low power consumption, high performance in read operation, and better shock resistance than the conventional designs. However it also has some disadvantages need to be improve like price, performance of write operation, and the most important issue is the endurance of flash memory.

Flash memory can be divided into NOR flash memory and NAND flash memory in the market [1] [2], and it becomes the most popular storage device in non-volatile portable electronic device. NOR flash memory can execute-in-place (XIP) (random access) because it can access each byte. Therefore, NOR flash memory has fast read access time especially in small random data access (ex: 1MB-4MB). In the other hand, the NAND flash memory erase time is faster than NOR flash memory, because the size of erase blocks of NOR flash memory is larger than NAND flash memory. It means NOR flash memory has longer execution time in the program/erase access. However NOR flash memory also has disadvantages of poor chip density and high cost per bit. NOR flash memory is usually used for code storage which often execution in place such as simple home appliances, low-end mobile handsets and embedded designs. On the other hand, NAND flash memory has been developed to become an high chip density data storage and it reads/writes for I/O interference with page, so the write access time is shorter than NOR flash memory, but the read access time is longer than NOR flash memory. After the design trade-off, it was give up execute in place (XIP) to become high density and small cell size storage devices. Table 1.1 Shows the comparison of NAND and NOR flash memory.

Table 1.1: The comparison table of NAND flash memory and NOR flash memory; this table is captured from the Website of http://www.elnec.com/sw/an_elnec_nand_flash.pdf

	NAND	NOR
Capacity[*1]	~32Gbit	~1Gbit
Access method	Sequential	Random
Performance	Fast read (serial access cycle) Fast write Fast erase (approx. 2ms/block)[*2]	Fast read (random access) Slow write Slow erase (approx. 1s/block)[*3]
Life span	100,000-1,000,000	10,000-100,000
Price	Low	High

*1-By NAND flash manufactures materials(available at 01/2006)

*2-Toashiba TH85NVG1S3A(1 block is 16 kB)

*3-Intel StrataFlashP30family(1 block is 128 kB)

Many applications choose NAND flash memory to develope their design instead of NOR flash memory [3] [4]. Because capacity and price are very important issues for smart phones, solid-state disks (SSDs) and MP3 players.

NAND flash memory has two major types: single-level cell (SLC) and multi-level-cell (MLC) [5] [6] [7]. SLC means per cell stores one bit information, it has two states: 0(program) and 1(erase). SLC has higher speed and higher reliability than MLC. On the other hand, MLC can contain more than one bit information so it has large capacity. For example, 2-bit MLC has 4 states: 00, 01, 10 and 11 to store data. Although MLC has lower speed and worse reliability, it has lower price with higher capacity than SLC. Different applications will choose a suitable type NAND flash memory for the system. Recently on the market TLC is developed, TLC means triple level cell (3-bit cell) .Table 1.2-1 is the comparison for different types of NAND flash memories.

Table 1.2-2 is the comparison table of price and density. Different types of NAND flash have different price, SLC has higher price and lower chip density although it has higher endurance cycle. Oppositely, TLC has very low price and highest chip density as compared with SLC. Different applications require different types of NAND flash memory. If TLC is chosen a very low endurance cycle of TLC requires a good policy to handle the erase operation.

Table 1.2-1: The comparison table of SLC and MLC; this table is captured from the Website of http://www.psism.com/SLC%20vs%20MLC.pdf

	SLC	MLC
Voltage	3.3V/1.8V	3.3V
Technology/Chip Size	0.12um	0.16um
Page Size/Block Size	2KB/128KB	512B/32KB or 2KB/256KB
Access time (Max.)	25us	70us
Page Program Time(Typ.)	250us	1.2ms
Partial Program	Yes	No
Endurance	100k	10k
Write Data Rate	8MB/s+	1.5MB/s

Table 1.2-2: The comparison table of price and density; this table is captured from the Website of http://www.cnyes.com/fc/metal/fc_flash.asp

	SLC	MLC	TLC
Density	32GB	32GB	32GB
Price	44.3(US)	2.29(US)	1.93(US)
endurance	100K	5-10K	100-500

The functional block diagram for the Samsung K9GAG08X0M flash chip is shown in Fig. 1.1 [8]. The NAND flash memory architecture of 2G*8bits flash chip contains 8-bit I/O port to translate commands, data and addresses. If we want to read/write data, it needs 5 cycles to translate the address data (2 cycles for the column address and 3 cycles for the row address). A flash memory contains many blocks, and each block is consisted of a fixed number of pages. In the Samsung 2G MLC chip, each page is 4K bytes. A page is divided into data area and spare area, and data are stored in data area and meta data of ECC are stored in the spare area. The data structure of flash memory is shown in Fig. 1.2.

There are some special characteristics in the flash memory such as the unit of read/program is a page and the unit of erase is a block and the speed in different operations has great difference. The most important technique of the flash memory is that it cannot be overwritten; it must be erased before new data can be stored.

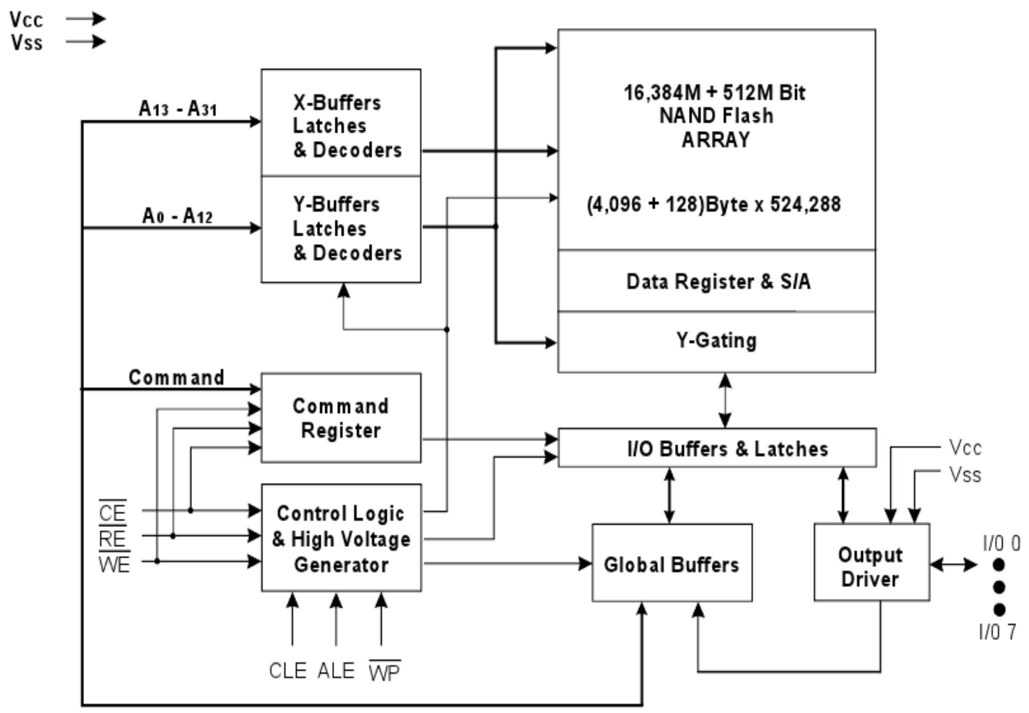

Vcc
Vss
A13 - A31
A0 - A12
X-Buffers Latches & Decoders
Y-Buffers Latches & Decoders
16,384M + 512M Bit NAND Flash ARRAY
(4,096 + 128)Byte x 524,288
Data Register & S/A
Y-Gating
Command
Command Register
I/O Buffers & Latches
Vcc
Vss
CE
RE
WE
Control Logic & High Voltage Generator
Global Buffers
Output Driver
I/0 0
I/0 7
CLE ALE WP

Fig. 1.1: The architecture of the flash memory.

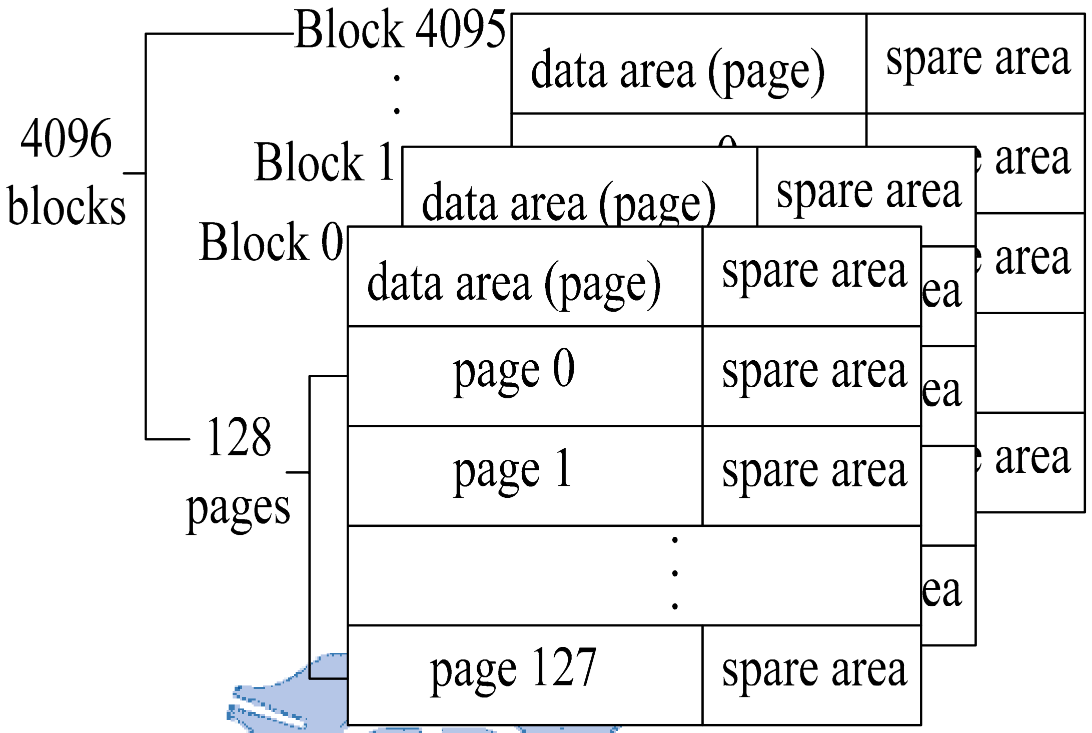


Fig. 1.2: The data structure of flash memory

1.2 NAND flash memory system overview

Fig.1.3 shows the NAND flash memory system architecture. Firstly, logical page addresses are translated into physical page addresses by the flash translation layer (FTL) [8] [9] [10]. Then data are programmed into the physical address of the flash memory. When free space is not enough, garbage collection (GC) will start to collect invalid data and release free space. When some blocks of the flash memory are used more often, these blocks of the flash memory will be worn out more often and causes the flash memory retired earlier. Thus, wear leveling is required to lengthen the life time of the flash memory [11] [12]. A flash memory needs to manage which block cannot be used, and this is called bad block management [13]. In a simple flash memory system, we need a flash translation layer (FTL), a garbage collection (GC), a wear leveling and a bad block management. These techniques can let user be easy to use the flash memory and maintain the performance of the flash memory.

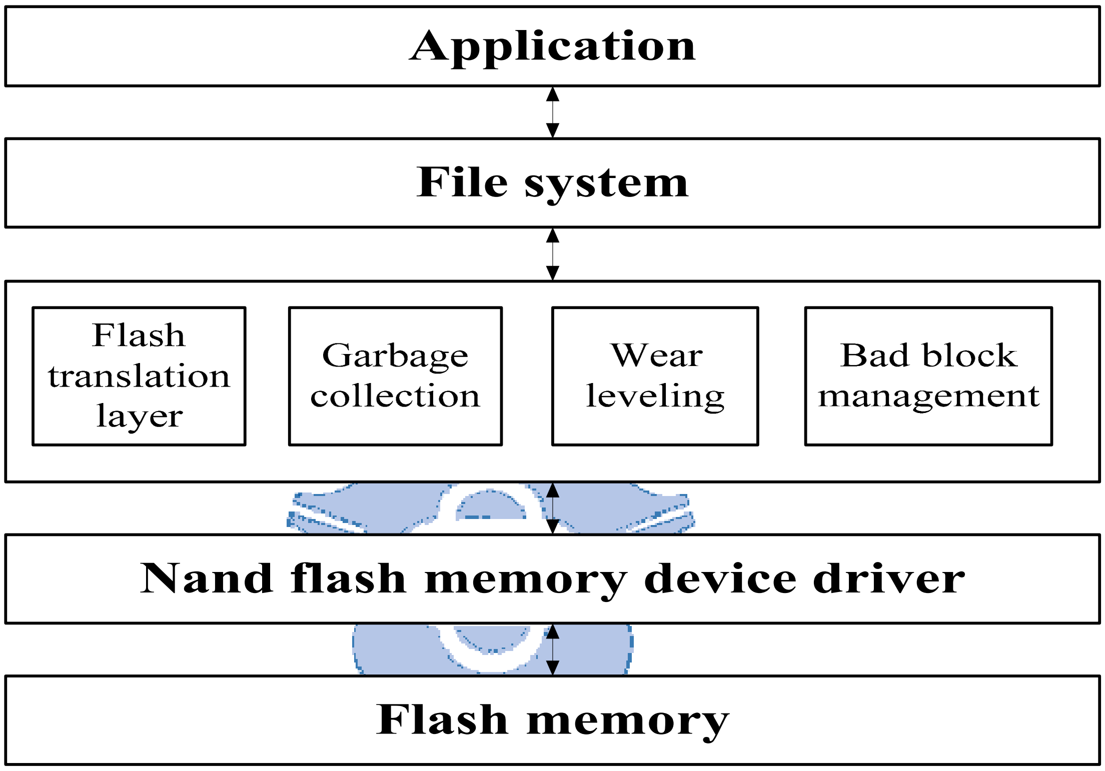


Fig.1.3: The flash memory system architecture.

1.2.1 Flash translation layer

NAND flash memory has many design bottlenecks. First, it cannot be overwritten, it must be erased before new data can be stored in the same physical address. However, erase operation usually takes several million-seconds. For performance consideration, updates to data are usually handled in an out of place fashion, rather than directly overwriting the old data. So data should be written to another page of the flash memory, and the original page of the data is marked as an invalid page. To solve this problem, flash translation layer (FTL) has been proposed. The FTL translates the logical page address (LPA) into physical page address (PPA) of the flash memory chip. Many flash translation layer mechanisms have been proposed such as the page level mapping (FTL), the block level mapping (NFTL), and the hybrid mapping. Fig. 1.4 shows a example operation of the page-level FTL, there have 5 write request (5,2,0,5,2) and LPAs address (5,2) have be rewritten. In each action of FTL, LPA will be mapped in a new PPA. In addition, the original PPA will be marked as invalid data. When data need to be rewritten, FTL will find free PPAs to map instead of erasing the current physical blocks. By using FTL, we can avoid the frequently erase the blocks. On the other hand, the performance can be improved and the free spaces can be best allocated with FTL in the flash memory. Although FTL has many advantages, but to record all the mapping table needs a large memory space. According to above, NFTL [9] and hybrid-FTL [25] approaches had been proposed. NFTL approach is proposed to solve the problem of memory space for a mapping table, so it uses a block level mapping table. Therefore, NFTL can decrease required memory space of the mapping table, but the flash memory space utilization is not better than a page-level FTL. Finally hybrid-FTL [25] is proposed which uses both page level and block level mapping.

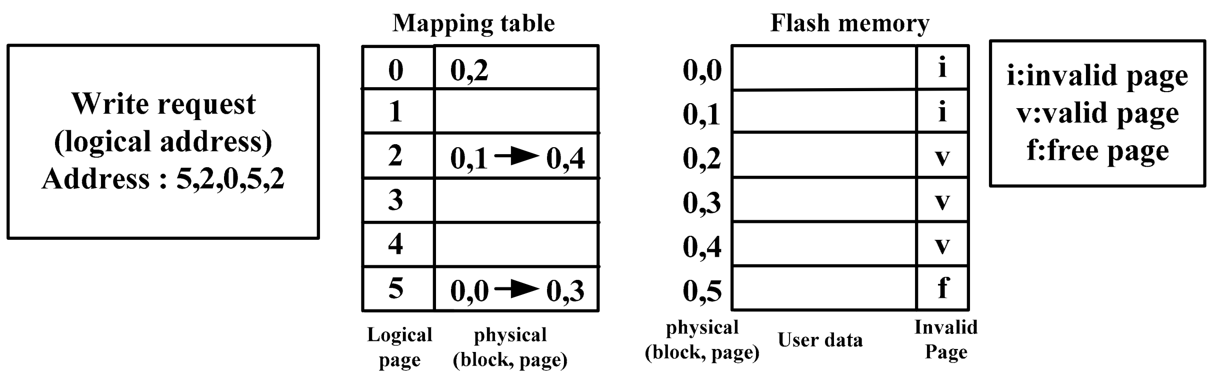


Fig.1.4: page level flash translation layer.

1.2.2 Garbage collection

When data are written into the flash memory, the amount of free pages is reduced, and thus the garbage collection (GC) is performed to recycle the spaces occupied by invalid pages. The GC operation collects valid pages in the selected block, copies valid pages to an another free block, and then it erases the selected block to recycle free pages. Many approaches have be proposed, Greedy algorithm [10] is the most effective algorithm to recycle blocks of the flash memory. The Greedy algorithm find the block which has maximum invalid pages then erases it. Therefore Greedy algorithm has minimum overhead in each GC operation. Fig.1.5 shows the operation of GC.

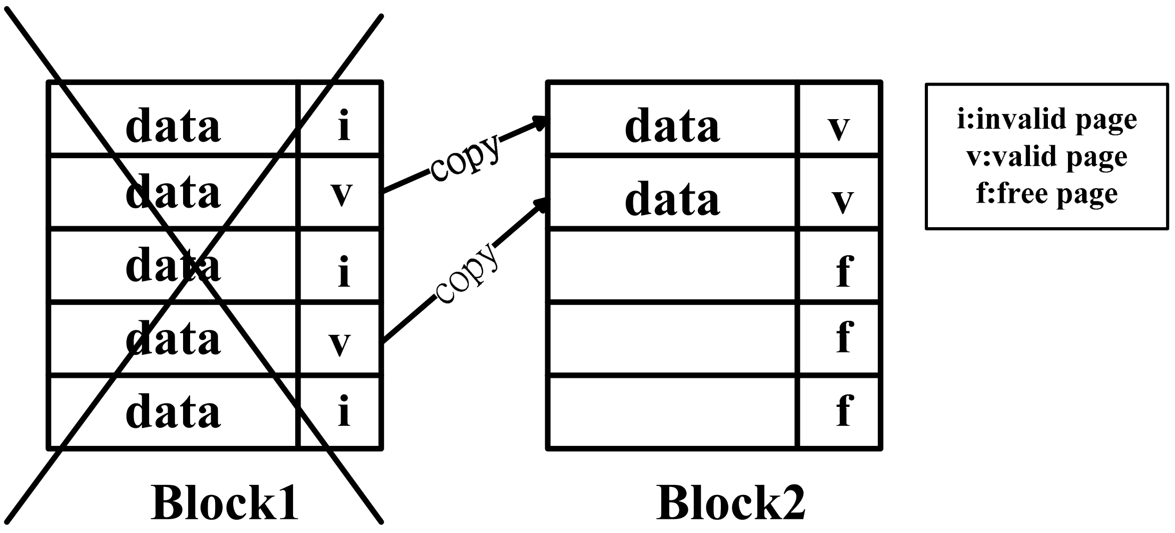


Fig.1.5: The garbage collection operation.

1.2.3 Wear leveling

The bottleneck in a NAND flash memory is the endurance. Blocks of the NAND flash memory have a limited erase cycle, the limitation of the MLC NAND flash memory is about 10,000 times, and the SLC NAND flash memory is about 100,000 times. In addition, GC operation with Greedy algorithm may wear out some blocks of flash memory chip due to the localities of data access. For example, if some blocks stored hot data, they will be invoked by GC operation more frequently and these blocks will have high erase cycle times. These blocks may be retired earlier, and this causes the reliability problem of the flash memory. To solve this problem, many wear leveling techniques have been proposed [14]-[24]. Wear leveling balances the erase cycle between different blocks in the flash memory, and thus the life time of the flash memory can be extended.

1.2.4 Bad block management

Flash memory contains early bad blocks and latter bad blocks [13]. When flash memory is fabricated, some bad blocks exist, it is called early bad blocks. Generally, the early bad blocks are about 1% of the total capacity of the flash memory. Due to the erase operation, the flash memory produces latter bad blocks. Generally, flash memory can tolerate early bad blocks and latter bad blocks within 1% and 5%, respectively.

To avoid data program/read to a bad block, it needs a policy to manage bad blocks, and then the reliability of the flash memory can be further improved. When flash memory starts up, bad block management will check each block to rebuild a bad block table. Accordingly, bad block management will keep updating bad block table if any blocks are broke.

1.3 Thesis Organization

In this thesis, we discuss about the file system contains a flash translation layer (FTL), a garbage collection (GC) and a wear leveling. We propose a low-complexity high-performance wear-leveling algorithm which named sequential garbage collection (SGC) for flash memory system design. SGC algorithm can improve the lifetime of the flash memory. The rest of the thesis is organized as follows.

In Chapter 2, we discuss about wear leveling type and methods. Two major types of wear leveling: dynamic and static are discussed. We introduce wear leveling algorithms which have been proposed in recent years.

In Chapter 3, the design for sequential garbage collection has been proposed. We propose a low-complexity high-performance wear-leveling algorithm which named sequential garbage collection (SGC) for flash memory system. For verification, we also build a simple model of the flash memory file system. This model contains: FTL (page level mapping), GC operation and wear leveling. We rebuild algorithms of Greedy garbage collection (GC) and static wear leveling (SW) to compare with the proposed SGC algorithm. Chapter 3 also shows the experimental results of these algorithms, besides we summarize the experimental results in term of erase count, overhead and lifetime for each algorithm.

In Chapter 4, we use FPGA to implement the methods introduced in Chapter 3. It can prove the proposed SGC algorithm is simple and easy to be implemented by hardware design. In addition, we also compares about the resource cost with other methods.

In Chapter 5, we make a conclusion of this thesis, and discuss about the further work.

Chapter 2

Related work of wear leveling

2.1 Dynamic wear leveling and static wear leveling

In recent years, chip density of flash memory is growing up but the endurance of flash memory is decline. Erase cycle of each block is about 10k times. Flash memory will retire very fast if we do nothing.

The data stored in the flash memory can be divided into static data and dynamic data. Static data contains operating system files and user files. Dynamic data contains log files. Similarly, wear leveling also has two major types: static wear leveling [23] and dynamic wear leveling [14]. We compare of no wear leveling, dynamic wear leveling and static wear leveling in this Chapter.

For example, we program a 4KB file into 2MB flash memory repeatedly. The flash memory contains 100 blocks and each block is 2KB. Therefore, it means two logical addresses are writing repeatedly. In addition, if we assume the endurance of each block is 10000 times, and the request time including erase and program a block is about 0.10 second with the Samsung 2GB NAND flash memory.

In Eq. 2.1, we calculate a block erase and program times with the Samsung 2GB NAND flash memory. In the datasheet of Samsung 2GB NAND flash memory, each block erase time is 1.5ms and each page program time is 800μs.

A block erase and program times = 1.5 ms + (800μs*128(page number of per block))

= 1.5 ms + 102.4ms = 0.1039 second (2.1)

In Eq. 2.2, if we do not perform wear leveling and two physical blocks are erased and programmed repeatedly. When we program a 4KB data into flash memory repeatedly, the flash memory will break out in 34 minutes. This means that wear leveling is a necessary technique to extend the life time of flash memory.

Life time of no wear leveling= 10,000 * 2(block) * 0.1039 (second time per block)

= 2078 seconds=34 minutes (2.2)

Dynamic wear leveling only moves the dynamic data in the flash memory. In Eq. 2.3, it calculates the life time with dynamic wear leveling, if the dynamic data occupied 20% of the flash memory, and then only 20 blocks are involved in wear-leveling. In this testing, we have 100 blocks of the flash memory, so dynamic data occupies 20 blocks.

Life time of dynamic wear leveling

= 10,000 * 20(block) * 0.1039 (second time per block)

= 20780 seconds=5.66 hours (2.3)

Static wear leveling will move static data in the flash memory and thus all blocks are involved in the wear-leveling. Eq. 2.3 shows the life time with static wear leveling.

Life time of static wear leveling

= 10,000 * 100(block) * 0.1039 (second time per block)

= 103900 seconds=28.86 hours (2.4)

2.2 Survey of wear leveling algorithm

There are many wear-leveling algorithms have be proposed in prior studies to solve the problem of GC operation and extend the lifetime of the flash memory. The purpose of wear-leveling algorithm is evenly distributing block erases over the flash memory, and then the endurance of the flash memory can be enhanced. The performance of dynamic wear-leveling depends on hot data identification, and the cold data always stay at their blocks regardless of whether updating of hot data wear out the other blocks. As a result, the endurance improvement is strongly dependent on how many blocks are already occupied by cold data.

As follows, we define the type of data and blocks. Data can be divided into two types: hot data mean the same data has been updated more frequently. Cold data mean data do not be written frequently. Block also has two types: hot blocks mean the block has been erased frequently. Cold blocks mean the block has been erased infrequently. If the standard deviation of the block erase count is very large, it means the block erase counts are not balanced. Oppositely, a small standard deviation means block erase counts of the flash memory are balanced.

Before wear leveling action the hot data are stored in maximum erase count block (hot block), the block will updated more frequently. So this block will be erased more frequently, and makes some block have a very high erase count.

2.2.1 Hot-cold swapping

In Hold-cold swapping [14], when the difference between the maximum block erase count and minimum block erase count is larger than the specified threshold. Wear leveling action will swap the data that have been stored in the maximum erase count block and minimum erase count block. The hot data which stored in the hot blocks are moved to cold blocks, therefore the cold block will be erased if next erase operation executes. That can balance erase count of blocks in the flash memory.

Although this algorithm can balance erase count in the flash memory, however it needs a threshold value to control whether the wear leveling will be executed. If threshold is too large, flash memory will execute wear leveling infrequently, thus it will cause poor performance of wear leveling. Besides this method will require high overhead when data are swapped. The most important problem of this method is that, we cannot ensure the cold data will be stored into hot block after wear leveling. Unfortunately, hot block will be erased immediately if hot block is stored hot data after swap. At this situation, hot-cold swapping algorithm cannot effectively balance erase count in the flash memory.

2.2.2 Static-dynamic wear leveling

Static-dynamic wear leveling contains dynamic wear leveling [15], static wear leveling and file allocation table (FAT) filter. In dynamic wear leveling: free physical block is combined in a wear leveling pool. When new data are programed, block will be allocated with a physical block address by round-robin policy. If there are not enough free physical blocks in wear leveling pool, dynamic wear leveling will copy valid data to the last block in pool. In static wear leveling, when there have static files to be stored, this method will ignore dynamic wear leveling and then uses the static wear leveling. It will use hot cold swapping to achieve static wear leveling. In addition, it also implements a FTA filter to monitor block to improve wear leveling.

Data need to be divided into static data and dynamic data in this method, but it cannot correctly analyze data belong to which type. Therefore, it requires more information to analyze the data and causes the high design complexity.

2.2.3 Two level of wear leveling

Two-level wear leveling is implemented in the FTL [16]. In first leveling: free physical blocks will gather in wear leveling pool and string with a chain. When new data come to be programed, it will be allocated to the minimum erase count block (cold block). If wear leveling pool does not have enough physical blocks, wear leveling operation will reduce valid data in the minimum erase count block. The chain of physical blocks is stringing from the minimum erase count block (cold block) to the maximum erase count block (hot block). In second leveling: if the difference between maximum erase count and minimum erase count is larger than the specified threshold, the static data will be moved to the free block. Then the cold blocks can be restored for new data.

This method maintains average utilization rate for blocks in flash memory. However, the specified threshold needs to be tuned in different applications. In addition, two level algorithm needs to management the chain in the additional system memory.

2.2.4 Erase pool

Erase pool algorithm composites an erase pool with free physical blocks [17]. When same logical address data need to be programed, it will be mapped to free blocks in the erase pool. If block has been reached erase endurance limitation, it will be moved to the outside block of the erase pool. Then it copies valid data to another free block. Although the space will be reduced in erase pool, the capacity of the flash memory is not affected. When the blocks of erase pool are exhausted, it means flash memory reaches the end of life.

This method can detect the block which reaches the block erase endurance limitation and then stops to use it. This method would extend the lifetime of flash memory but worse the flash memory utilization.

2.2.5 Old block protection

Old block protection wear leveling is applied to protect the block which reaches maximum erase count [18]. When the just-erased block is also the maximum erase count block and the erase count is difference with the minimum erase count block is larger than the specified threshold. The valid data in minimum erase count block will be copied to just-erased block. When just-erased block also has the maximum erase count block, it means that the hot data is stored in this block. Then this block will be joined erase more quickly. Therefore we move the valid data in the minimum erase count block to the just-erased block.

This method would move the cold data to the hot block, it can avoid just-erased block being erased immediately. In addition, it can balance erase count of block in the flash memory. However it also needs a threshold to maintain the wear leveling performance.

2.2.6 Dual-pool

Dual pool algorithm is used to implement two-pool wear leveling [19]. Pools can be divided in to the cold pool and hot pool; Q_{HP}^{EC} , Q_{CP}^{EC} express two queues is strung with hot pool and cold pool, $H^{+}Q$) and $H^{-}(Q)$ are expressed maximum queue-head and minimum queue-head. When the erase count difference between EC($H^{+}(Q_{HP}^{EC})$) and EC($H^{+}(Q_{CP}^{EC})$) are larger than the specific threshold. In the first step, valid data in $H^{+}(Q_{HP}^{EC})$ move to other block that has free pages and erase block $H^{+}(Q_{HP}^{EC})$. Second, valid data in $H^{-}(Q_{CP}^{EC})$ move to block $H^{+}(Q_{HP}^{EC})$ and erase block $H^{-}(Q_{CP}^{EC})$. Then $H^{-}(Q_{CP}^{EC})$ and $H^{+}(Q_{HP}^{EC})$ will be swapped to another pool. The most important thing in this method is classified the blocks into the cold pool or the hot pool. It can ensure which block has involved wear leveling and will not be erased repeatly at short time.

This method is similar to hot-cold swapping method. Dual pool method can manage block and data are more rigorous than hot-cold swapping method. Dual pool adjusts blocks belong to which pool in every write request. For this reason, it can maintain a small standard deviation of the block erase counts, but it needs many memory spaces to record information required for the algorithm.

2.2.7 Conditional threshold wear leveling

Conditional threshold wear leveling is used to maintain multi-channel flash memory [20]. When erase operation is trigged and the erase count difference between just-erased block and minimum erase count block is larger than the specific threshold, Conditional threshold wear leveling will copy the valid data in minimum erase count block to just-erased erase block. To reduce runtime of wear leveling, it uses the blocks in different channel that can read and erase at the same time. So both just-erased block and minimum erase count block need to be in different channel. The threshold definition is also different with previous methods. It has two levels definition of threshold; the first level threshold will change with average erase count. In addition, the second level threshold is a parameter for wear leveling.

This method is used to multi-channel and multi-chip flash memory architecture to improve the erase time; because the management of blocks in multi-channel and multi-chip are very difficult and complex. The just-erased block doesn't guarantee to contain hot data, and thus this algorithm may perform unnecessary data movement. In addition, the second level threshold tuning will influence the performance of wear-leveling.

2.2.8 Efficient garbage collection policy

Efficient garbage collection policy divides data and blocks into three types [21]. This method using modification-aware (MODA) page allocation method to manage block and create three lists of hot data, worm data and cold data. The priority for garbage collection action is from the hot list and warm list to the cold list, and all blocks have opportunity to invoke garbage collection. Therefore, it can guarantee that all blocks have good performance of utilization in the flash memory.

This method has no policy for wear leveling, because the garbage collection action also maintains good performance of wear leveling. However, it needs to define four thresholds to control which block can join garbage collection policy. If we change the application for the flash memory, those thresholds need to be redefined, otherwise it cannot keep the same performance.

2.2.9 Static wear leveling

Static wear leveling is used a bit table to record which block has been erased [22] [23]. When there is difference between the number of total block erase count and the number of block which has been erased during affter reset is larger than the specified threshold, it will move the static data to the free block to balance the erase count in each block. Therefore, a block erasing table (BET) is created to identify which block has been erased after reset. Then, the static wear leveling will move valid data in the block that has not been erased so far. This algorithm prevent cold data from staying at any block for a long time, and then the maximum block erase count difference between any two blocks can be minimized.

This method can implement with hardware approach rather than software approach because of it is very simple. However, the wear leveling threshold for various system environments can be very different. Using inadequately tuned parameters can cause unexpectedly high wear leveling overhead and worse performance of wear-leveling.

2.2.10 Lazy wear leveling

Lazy wear leveling is used to analyze logical blocks and physical blocks [24]. The latest update of logical block is record. Elder block means that the erase count of this block is larger than the average count. Otherwise, it is a junior block. This algorithm is used to erase the logical block which not updates recently and copies valid data to the elder block. An automatically on-line threshold tuning algorithm is proposed in [24].

The proposed algorithm monitors dynamic disk workloads to adjust the duty-cycle of wear-leveling, and thus the wear-leveling overhead and wear evenness can be trade-off. However it requires much information to control the algorithm and it will increase the design complexity and it also requires large memory space to record information.

2.3 Summary

Action of wear leveling needs extra execution time and memory space. Extra time is consumed by coping cold data to hot block; Extra memory space is used to record the information of wear leveling. Although wear leveling takes overhead and design complexity, it can solve the early retire problem of the flash memory. So wear leveling needs to trade-off both the performance and overhead in flash memory system design.

We summarize wear leveling of existing methods as follows. Three conditions are defined: first is what time need to do wear leveling; second is which block needs to do wear leveling; third is where the data to place. These conditions can improve performance of wear leveling, and extend lifetime of flash memory. However it also increases cost of execution time and memory space.

Table 2.1 we compare the pervious algorithms. In table 2.1, threshold truing expresses this wear leveling needs a threshold. Triggering condition expresses what kind of situation in the flash memory needs to invoke wear leveling. Each wear leveling operation requires for addition block erase and page copy. Overhead of each wear leveling operation includes that in each wear leveling, the block needs to be erased and whether the block which needs to be copied valid data and move to another block. Each algorithm needs extra memory space to record information for wear leveling. However memory space is an expensive cost for the embed system, so reduction memory space is very important.

Table 2.1: The comparison table of wear leveling algorithms

method	Threshold tuning	Triggering condition	Overhead of each wear leveling operation	RAM space
Hot-cold swapping	Yes	Check the threshold between maximum erase count and minimum erase count	Block erase:2 Block copy:2	EC 4096*17(bit)
Static-dynamic	Yes	periodically	Block erase:2 Block copy:2	EC + free_pool 4096*17(bit)+4096
Two level	Yes	periodically	Block erase:2 Block copy:2	EC 4096*17(bit)
Erase pool	No	Block reach limited erase count	Block erase:1 Block copy:1	EC 4096*17(bit)
Old-block protection	Yes	On block erase	Block erase:1 Block copy:1	EC 4096*17(bit)
Dual pool	Yes	After each write	Block erase:2 Block copy:2	EC+EEC+hot_pool+cold_pool 4096*17*2(bit)+4096*2
Conditional threshold	Yes	After each erase	Block erase:1 Block copy:1	EC+EC_channel$_{avg}$ 4096*17(bit)+4*17
Efficient GC	Yes	After each write	Block erase:1 Block copy:1	EC+ hot_pool+cold_pool+warm_pool 4096*17(bit)+4096*3

Static wear leveling	Yes	After each erase	Block erase:1 Block copy:1	EC+BET 4096*17(bit)+ 4096
Lazy Wear leveling	No(online tuning)	After each erase	Block erase:1 Block copy:1	EC+ time_logical page 4096*17(bit)+ 4096*10

Chapter 3 Architecture and implementation

3.1 Design of wear leveling

In this thesis, a low-complexity high-performance wear-leveling algorithm which named sequential garbage collection (SGC) for flash memory system design is presented. The proposed algorithm combines GC operation and wear-leveling into a single process. In addition, the proposed SGC doesn’t require any tuning threshold parameter as compared as prior researches. Thus it can be applied to various systems without prior knowledge of the system environment. The low-complexity low-cost SGC algorithm makes it is easy to be implemented by firmware-based or hardware-based approaches.

In a flash memory system, the embedded micro-controller has very limited memory space, and the hardware and computing resources are constricted. Thus the proposed SGC algorithm requires a low memory space and demands limited hardware resources. The basic idea to combine GC operation and wear-leveling into a single process is very simple that the blocks should be erased in a sequential manner.

3.2 Sequential garbage collection Algorithm 1

Algorithm 1 shows the pseudo code of the proposed SGC1 algorithm. In SGC1 algorithm, free_block means the number of free blocks in the flash memory, and index is the index number of the block which erased last time. If the number of free blocks becomes one, the controller executes SGC1 algorithm and performs GC operation on the selected block. GC operation copies valid pages to the free block and erases the selected block.

The proposed SGC1 algorithm evenly distributes block erases over the entire flash memory. Thus it has greatest wear-leveling performance. However, the GC operation doesn't consider the number of invalid pages in the selected block, and therefore, the SGC1 algorithm may have many extra living pages copy in worse case conditions.

```
----------------------------------------------------------------------------------
Algorithm 1: SGC1 (Sequential Garbage Collection V1)
----------------------------------------------------------------------------------
Input: free_block, index
Output: null
  1: if free_block = 1  then
  2:      index ← (index+1)  MOD  BlockSize;
          /* BlockSize is number of blocks in the flash */
  3:      GarbageCollection(index);
          /*Request the garbage collection to erase the selected block and
            copy valid pages to the free block*/
  4: end
```

3.3 Sequential garbage collection Algorithm 2

To reduce extra living page copy overhead of the SGC1 algorithm, a bit vector which named invalid page flag (IPF) is added. The IPF records whether a block contains more than 75% invalid pages. For example, if IPF[*index*] is equal to 1, then it means the block with index number: *index* contains more than 75% invalid pages. Oppositely, if IPF[*index*] is equal to 0, then it means the block has less than 75% invalid pages. The IPF vector can be updated during the write operation, and each block requires only one-bit flag. The GC operation will erase the blocks with IPF=1 with a higher priority.

Algorithm 2 shows the pseudo code of the proposed SGC2 algorithm. In SGC2 algorithm, *fcnt* is the number of "1" in the IPF vector. If *fcnt* is not zero, there are blocks having more than 75% invalid pages. In addition, *seq* and *index* are the index numbers for searching for the target block for current GC operation. If the number of free blocks becomes one, the controller executes SGC2 algorithm and performs GC operation on the selected block. In SGC2 algorithm, if *fcnt* is zero, there has no block with a higher priority. Then the blocks are erased in a sequential manner (step 2-6). Otherwise, if *fcnt* is not zero, there are blocks having more than 75% invalid pages. The SGC2 algorithm searches for the block with IPF=1 and then performs GC operation on the selected block (step 8-11).

The SGC1 algorithm erases blocks in a sequential manner but it may have many extra living pages copy overhead. The SGC2 algorithm uses the IPF vector to reduce the overhead in the SGC1 algorithm, but it requires some clock cycles to search for high priority blocks. The proposed SGC1 and SGC2 algorithms evenly distributes block erases over the entire flash memory, and thus the lifetime of the entire flash memory can be greatly lengthened. The proposed SGC algorithm requires a low memory space and demands limited hardware resources, and thus it is easy to be implemented in the flash memory system.

```
---------------------------------------------------------------------------------------
Algorithm2 : SGC2 (Sequential Garbage Collection V2)
---------------------------------------------------------------------------------------
Input: free_block, fcnt, seq, index, and IPF
Output: null
  1: if  free_block = 1  then
  2:        if  fcnt = 0  then
  3:              seq ←(seq+1) MOD BlockSize;
                  /* BlockSize is number of blocks in the flash*/
  4:              index ←seq;
  5:              GarbageCollection(index);
                  /* Request the garbage collection to erase the selected
                     block and copy valid pages to the free block*/
  6:        end
  7:        else
  8:              while  IPF[index]=0  do
  9:                    index ←(index+1) MOD BlockSize;
 10:               end
                  /* IPF=1 means more than 75% invalid pages
                     in the block */
 11:               GarbageCollection(index);
                  /* Request the garbage collection to  erase the selected
                     block and copy valid pages to the free block*/
 12:                index ←(index+1) MOD BlockSize;
 13:        end
 14: end
```

3.4 Experimental results

3.4.1 Environment setup

Table 3.1: Setting for the simulation

Storage capacity	2GB MLC
Storage block number	4096
Page number per block	128
Per page size	4kb
Endurance	10k-5k
Page read time	60ms (2,400 cycles)
Page write time	800ms (32,000 cycles)
Block erase time	1.5ms (60,000 cycle)

We build the cycle-accurate simulation environment of a 2GB flash memory to verify the effectiveness of the proposed SGC algorithm. The simulation parameters are listed in Table 3.1. As shown in Table 3.1, the erase operation takes much longer clock cycles than the other operation.

For comparisons with existing wear-leveling algorithms, we rebuild the GC operation with greedy algorithm and named as **GA**. In GA, GC operation will select blocks that have the highest number of invalid pages, and thus GA doesn't consider the wear-leveling problem. We also rebuild the static wear-leveling algorithms [22] [23] (with threshold=10 and threshold=100) and named as **SW**. In the simulation, totally 120GB data are written to verify the performance of wear-leveling algorithms. The example for these wear leveling algorithms are shown in Fig.3.1 to Fig.3.4 .

Fig.3.1 is Greedy garbage collection, In_page means invalid page number in each block and EC means count of the erase block. The Greedy garbage collection will erase the block which has maximum number with In_page and copies the valid data to free block. If hot data write into the same block, it will erase this block more frequently and cause larger erase count.

Fig.3.2 is static wear leveling, ecnt means the total number of block erases performed since the BET was reset, fcnt means the number of 1s in the BET, and BET means block erase table (the 0 in the table means the block has not be erased after reset) and EC means erase count of the block. The static wear leveling will erase a block that has not been erased so far.

Fig.3.3 is the proposed SGC1 algorithm. SGC1 evenly distributes block erases over the entire flash memory.

Fig.3.4 is the proposed SGC2 algorithm, fcnt means the number of 1s in the IPF, and IPF means 75% invalid page flag (1: a block contains more than 75% invalid pages). SGC2 will search for the block with IPF=1 and then performs GC operation on the selected block, if there are no block with IPF=1, we will recycle the block in sequential.

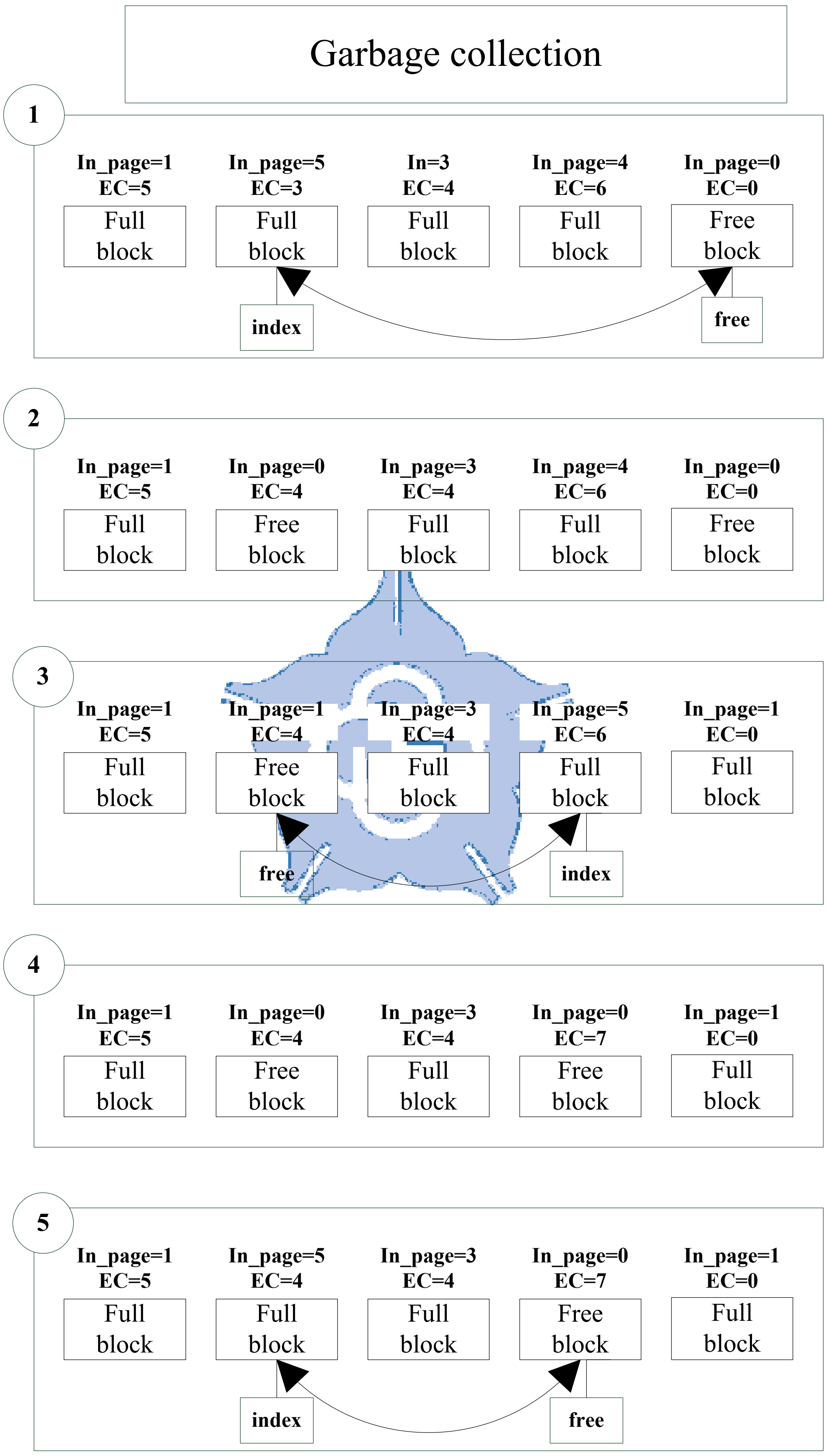

Garbage collection
1
In_page=1 EC=5 Full block
In_page=5 EC=3 Full block
In=3 EC=4 Full block
In_page=4 EC=6 Full block
In_page=0 EC=0 Free block
index
free
2
In_page=1 EC=5 Full block
In_page=0 EC=4 Free block
In_page=3 EC=4 Full block
In_page=4 EC=6 Full block
In_page=0 EC=0 Free block
3
In_page=1 EC=5 Full block
In_page=1 EC=4 Free block
In_page=3 EC=4 Full block
In_page=5 EC=6 Full block
In_page=1 EC=0 Full block
free
index
4
In_page=1 EC=5 Full block
In_page=0 EC=4 Free block
In_page=3 EC=4 Full block
In_page=0 EC=7 Free block
In_page=1 EC=0 Full block
5
In_page=1 EC=5 Full block
In_page=5 EC=4 Full block
In_page=3 EC=4 Full block
In_page=0 EC=7 Free block
In_page=1 EC=0 Full block
index
free

Fig.3.1: Greedy garbage collection

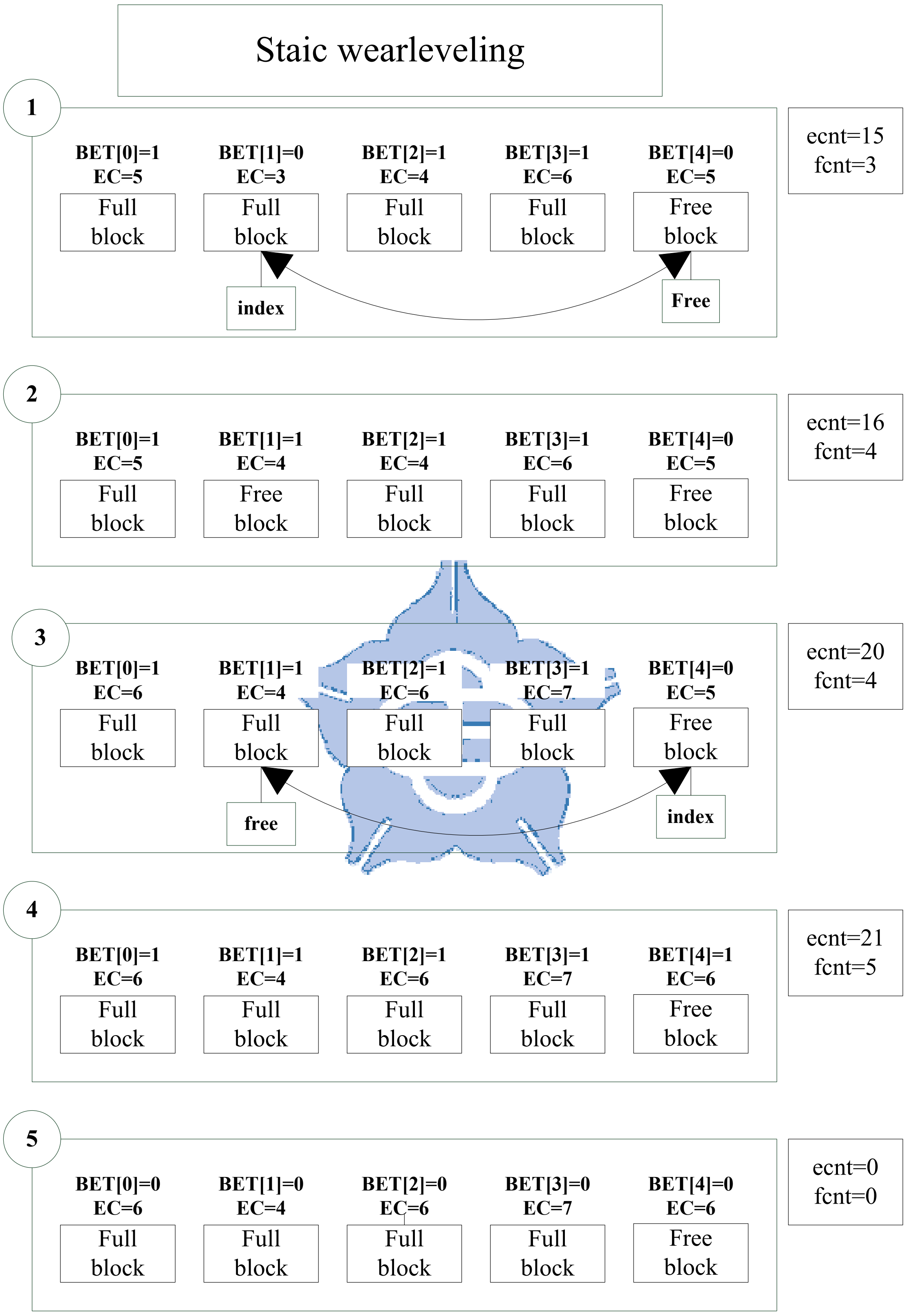


Fig.3.2: Static wear leveling

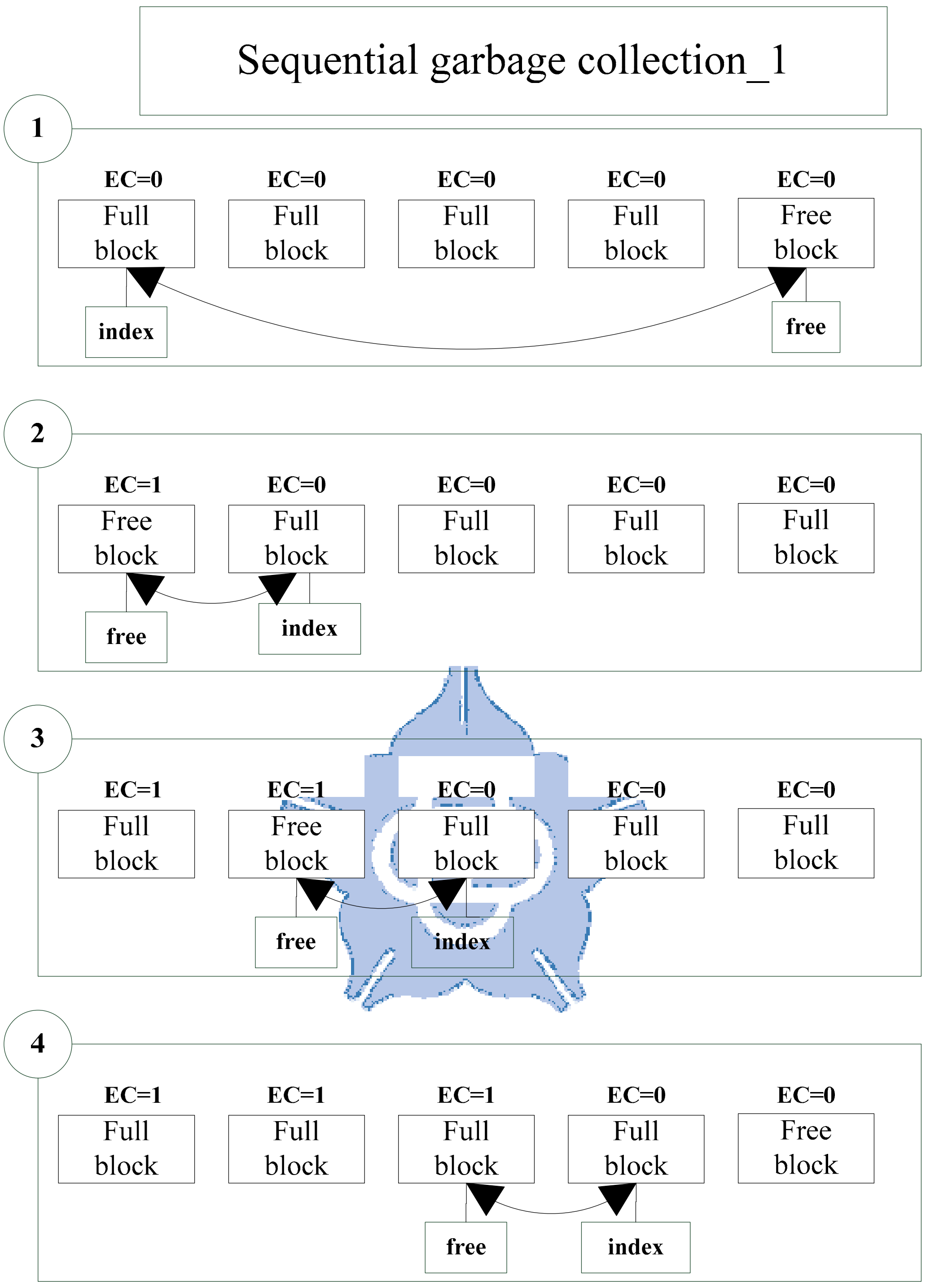


Fig.3.3: Sequential garbage collection_1

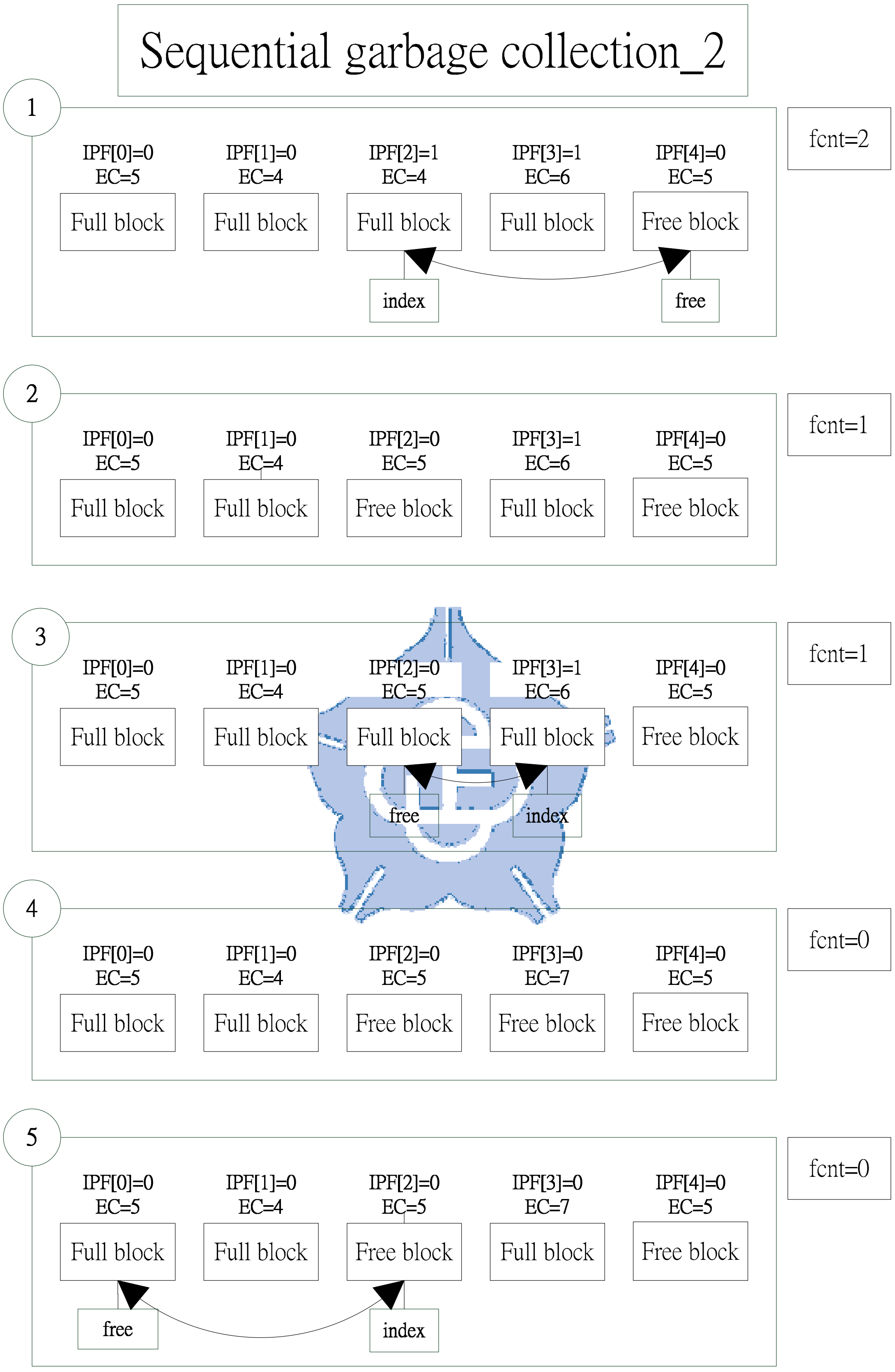


Fig.3.4: Sequential garbage collection_2

3.4.2 Lifetime of flash memory

Lifetime is an important issue of the flash memory due to the endurance of the flash memory. To measure flash memory lifetime, we use max erase count to show the flash memory lifetime, and a larger maximum erase count means the lifetime of the flash memory is shorter. The maximum block erase count after writing 120GB data is shown in Fig.3.5. In GA, greedy algorithm performs GC operation on the blocks with highest number of invalid pages. Therefore it may often erase on the same block which stored the hot data. The maximum block erase count for GA is 632 in this simulation. In SW, when the difference between the maximum block erase count and minimum block erase count is larger than the specified threshold, SW moves the static data to the hot block to balance the erase count in each block. The maximum block erase count for SW is 210 with threshold value: 10 in this simulation and the maximum block erase count is reduced as compared to GA. The maximum block erase count for SW is 582 with threshold value: 100, and the maximum block erase count cannot be effective reduced because using a higher threshold will worsen the performance of wear-leveling.

The proposed SGC1 algorithm always erases blocks in a sequential manner, and therefore it can achieve the lowest maximum block erase count which is 139 in this simulation. However it may have many extra living pages copy overhead, and thus, the proposed SGC2 algorithm uses the IPF vector to reduce the overhead in the SGC1 algorithm. The maximum block erase count of SGC2 is 157, and it is a little larger than SGC1. The proposed SGC algorithm can effectively reduce the maximum block erase count, and thus the lifetime of the flash memory can be greatly lengthened.

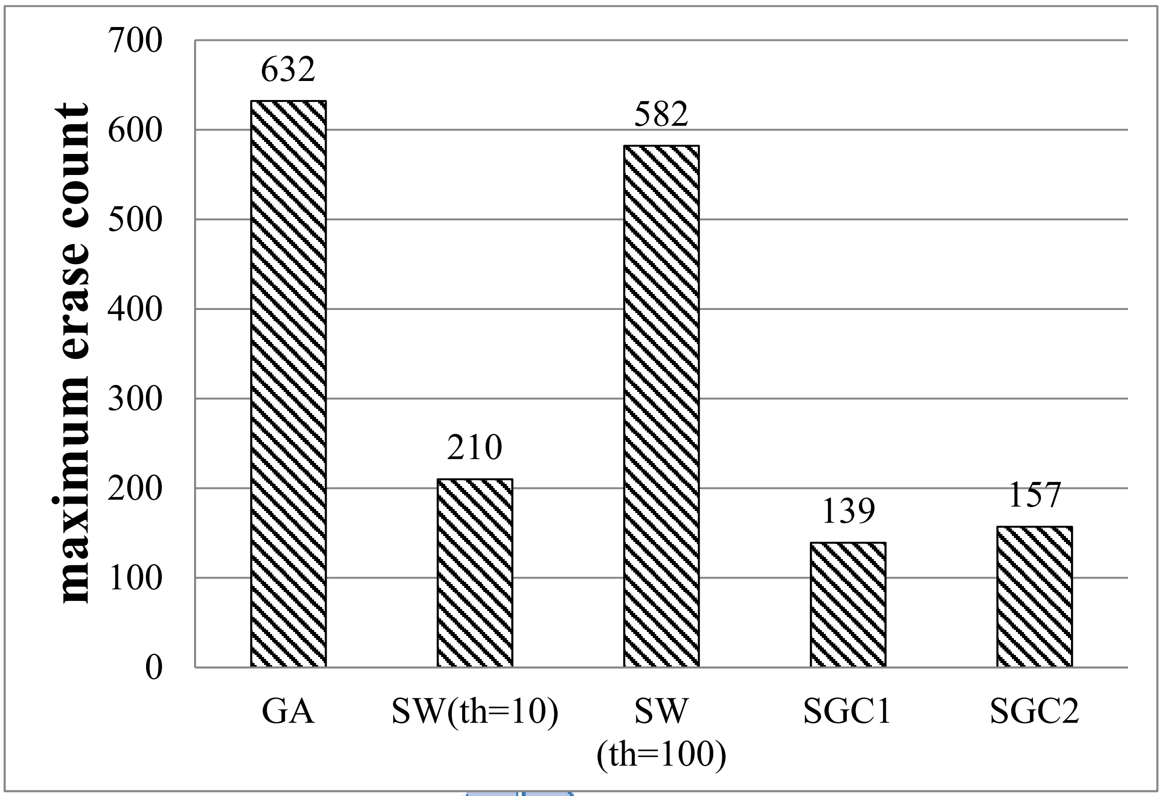


Fig.3.5: maximum block erase count.

3.4.3 Average Block Erase Counts

Fig.3.6 shows the normalized average block erase count for four algorithms. SW still uses the greedy algorithm to perform GC operation, but it requires extra GC operations to balance the erase count in the blocks, thus the average erase count is larger than GA. In SGC1, the efficiency in recycling the space occupied by invalid pages is not as good as greedy algorithm, thus it causes many extra erase operations. Therefore, SGC2 uses the IPF vector to reduce the overhead in SGC1, and then the average erase count is greatly reduced.

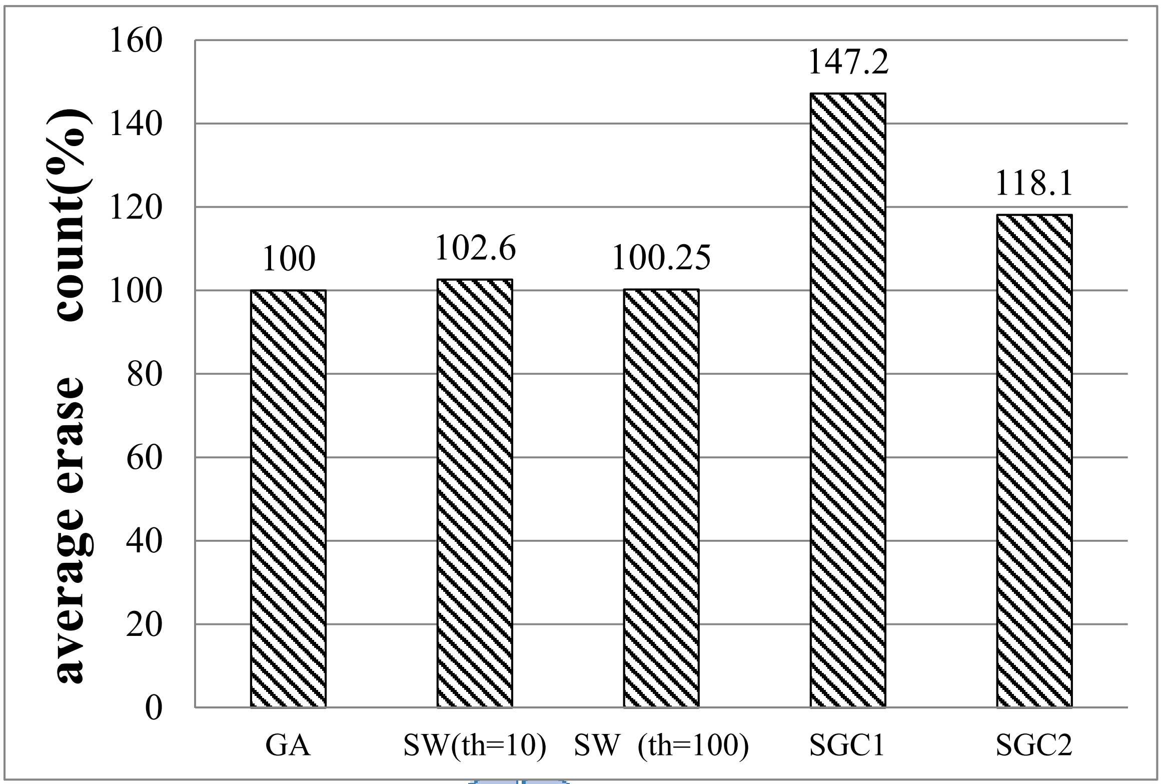


Fig.3.6: Normalized average erase count.

3.4.4 Distribution of Block Erase Counts

Fig. 3.7 shows the standard deviation of block erase count in four algorithms. The SGC1 algorithm always erases blocks in a sequential manner, and thus it can achieve a smallest standard deviation value. SGC2 uses the IPF vector to reduce the overhead in SGC1, and it also achieves a very small standard deviation as compared to SW. Fig.3.8 shows growing up of the standard deviation versus the amount of written data. As we can expect, the standard deviation in SGC1 will not have any changes, and SGC2 also achieves a very small standard deviation. However, in SW, the standard deviation is growing up when the amount of written data is increased.

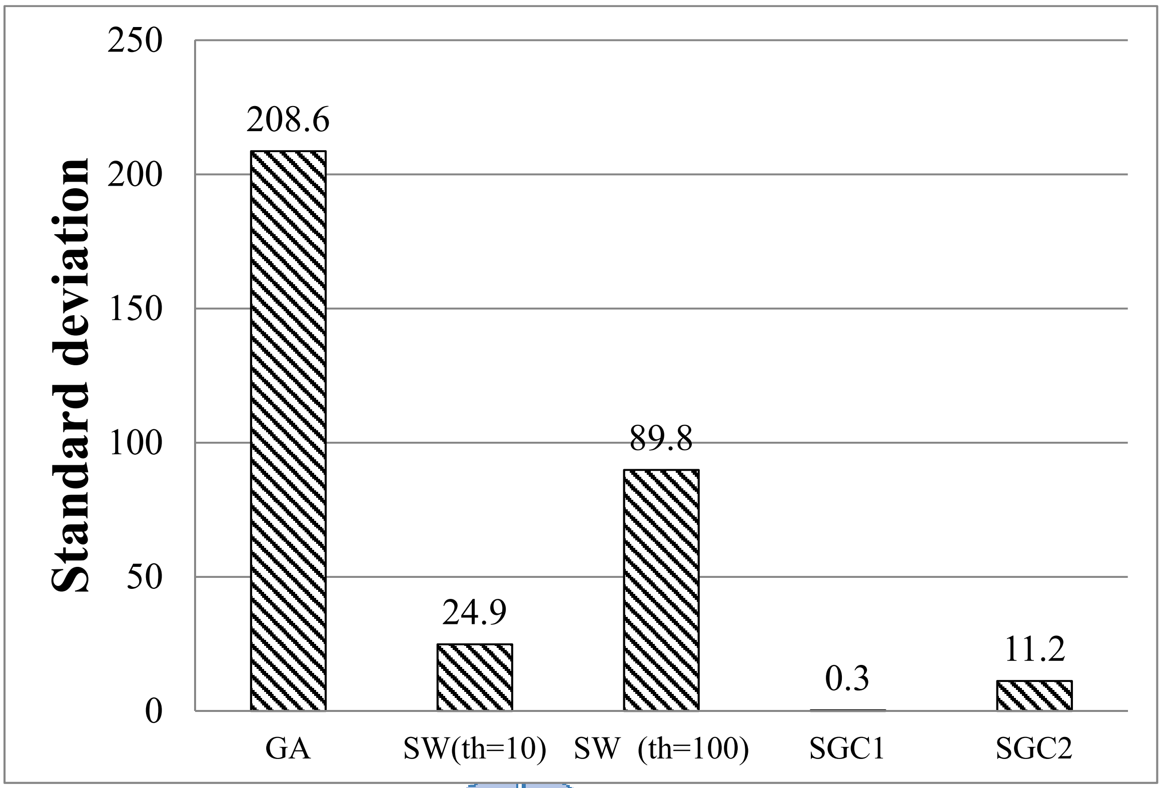


Fig.3.7: Standard deviations of block erase count

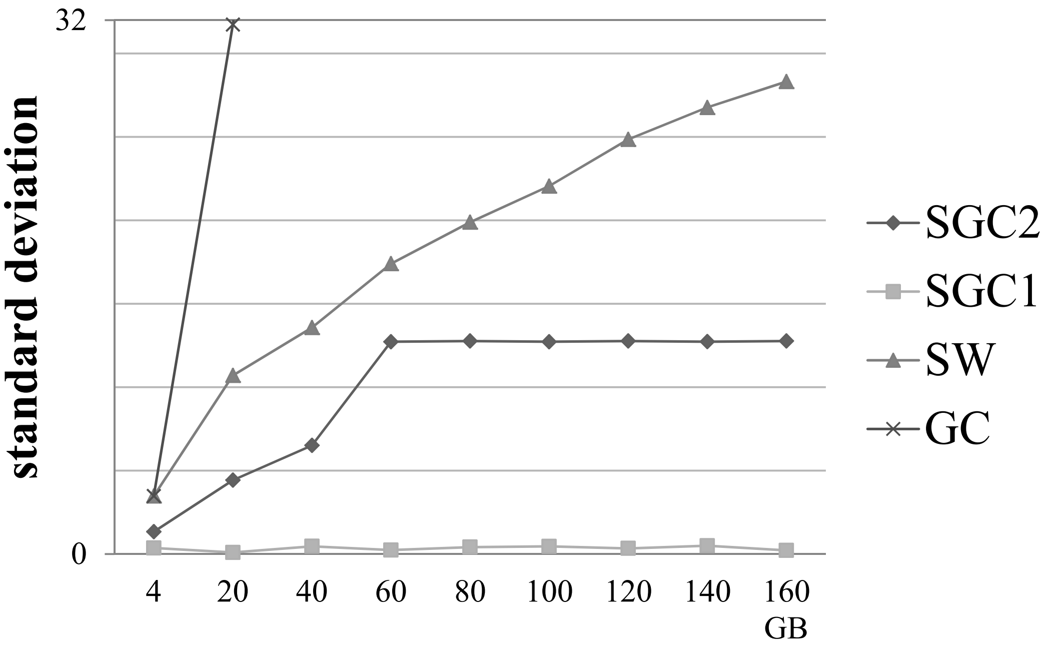


Fig.3.8: changes in standard deviations

3.4.5 Extra overhead

Extra overhead of wear leveling includes: extra block erase and extra live-page copy. As shown in Fig.3.9 and Fig.3.10, GA will collect the block which has maximum invalid pages so it has lowest overhead of both block erase and page copying. SW is bases on GA; if it is not triggered, SW and GA have the same performance in live-page copy. If SW is with a high threshold (threshold=100), it will not perform wear leveling action frequently, so it does not increase much overhead as compared with GC. The SGC1 algorithm always erases blocks in a sequential manner; it does not consider the number of invalid pages in each block. Therefore, SGC1 has higher living page copy, besides it will slow down the performance of the flash memory. The SGC2 algorithm erases the block that has more than 75% invalid pages with a higher priority so it can decrease the overhead with each erase.

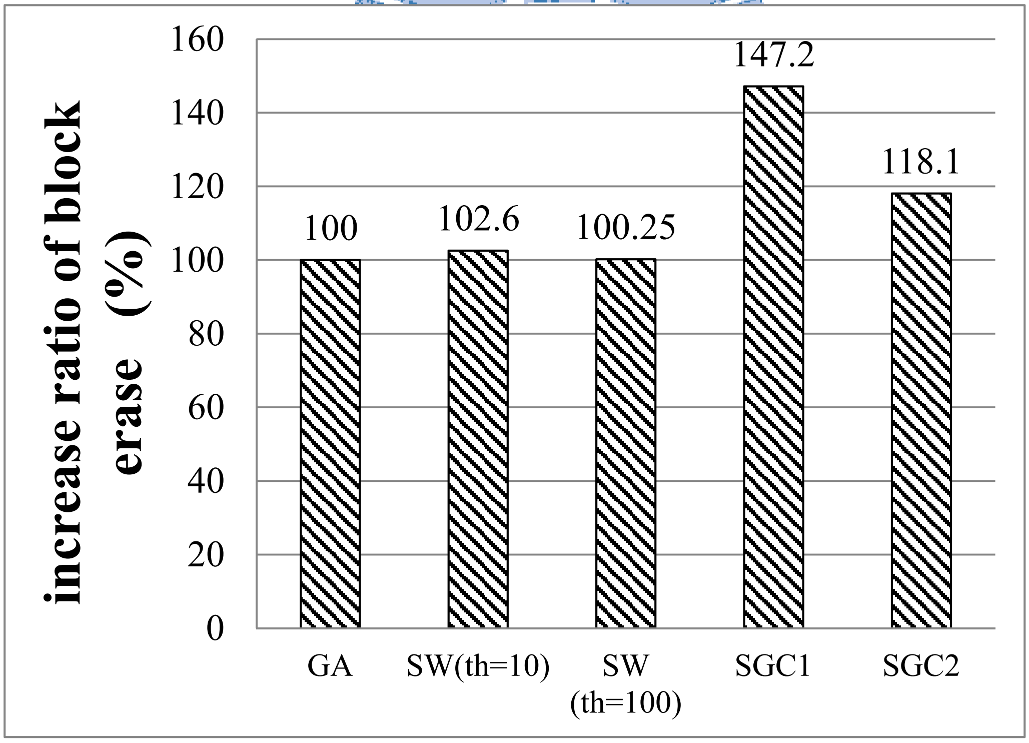


Fig.3.9: The increased ratio of block erase(%)

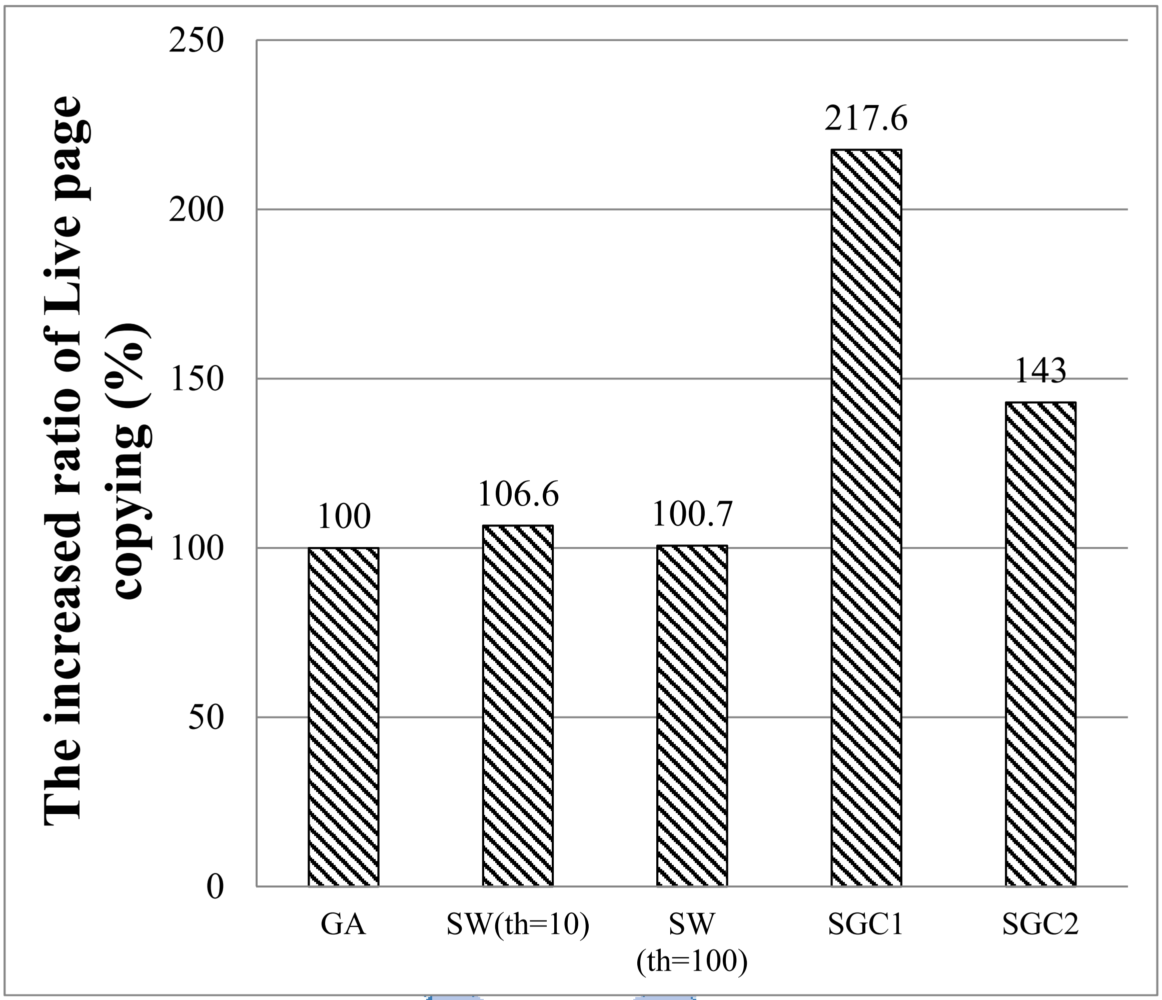


Fig.3.10: The increased ratio of Live page copying (%)

3.4.6 Total cycles for writing data

Fig.3.11 shows the total cycles for writing 120GB data in four algorithms. In Fig.3.11, the “write” means the total cycles taken in page write operations, and “erase” means the total cycles in erase operations. The “search” is the total cycles in searching the block to be erased in each algorithm. The “live page copy” means the total cycles spent in living pages copy.

In SGC1, the efficiency in recycling the space occupied by invalid pages is not as good as greedy algorithm, thus it requires more extra erase operations and results in more cycles spent in living pages copy. SGC2 uses the IPF vector to reduce the overhead in SGC1, and it can reduce the total cycle time as compared to SGC1. Although the proposed SGC2 algorithm has longer execution cycles than SW for writing 120GB data, it is rare to continuously write so many data in a 2GB flash memory. As a result, the read and write speed degradation will not be so worse in a real application.

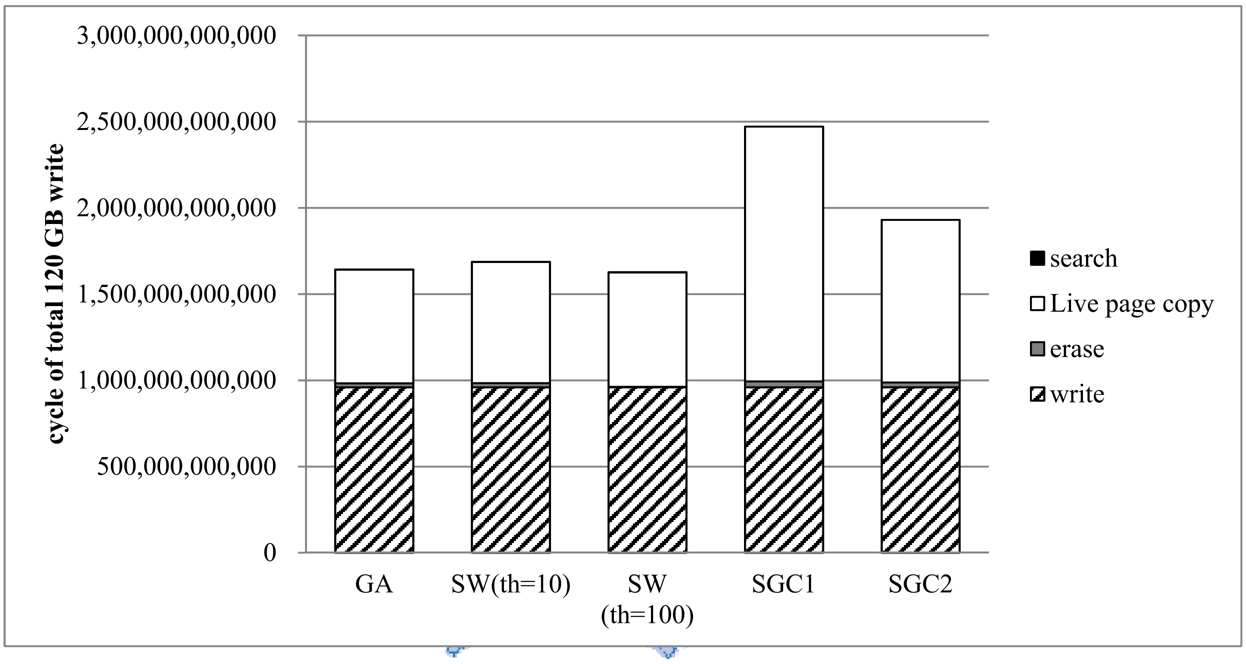


Fig.3.11: Total cycles for writing 120GB data.

3.5 Summary

Table 3.2 shows the summary for four methods tested in 120GB write request. GA has great performance in recycling invalid pages. Thus it has both lower overhead and higher speed for flash memory execution. However, it causes the flash memory retire early with unbalanced block utilization. SW with GA algorithm can reduce the overhead of living page copy. However, it needs more RAM space to manage the BET table. SGC1 can achieve lowest maximum erase count and standard deviation. Otherwise, SGC1 has higher overhead in living page copy because it does not consider number of invalid pages in the block, so it may erase the block which stored a lot of valid pages. SGC2 improves the disadvantage of SGC1, and it can reduce the overhead in SGC1. Although there are a little increase in maximum erase count and standard deviation is SGC2,we still choose SGC2 for a flash memory system for performance consideration.

Table 3.2: Performance Comparisons.

	GA	SW	SGC1	SGC2
Average erase count	93.9	96.3	138.1	110.8
Max erase count	632	210	139	157
Standard deviation	208.6	24.9	0.3	11.2
Normalized living pages copy (%)	100	106.6	217.6	143.0
Total cycles for writing 120GB data ($\times 10^9$)	1,661	1,707	2,471	1,958
Normalized total cycles for writing 120GB data (%)	100	102	148	117
RAM space (bit)	4096*17	4096*(17+1)	0	4096

Chapter 4 FPGA simulation result

We use the Socle Technology Corporation MDK-3D development board to verify the proposed SGC algorithm. The CPU is ARM1176JZF and the frequency is up to 1GHz. The AHB frequency is up to 200MHz and support the NOR-flash/NAND-flash/DDR2.

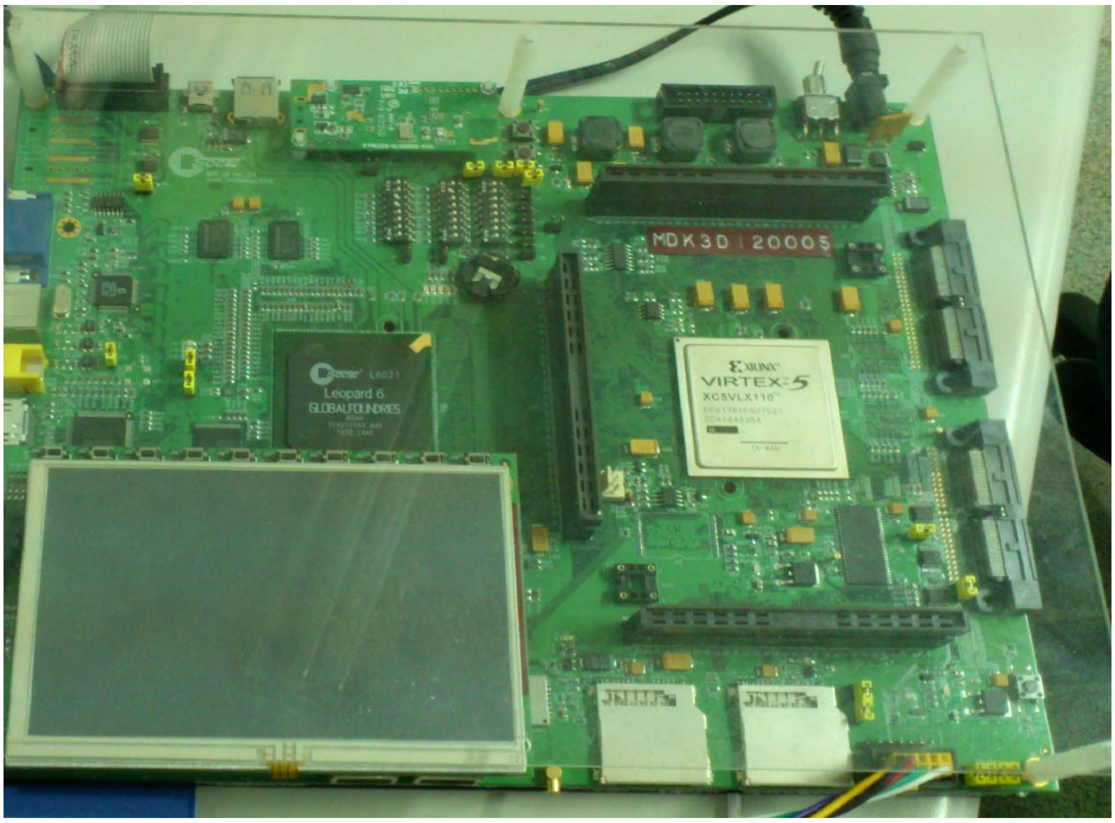


Fig.4.1: MDK-3D board photo

We use Verilog code to rebuild the SGC algorithm. It contains 4096 blocks and each block contains 128 pages. FTL is a complexity design and needs large memory space to store mapping table, so we do not implement it in FPGA simulation. Therefore, we produce PPAs to control our flash memory.

Fig.4.2 is the block diagram of our design in FPGA testing. Flash controller consists with sequential garbage collection. Flash memory stores data and information for flash controller. Test patterns use the bus to writing data into flash controller and store in 2GB flash memory.

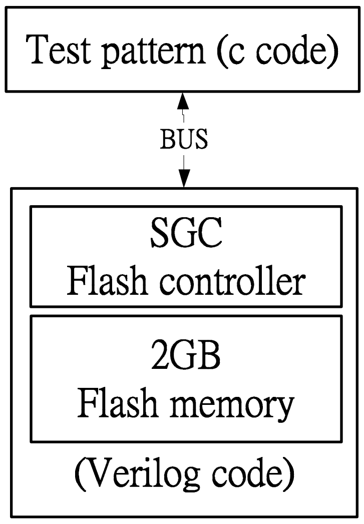


Fig.4.2: FPGA testing module

Fig.4.3 is the block diagram of our design in FPGA. Address and command translate into system controller and memory controller respectively with AHB bus; patterns are been stored into SRAM. After testing, each block erase count in the flash memory is translating into APH bus.

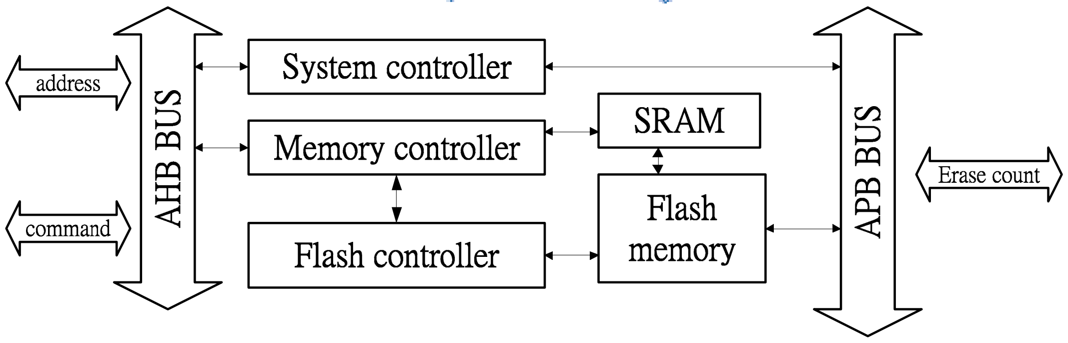
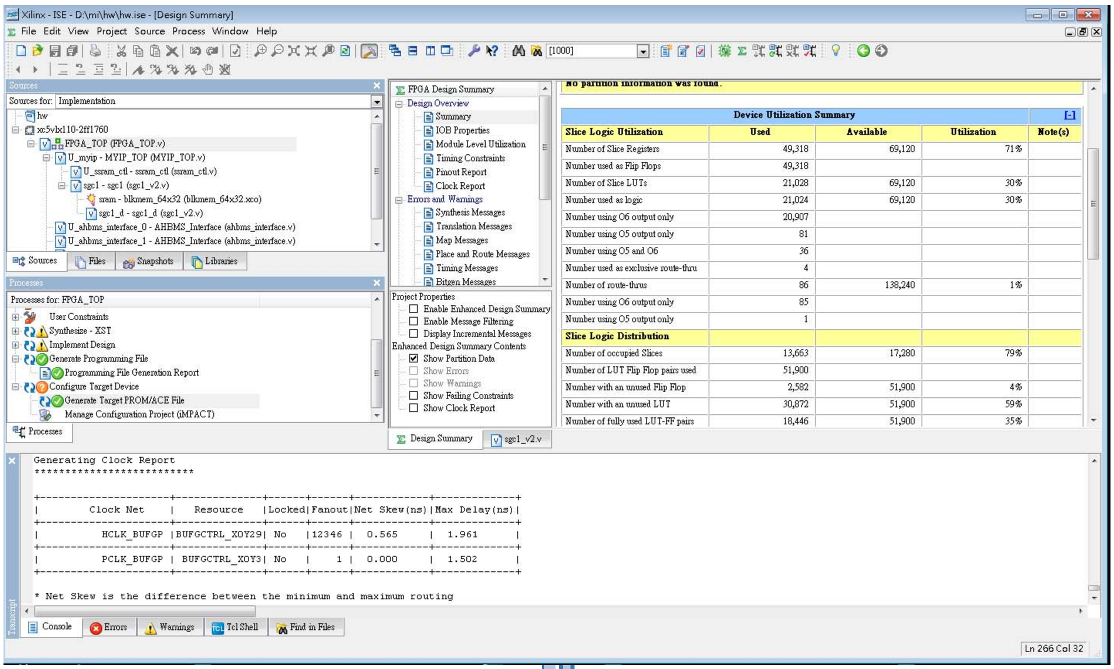


Fig.4.3: FPGA design module

In FPGA design, we rebuild a 2GB flash memory module to implement our SGC algorithm design. There have same performance of FPGA simulation and C simulation, therefore it can prove the algorithm of SGC can implement in hardware.

SGC algorithm can balance erase count of blocks also can implement in hardware.



No partition information was found.

Device Utilization Summary				
Slice Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Registers	49,318	69,120	71%	
Number used as Flip Flops	49,318			
Number of Slice LUTs	21,028	69,120	30%	
Number used as logic	21,024	69,120	30%	
Number using O6 output only	20,907			
Number using O5 output only	81			
Number using O5 and O6	36			
Number used as exclusive route-thru	4			
Number of route-thrus	86	138,240	1%	
Number using O6 output only	85			
Number using O5 output only	1			
Slice Logic Distribution				
Number of occupied Slices	13,663	17,280	79%	
Number of LUT Flip Flop pairs used	51,900			
Number with an unused Flip Flop	2,582	51,900	4%	
Number with an unused LUT	30,872	51,900	59%	
Number of fully used LUT-FF pairs	18,446	51,900	35%	

```
Generating Clock Report
**************************

+---------------------+----------------+------+------+------------+-------------+
|      Clock Net      |    Resource    |Locked|Fanout|Net Skew(ns)|Max Delay(ns)|
+---------------------+----------------+------+------+------------+-------------+
|          HCLK_BUFGP |BUFGCTRL_X0Y29| No   |12346 |  0.565     |  1.961      |
+---------------------+----------------+------+------+------------+-------------+
|          PCLK_BUFGP | BUFGCTRL_X0Y3| No   |     1 |  0.000     |  1.502      |
+---------------------+----------------+------+------+------------+-------------+

* Net Skew is the difference between the minimum and maximum routing
```

Fig.4.4: FPGA hardware design

Table 4.1: Device utilization summary of FPGA implement.

Number of slice register	49,318
Number of slice LUT	21,024

Because the performance results of FPGA implement is the same as C simulation. It can prove that sequential garbage collection (SGC) methods can the implemented by hardware.

Chapter 5 Conclusion and future work

In this thesis, a low-complexity high-performance wear-leveling algorithm for flash memory system design is presented. The proposed SGC algorithm combines GC operation and wear-leveling into a single process, and it doesn't require any tuning for threshold parameters. Simulation results show that the proposed SGC algorithm has the lowest maximum block erase count and smallest standard deviation. In addition, the low-complexity low-cost SGC algorithm makes it is easy to be implemented by firmware-based or hardware-based approaches. Thus the lifetime of the flash memory can be greatly lengthened by the proposed SGC algorithm.

The SGC algorithm has high overhead needs to be improved, thus SGC2 can reduce 30 % overhead of SGC1. By reducing overhead, we can increase performance of the flash memory. In the future, we can use SGC algorithm to maintain standard deviation and addition other effective policy to reduce overhead in living page copy.

Reference

[1] M-Systems, "Two Technologies Compared: NOR vs. NAND," http://139.138.48.19/pdf/NAND/MSystems/MSystems_NOR_vs_NAND.pdf

[2] TOSHIBA, "NAND vs. NOR Flash Memory Technology Overview," http://www.maltiel-consulting.com/NAND_vs_NOR_Flash_Memory_Technology_Overview_Read_Write_Erase_speed_for_SLC_MLC_semiconductor_consulting_expert.pdf

[3] GRUPP, L., DAVIS, J., AND SWANSON, S., "The bleak future of nand flash memory," http://static.usenix.org/events/fast/tech/full_papers/Grupp2-8-12.pdf

[4] ELNEC, "NAND Flash Memories and Programming NAND Flash Memories Using ELNEC Device Programmers," http://www.elnec.com/sw/an_elnec_nand_flash.pdf

[5] Samsung, "The Samsung SLC NAND flash Advantage," http://www.psism.com/SLC%20vs%20MLC.pdf

[6] TOSHIBA, "SLC toggle," http://www.semicon.toshiba.com.tw/product/memory/selection/nand/slc/toggle/index.html

[7] Samsung Electronics, "K9GAG08U0M 2G×8bit NAND Flash Memory Data Sheet," 2006.

[8] Intel, "Understanding the flash translation layer (FTL) specification," http://developer.intel.com, 2010.

[9] T.-S. Chung, D.-J. Park, S. Park, D.-H. Lee, S.-W. Lee, and H.-J. Song, "A survey of flash translation layer," in *Proceedings of Journal of systems Architecture (JSA)*, Vol.55, pp. 332-343, May. 2009.

[10] Atsuo Kawaguchi, Shingo Nishioka, and Hiroshi Motoda, "A flash-memory based file system, " *in Proceedings of USENIX Annual Technical*

Conference, Jun. 1995, pp. 155-164.

[11] Micron, "Wear-Leveling Techniques in NAND Flash Devices," http://www.micron.com/~/media/Documents/Products/Technical%20Note/NAND%20Flash/151tn2942_nand_wear_leveling.pdf

[12] Wear-Leveling and Life Span, "Wear-Leveling and Life Span," http://www.magicram.com/images/uploads/file/Wear%20Leveling%20Mechanism.pdf

[13] Micron, "Bad Block Management in NAND Flash Memory," http://www.micron.com/~/media/Documents/Products/Technical%20Note/NAND%20Flash/tn2959_bbm_in_nand_flash.pdf

[14] Lin-Pin Chang and Tei-Wei Kuo, "Efficient management for large-scale flash-memory storage systems with resource conservation, " *ACM Transactions on Storage (TOS)*, vol. 1, no. 4, pp. 381-418, Nov. 2005.

[15] M-Systems, "TrufFFSr Wear-Leveling Mechanism." http://csis.bits-pilani.ac.in/faculty/sundarb/courses/old/spr11/dstn/readings/sheets/trueffs.pdf

[16] STMicroelectronics, "Wear Leveling in Single Level Cell NAND Flash Memories," http://www.eetasia.com/ARTICLES/2004NOV/A/2004NOV29_MEM_AN09.PDF?SOURCES=DOWNLOAD

[17] SanDisk, "Sandisk Flash Memory Cards Wear Leveling," http://www.scribd.com/doc/7010332/SANDISK-Flash-Memory-Cards-Wear-Leveling

[18] T. Gleixner, F. Haverkamp, and A. Bityutskiy, "UBI -Unsorted Block Images," http://linux-mtd.infradead.org/doc/ubidesign/ubidesign.pdf

[19] Li-Pin Chang, "On efficient wear leveling for large-scale flash-memory storage systems," *in Proceedings of ACM Symposium on Applied Computing (SAC)*, Mar. 2007, pp. 1126-1130.

[20] Wen-Kai Hsieh and Hsi-Pin Ma, "Conditional threshold wear-leveling algorithm for multi-channel NAND flash memory, " *in Proceedings of International Symposium on VLSI Design, Automation, and Test (VLSI-DAT)*, Apr. 2010, pp. 147-150.

[21] Kee-Hoon Jang and Tae-Hee Han, "Efficient garbage collection policy and block management method for NAND flash memory," *in Proceedings of International Conference on Mechanical and Electronics Engineering (ICMEE)*, Aug. 2010, pp. 327-331.

[22] Yuan-Hao Chang, Jen-Wei Hsieh, and Tei-Wei Kuo, "Improving flash wear-leveling by proactively moving static data, " *IEEE Transactions on Computers*, vol. 59, no. 1, pp. 53-65, Jan. 2010.

[23] Yuan-Hao Chang, Jen-Wei Hsieh, and Tei-Wei Kuo, "Endurance enhancement of flash-memory storage systems: an efficient static wear leveling design," *in Proceedings of ACM/IEEE Design Automation Conference (DAC)*, Jun. 2007, pp. 212-217.

[24] Li-Pin Chang and Li-Chun Huang, "A low-cost wear-leveling algorithm for block-mapping solid-state disks," *in Proceedings of ACM SIGPLAN/SIGBED Conference on Languages, Compilers, Tools and Theory for Embedded Systems (LCTES)*, Apr. 2011, pp. 31-40.

[25] H. S. Lee, H. S. Yun, and D. H. Lee, "HFTL: hybrid flash translation layer based on hot data identification for flash memory," *IEEE Trans. Consumer Electronics*, vol. 55, no. 4, pp. 2005-2011, Nov. 2009.

[26] "NAND flash price", http://www.cnyes.com/fc/metal/fc_flash.asp