



저작자표시-비영리-변경금지 2.0 대한민국

이용자는 아래의 조건을 따르는 경우에 한하여 자유롭게

- 이 저작물을 복제, 배포, 전송, 전시, 공연 및 방송할 수 있습니다.

다음과 같은 조건을 따라야 합니다:



저작자표시. 귀하는 원저작자를 표시하여야 합니다.



비영리. 귀하는 이 저작물을 영리 목적으로 이용할 수 없습니다.



변경금지. 귀하는 이 저작물을 개작, 변형 또는 가공할 수 없습니다.

- 귀하는, 이 저작물의 재이용이나 배포의 경우, 이 저작물에 적용된 이용허락조건을 명확하게 나타내어야 합니다.
- 저작권자로부터 별도의 허가를 받으면 이러한 조건들은 적용되지 않습니다.

저작권법에 따른 이용자의 권리는 위의 내용에 의하여 영향을 받지 않습니다.

이것은 [이용허락규약\(Legal Code\)](#)을 이해하기 쉽게 요약한 것입니다.

[Disclaimer](#)

**Master's Thesis**

**Executing Hard Real-Time Programs on NAND Flash  
Memory Considering Read Disturb Errors**

**August 2017**

**Graduate School of Seoul National University**

**Computer science**

**Victoria Shangina**

Executing Hard Real-Time Programs on NAND Flash Memory Considering Read  
Disturb Errors

지도교수 이창건

이논문을 공학석사학위논문으로 제출함

2017년 6월

서울대학교 대학원

컴퓨터 공학부

빅토리아

빅토리아의 석사학위 논문을 인준함

2017년 8월

위원장 민상렬

부위원장 이창건

위원 하순희



## **Abstract**

Victoria Shangina

School of Computer Science and Engineering

Seoul National University

Recently, the demands for NAND flash based storage devices increased significantly due to fast grow of interest to the IoT and embedded systems. Along with undeniable advantages that usage of NAND flash can give, there are still unsolved issues concerning reliability. This dissertation presents overview of existing techniques that allow overcoming this reliability issues.

More specifically, this research suggests a technique that will allow overcoming reliability issue of the embedded system that uses NAND flash memory for program execution. This paper introduces a technique that will allow managing READ DISTURB ERRORS and improving reliability in the case when the NAND memory is used as storage of executable programs in Real-Time system. This dissertation purposes applying relocation technique to frequently read or READ HOT pages. The relocation technique is invoked when the physical limit of read operations of NAND page – Threshold – is about to be reached. Also, this technique takes on account several limitations related to hard real-time systems constraints and NAND flash memory physical features.

With the implementation of this technique, the improvement in terms of reliability and RAM size in comparison with shadowing and HRT-PLRU was overviewed. In terms of reliability, the improvement is that new approach guarantees READ DISTURB will not happen. In terms of RAM size, new approach required 36% more RAM pages that HRT-PLRU approach and 40% less RAM pages that Shadowing approach

Keywords: Real-time, NAND Flash Memory, Hard Real-Time, Read Disturb, Page relocation

Student number

2015-23299

# Contents

Abstract .....	i
Contents .....	iii
List of figures .....	iv
1 Introduction .....	1
2 Related works .....	6
3 Background and problem description .....	10
3.1 NAND flash memory .....	10
3.2 HRT-PLRU .....	13
3.3 Reliability issues of the NAND flash memory .....	18
3.4 Problem description .....	28
3.5 System notation .....	30
4 Approach .....	31
4.1 Per-task analysis .....	31
4.2 Convex optimization .....	37
5 Evaluation .....	41
6 Future works .....	46
7 Conclusion .....	47
Summary (Korean) .....	48
References .....	49

## List of Figures

Equation 1: RAM partition .....	16
Equation 2: Total RAM size .....	16
Equation 3: Relocation task period calculation .....	37
Equation 4: Relocation time .....	38
Figure 3-1: NAND flash memory plane. Adapted from.....	11
Figure 3-2: NAND flash chip architecture .....	12
Figure 3-3: HRT-PLRU scheme .....	13
Figure 3-4: Partitioned RAM approach.....	15
Figure 3-5: Read disturb.....	21
Figure 3-6: Read Disturb.....	28
Figure 3-7: Execution sequence (a) without relocation (b) with relocation .....	28
Figure 3-8: NAND page flow graph .....	29
Figure 4-1: (a) RAM conflict graph and (b) RAM state transition graph .....	32
Figure 4-2: RAM configuration .....	33
Figure 4-3: Per-Task analysis .....	35
Figure 4-4: Result of Per-Task analysis.....	36
Figure 4-5: Transformation from $WCET_i(S_i)$ to $U_i(S_i)$ and $U_i(S_i)$ .....	39
Figure 5-1: Relocation tasks design .....	42
Figure 5-2: Per-task analysis results.....	45
Figure 5-3: Comparison of approaches basing on pages number needed .....	45

Table 1: NAND vulnerabilities and its solutions.....	20
Table 2: NAND flash characteristics.....	38
Table 3: Task characteristics.....	41
Table 4: Relocation task characteristics .....	42

# 1 Introduction

During the last two decades embedded systems became widespread in our society. Many of the embedded systems applications require real-time constraints. For example modern cars became tremendously sophisticated and they contain various scanners, sensors and other components that require sufficient management. Beside this, car production companies following the recent embedded systems trends started to include mature infotainment system. Hence, modern cars have to contain 50 to 100 Electronic Control Unit(s) (ECU(s)) which execute a number of applications, ranging from safety critical control systems to comfort systems. The timely reaction of those applications to user and sensor input is of preeminent importance. These requirements become even more rigid in case of the autonomous cars.

For embedded systems to provide further advanced features software size has been significantly increased. For instance, General Motors automobiles in 1995 contained 1 million lines of program code. Premium class automobile in 2009 contained close to 10 million lines of software code. The article by Robert N. Charette in IEEE SPECTRUM states that the S-class Mercedes-Benz requires over 20 million lines of code alone and that the car contains nearly as many ECUs as the Airbus A380. Moreover, autonomous car does not seem to be unrealistic in 2017 because many car companies and universities (Nissan, Mercedes Benz, Naver, Tesla Motors

and etc.) announces self-driving car projects to be going. In case of self-driving cars in addition to regular car infotainment system and inner parts control there are radar/lidar for obstacle detection, vision processing for lane keeping, image recognition and path-generation programs. Such a sophisticated processing requires even larger amount of code.

Also, car embedded system must support real-time constraints to ensure safety and fast response. Typically, a real-time system comprises applications which have strict timing requirements, e.g. restriction on their finishing time (execution deadline). Most of these applications consist of a collection of small tasks that run concurrently on a processor. Significant place in the real-time systems theory is given to scheduling algorithms. Thus, automobile code execution must be represented as recurrent tasks which have a fixed frequency of execution and a deadline by which execution must be finished.

However, ECU architecture with small RAM and flash memory had not supported that amount of code along with need of hard real-time timing properties. Revision of those problems was made in series of researches that include “HRT-PLRU: a new paging scheme for executing hard real-time programs on NAND Flash memory” by Kyoung-Soo We, Chang-Gun Lee, Kyongsu Yi, Kwei-Jay Lin and Yun-Sang Lee.

This work is following the intension to use NAND flash memory for

executable code storage. NAND flash memory has been widely used for mobile embedded systems for a long time. It became popular due to non-volatility, shock resistance, and low-power consumption features of NAND flash memory.

However, NAND Flash memory has challenging physical characteristics such as page-based read-write operations, long delay for write operation, erase-before-write requirement, garbage collection delay and limited erase counts. Especially, the fact that a read operation is performed only on 2KB page-basis – byte-level random access is not allowed – makes it hard to use NAND based Flash memory for executable code storage. Moreover, that fact makes it impossible to execute program directly from NAND flash memory.

Moreover, NAND flash memory has variety of reliability issues that can provoke unpredictable delays and faults. That is why this research intention is to address Read Disturb reliability issue for hard real-time program execution on NAND.

Most commonly used approach in industry is *shadowing* that copies the entire program codes in NAND flash memory into RAM and executes them from RAM. However, this approach requires a large amount of RAM which sacrifices the price benefit of NAND flash memory. In order to run program in NAND flash with a smaller RAM capacity *demand-paging* mechanism

was used. This approach requires big time overhead as pages are constantly pushed into RAM and pushed from RAM according to LRU policy.

To address this issue the shadowing and demand-paging approach must be combined. In this research it is considered that each hard real-time task has an allocated partition of RAM. Also, each of these allocated parts of RAM is divided into pinned and demand-paging areas. The combination of these two areas is called RAM configuration. Depending on RAM configuration access pattern to NAND pages can be different. Thus, many various RAM configurations are considered in current dissertation.

Additionally, to overcome reliability issues NAND page relocation technique is suggested. Thus, to choose an optimal RAM configuration beside of read delays relocation delays need to be considered. As developed approach consider multiple concurrent hard real-time tasks, optimal RAM configuration must be found for each task separately and combines in a final RAM configuration. This final RAM size supposes to be minimal among all possible RAM sizes that allow addressing requirements stated before.

The intention of this thesis is to guarantee hard real-time constraints by managing reliability issues of NAND-flash memory with minimal possible RAM configuration. This research proposal is to address this problem as following. Basing on a known value of Threshold - number of reads after which NAND flash memory can start experience reliability issues - schedule

relocation task such a way so all hard real-time constraints are met. Suggested framework will include two main steps: (1) Per-Task analysis, (2) Convex optimization. The result of these two steps is a final minimal RAM configuration.

The remaining structure of this report is as follows. In Chapter 2 the overview of related works is represented. Chapter 3 includes the necessary background and definitions that are used in the rest of the thesis. Chapter 4 gives a suggested technique to address NAND flash reliability issues in case of read intensive access pattern. The performance analysis of the suggested technique is made in the Chapter 5. Future works are discussed in Chapter 6. Finally, research conclusion is presented in the Chapter 7.

## 2 Related works

As it has been stated before, embedded systems evolved in a degree, that to provide services to users it needs an enormous amount of code. In this connection, researchers started to consider NAND Flash memory as an alternative to a NOR Flash memory in terms of executable instructions (executable code) storage.

In series of researches were presented design flows for using NAND flash memory for soft/hard real-time programs.

The first design flow called RT-PLRU (real-time constrained pinning and LRU combination) was presented a framework that allows execution of a program from NAND flash. The main limitation of this initial approach to use NAND memory such a way was that only single task Soft Real-Time application was supported. As this is a Soft Real-Time approach, it considers that deadline can be violated at some point of execution. However, this is not a critical issue (as the RT-PLRU is targeting the media player portable systems) unless percent of frames executions that met deadline is greater than threshold probability [1].

As RT-PLRU approach was unpractical in terms of tasks number that can be executed, it was extended. The following approach called mRT-PLRU allowed to execute several tasks on one embedded system [1]. Deadline meeting of each task was guaranteed with probability higher than some

denoted probability. The technique included two steps. On the first step a per task analysis were performed. The result of this step was a function of “RAM size vs. optimal execution time”. With use of this function on the next step – stochastic-analysis-in-loop optimization results in minimal RAM size. With this RAM size embedded system supposes to satisfy the deadline meet probability. However, the main limitation of this approach is that it could not operate with data that had large difference with test data [2].

To extend applicability of mRT-PLRU the changes were presented by DuHee Lee in “Real-time code execution on NAND flash memory supporting wide-spectrum of input data for multiple tasks” [32]. As in mRT-PLRU main unsolved issue was that it could not work if the real input data had large difference from the sample data, the extension considered real-time tasks of mobile systems handling video contents as input.

Some of previous limitations were improved in next approach called HRT-PLRU. In contrast with previous RT-PLRU and mRT-PLRU frameworks HRT-PLRU was targeting general multi-tasking hard real-time system. It supported execution of six real-time programs with variable periods and code sizes. As a previous mRT-PLRU it considers that optimal RAM configuration combines pinned and on-demand LRU pages. By analyzing each program code with use of NAND-page flow graph adopted from similar approach in cache hit/miss analysis framework work resulted in “allocated RAM partition size vs. WCET” function. This function was

further analyzed and used in convex optimization. With use of convex optimization adopted from mRT-PLRU the final RAM size was found. This approach was later extended for sharable RAM approach in April 2014 work [3].

Other stack of researchers also put some effort to use NAND flash memory for instruction code execution [4]. Their intention was to deal separately with temporal and spatial locality. For that matter they considered a NAND flash memory system containing a buffer system. This buffer was divided in two parts: a fully associative temporal buffer for temporal locality and a fully associative spatial buffer for temporal locality. This has been done for effectivity reasons. Treating branch and serial instruction separately by different buffer parts allowed them to lower miss ratio.

As for NAND flash reliability related researches, many of them analyze different reliability issues and agree that many of them are product of the NAND flash architecture limitations and grow of cell per level fraction [5], [6]. The techniques suggested can be divided by NAND operation type and reliability issue nature of origin. Some researchers concentrated on reliability issues caused by write disturbs [7], [6], [8], while others are considering erase operation issues [9], [10], [11], [12], [13], [14], [15], [16] and only few are considering a read operation related issues [17], [18], [19], [20], [21]. In terms of real-time systems NAND flash reliability issues are considered as a source of possible unpredictable delays. To overcome this

issues various modifications of FTL were applied. These modifications address garbage collection [22], relocation techniques [19], and programming strategies [23] that overcome unpredictability of reliability issues.

### **3 Background and problem description**

#### **3.1 NAND flash memory**

Features of NAND Flash memory such as non-volatility, shock resistance, and low-power made it popular in embedded system area.

In comparison with HDD Flash memory NAND does not have moving parts as its main storage unit is floating gate. NAND Flash memory is a type of EEPROM (Electrically Erasable Programmable Read-Only Memory) that supports read, program and erase as its basic operations. The main component of NAND memory chip is the Flash memory array. Array is organized into planes. Each plane has a page buffer that senses and stores data to be read from or programmed into plane. Each plane physically represented by blocks. Each block composed of pages. So each plane can be overviewed as two-dimensional grid composed of rows – bit-lines, and columns – word-lines. And in the intersection of rows and columns is floating gate that stores voltage – logical bit of data (Figure 3-1).

However, in spite advantages, NAND has an important limitation: the data that is already written on flash memory cannot be updated immediately; block has to be erased before writing new data. This characteristic is also known as erase-before-write.

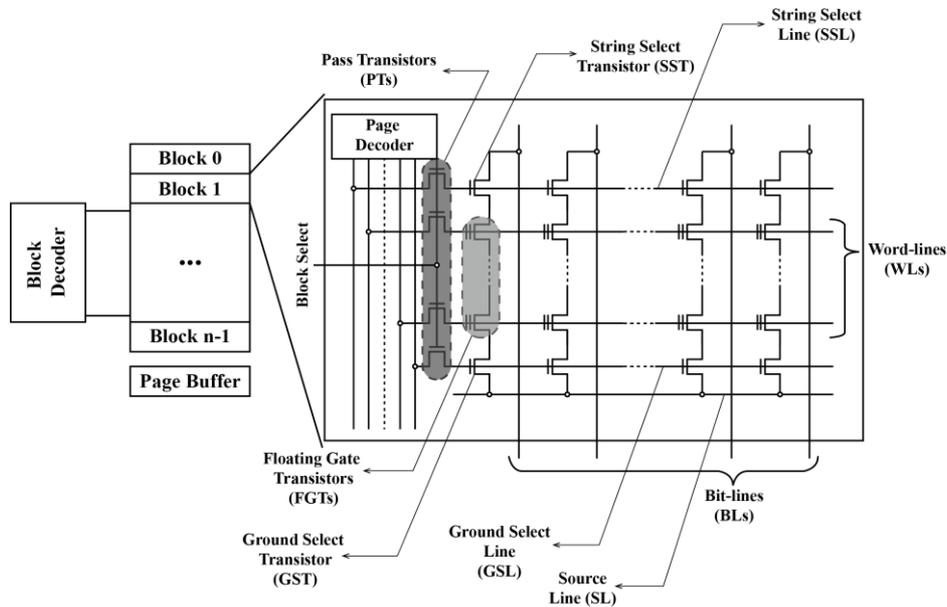


Figure 3-1: NAND flash memory plane. Adapted from [24]

To overcome this difficulty Flash Translation Layer was introduced (FTL). FTL is a widely used software technology, enabling general file systems to use Flash memory-based storage device in the same manner as a generic block device such as a hard disk. FTL provides core functionalities such as address translation, bad block management and Error Correction Code (ECC) checking. To overcome erase requirement of NAND, FTL shadows erase operation by different ways depending on FTL modification. From the File Management System point of view, when it requests write operation, only write happens.

A NAND flash memory chip consists of a number of blocks as shown in Figure 3-2.

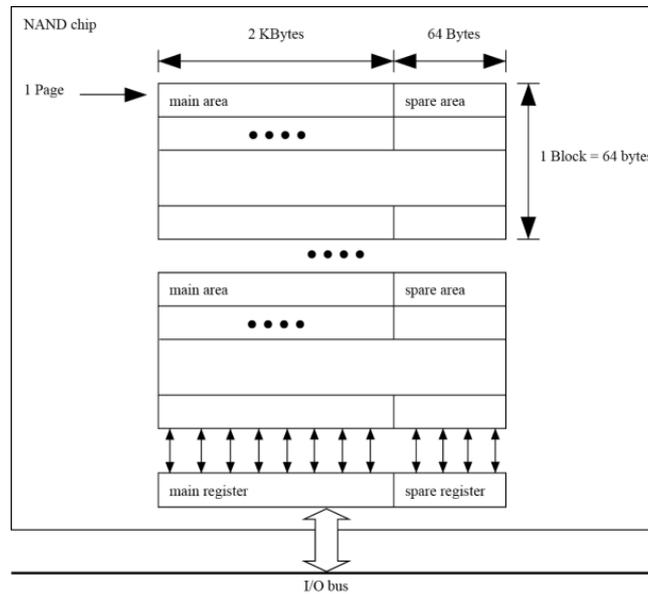


Figure 3-2: NAND flash chip architecture

Typically, each block had 64 pages where each page consists of 2 KBytes main area and 64 Bytes spare area [25]. The main area is used to store regular (valuable) information such as program codes/data while the spare area is used to store auxiliary information such as ECC and page-mapping information. Each page is read or written through a one-page size internal register (i.e., main register + spare register). Related to this internal structure, NAND flash memory exhibits challenging characteristics: (1) 2KB page based read/write, i.e., no byte-level random access, (2) long delay write, (3) erase before write with even a longer delay and erase count limits, and (4) garbage collection delay.

### 3.2 HRT-PLRU

Previously stated characteristics of NAND Flash based memory make it hard to execute programs directly from it. Therefore, in “HRT-PLRU: a new scheme for executing hard real-time programs on NAND flash memory” RAM is used as working storage for executing programs as shown in Figure 3-3. It is important to note that price of RAM is at least 10 times more expensive than that of NAND flash memory. That is why main target of the previous work was to minimize the required RAM size to reduce the unit production cost.

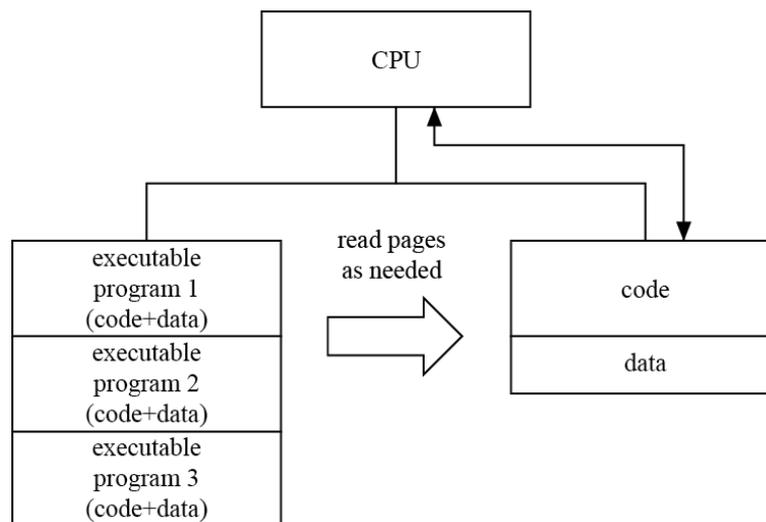


Figure 3-3: HRT-PLRU scheme

Also, Mihai Matei in January article “SSDs to become better & cheaper thanks to 3D NAND Flash” in Android Headlines states that in future due to the newly introduced technology, named 3D NAND, NAND flash memory may become cheaper. This will make it more popular type of storage in

future. Thus, attempted solution to use NAND flash memory as code storage can potentially become more of current interest for embedded system developers.

Introduction of 3D NAND flash technology can close the gap between SSDs and HDDs. 3D NAND flash memory which – unlike 2D/planar solutions – can stack cells vertically will maintain larger NAND cells. This can increase performance and endurance, but more importantly, Solid State Drive solutions using 3D NAND flash technology have the potential to overcome the key disadvantages compared to conventional HDDs: specifically the price-per-gigabyte gap and the overall storage capacity [26].

As we are speaking about using NAND flash as code storage, there are two opposite alternatives of memory usage and data distribution in system. First is to preload and keep (i.e. “pin”) all the NAND pages into the RAM. This is known as shadowing approach, the actual program execution does not cause any NAND read time. However, it requires one RAM page for every NAND page resulting in a huge RAM cost. Oppositely, if all the NAND pages are on-demand (i.e., “demand-paging”) requested into a single RAM page, it causes frequent RAM page faults whenever the program tries to access codes and data in a different NAND page. This can provoke deadline miss of the running tasks. And, as hard real-time is considered, deadline misses are not allowed. That is why as it was stated in previous work [3] main

target was to optimally balance pinning and demand-paging. It was done for both partitioned RAM approach and shared RAM approach.

In partitioned RAM approach partitions of RAM was optimally assigned to each task. It is important to determine its optimal paging policy such that the total RAM size is minimized as shown in Figure 3-4.

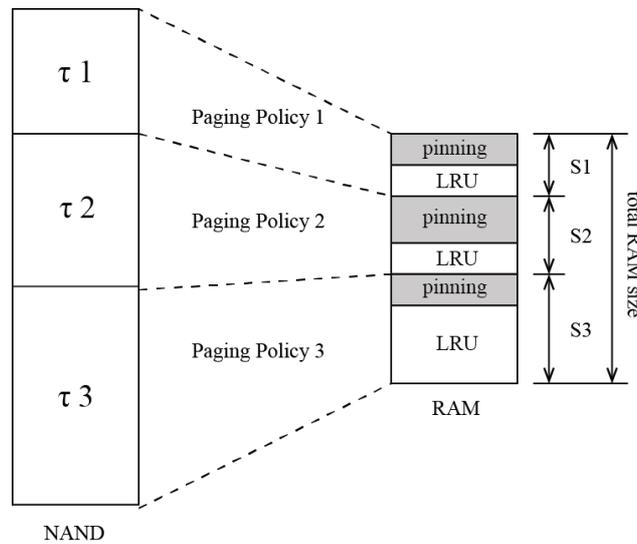


Figure 3-4: Partitioned RAM approach

The optimal paging policy, concerning this, means the optimal combination of (1) the pinning policy which preloads certain pages and keeps them into RAM and (2) the LRU policy which loads non-pinned pages into RAM on-demand while replacing pages based on the Least Recently Used (LRU) policy [3]. Total RAM size is minimized while deterministically guaranteeing the hard real-time deadlines of all the tasks.

For partitioned RAM approach process of the RAM size optimal

minimization can be described by two steps: (1) per-task analysis and (2) convex optimization.

On the first step each task is analyzed one by one to find the relation between “allocated RAM partition size versus worst case execution time (WCET)”. For each task  $\tau_i \in \Gamma = \{\tau_1, \tau_2, \dots, \tau_n\}$  framework finds its allocated RAM partition size  $S_i$  and its paging policy that includes (1) the number of RAM pages used for pinning denoted as  $S_i^{pinning}$ , (2) the LRU policy which loads non-pinned pages into RAM on-demand while replacing pages based on the Least Recently Used (LRU) policy denoted as  $S_i^{LRU}$  (Equation 1).

$$S_i^{LRU} = S_i - S_i^{pinning}$$

Equation 1: RAM partition

On the second step algorithm conducts a convex optimization to allocate minimum possible RAM to the tasks under the deterministic hard real-time schedulability constraint. Such that the total RAM size (Equation 2) is minimized while deterministically guaranteeing the deadlines of all the  $n$  tasks  $\Gamma = \{\tau_1, \tau_2, \dots, \tau_n\}$ .

$$\min \sum_{i=1}^n S_i$$

Equation 2: Total RAM size

The result of these two steps is optimally determined partition size of the RAM for every task. Thus using the partition size the combination of pinning/on-demand that is optimal to given tasks can be determined.

However, this process does not take on account physical features of the NAND flash memory. Unpredictable delay due to the NAND flash reliability issues is a target of this research. As it has been stated before main purpose of this research is to develop a technique that will address reliability issues connected with read intensive access pattern in NAND Flash memory.

### **3.3 Reliability issues of the NAND flash memory**

There are various NAND Flash reliability issues. Since 1989 when NAND flash memory was eventually introduced due to manufacturer's requirements for memory plane to be smaller and contain bigger capacity, NAND-flash planar floating gate technology had to shrink. Thus most of the electrical properties became worse. And nowadays researchers have to recognize that ability of planar floating gate architecture to shrink reached its limit.

Another factor that affects reliability is number of read and write operations that are performed on the NAND based device [5]. The reliability issues appeared much more frequently in uneven workloads.

From hardware point of view program operation involves tunneling charges to the floating gate, erase operation involves tunneling charges off the floating gate. However, granularity of these two operations is different: program operation is performed on the page granularity while erase operation is performed on granularity of block. The matter of fact, granularity of program operation and read operation is the same (both operations performed on the page level).

Read operation can effect neighboring pages. This effect is known as Read Disturb. The Read Disturb issue was deeply overviewed in several researches such as [7], [17], [18], [19], [20] and [21]. It was experimentally proved that Read Disturb can effect on consistency of information stored on

the NAND Flash memory [24].

During the research it appeared that main vulnerabilities of flash memory that can make it unreliable are coming from NAND architecture and the features of the physical process of the capture/emission of the single electrons [8], [5], [27].

Some authors diversify all problems due to its nature of appearance [8]. Other authors also are insisting on that fact that NAND architecture has weaknesses that make in unstable. Both insist on that NAND's reliability issues are based on architecture vulnerabilities and oxide features [5]. Some researchers give methods that can be used for reliability issues improvements [8], [28], [29].

In Table 1 main problems with their main cause and ways of improvement that researchers suggest were combined.

The targeting problem in this research is a Read Disturb error that can happen in case of read intensive access pattern. Read Disturb error can be included into the group of issues connected with disturbs (Table 1).

Read Disturb error is a result of the flash architecture. Inside each flash cell, data is stored as the threshold voltage of the cell, based on the logical value that the cell represents [7].

Issue group	Issue	Solution
Issue connected with oxide	Hot Hole Injection; Oxide degradation; Overprogramming; Data retention.	Management policies; Wear-leveling techniques; Error Correction codes.
Issues connected with disturbs	Program disturbs; Read disturbs; Gate Induced Drained Linkage (GIDL).	Deep trench isolation; Uniform Channel Program and Erase for reduces cell size; Highly scalable Flash-based Field Programmable gate Array;
Issues connected with interference	Cell-to-cell interference	Post compensation schemes; Pre-distortion schemes; Reduces symbol pattern of interfering cells for multi-level cells; Device/circuit techniques; Signal processing base cell-to-cell compensation method; Air gap in gate space and active space; Charge of circuit operation scheme.
Issues that appear randomly	Telegraph noise; Injection statics; Temperature instabilities.	Management policies; Need to change planar architecture.

Table 1: NAND vulnerabilities and its solutions

During a read operation to the cell, a read reference voltage is applied to the transistor corresponding to this cell. If this read reference voltage is higher than the threshold voltage of the cell, the transistor is turned on. Within a Flash block, the transistors of multiple cells, each from a different Flash

page, are tied together as a single bitline, which is connected to a single output wire. Only one cell is read at a time per bitline.

In order to read one cell (i.e., to determine whether it is turned on or off), the transistors for the cells not being read must be kept on to allow the value from the cell being read to propagate to the output. This requires the transistors to be powered with a pass-through voltage ( $V_{\text{pass read}}$  - pass read cells voltage), which is a read reference ( $V_r$  - read cells voltage) voltage guaranteed to be higher than any stored threshold voltage. Even though these other cells are not being read, this high pass-through voltage induces electric tunneling that can shift the threshold voltages of these unread cells to higher values (stressed cells in Figure 3-5), thereby disturbing the cell contents on a read operation to a neighboring page [17].

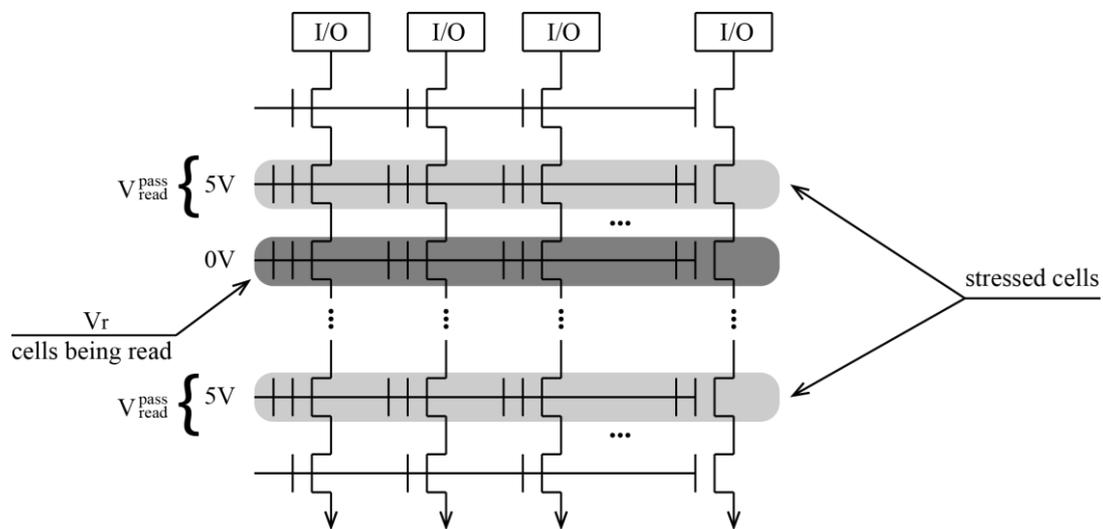


Figure 3-5: Read disturb [21]

As far as industry required size of flash cells to scale down, the transistor

oxide becomes thinner, which in turn increases this tunneling effect. With each read operation having an increased tunneling effect, it takes fewer read operations to neighboring pages for the unread flash cells to become disturbed (i.e., shifted to higher threshold voltages) and move into a different logical state.

There is an interesting question why the problem of the Read Disturbance and the problem of the Read Disturbance in case of read intensive access pattern (further “Read Hot”) have being ignored for a long time. There is opinion that it has being so because a single level flash (SLC) architecture Read Disturb errors were only expected to appear after an average of one million reads to a single flash block [30] on the first generation of multi-level cell (MLC) architecture devices it was expected to exhibit read disturb errors after 100,000 reads [30], [21]. However, it resulted in that new MLC devices are prone to read disturb after as few as 20,000 reads. This number can drop even further with scaling down the cells size [30], [21].

Also, it has been discovered that exposure of Read Disturb error can become worse in case of uneven distribution of reads across Flash blocks in workloads, where certain Flash blocks experience high temporal locality and can, because of this, more rapidly exceed read counts at which read disturb errors are induced.

To examine up to date NAND Flash memory researchers committed various

experiments on 2Y-nm MLC NAND flash chip [7]. These experiments resulted in following conclusions. With increasing number of pages and increasing number of program/erase cycles on a block read disturb on threshold voltage distribution and raw bit error rates increase. Also, cells with lower threshold voltages are more susceptible to errors as a result of read disturbs. Moreover, they checked what will happen if change pass-through voltage [7]. On the one hand, if decrease pass-through voltage Read Disturb effects on each individual operation becomes smaller. But on the other hand, read errors can increase due to reduced ability in allowing the read value to pass through the unread cells.

This evidences that to overcome the Read Disturb negative effects a proper NAND flash memory policy need to be introduced. Such management policies in case of the NAND flash memory are most commonly embodied by FTL.

Different modifications of the FTL were introduced lately. Some authors assume that garbage collection operation can produce delay on the upper layers of the system, which are File System and OS [9], [22], [31]. This is crucial for RT systems as this can produce deadline misses of the urgent high priority tasks. One of the approaches is basically to divide the garbage collection task into partial steps such that the time was taken to perform each step is no longer than the largest atomic flash operation and interleave them between the read/write operations [9]. Other paper suggests dividing

all the memory area into three parts: the valid data, invalid data, and the free area [22]. The main idea of this approach is as following. Flash memory controller generates partial garbage collection task, same as it is done in previously mentioned [9], when the number of the free pages in the flash memory is below the predefined threshold. And as the free space can be used as a write buffer, garbage collection can be delayed as long as possible. That is why this approach is named LAZY. The RFTL approach taking the similar idea and adapts it for RT area in such a way that garbage collection of the victim block can be divided into the  $k$  periods [31]. The main difference is that it defines the primary block, replacement block and the buffer block. Initially, all write requests are served by the primary block than when the block is mostly containing garbage information the valid pages are copied to the replacement block. At that time write requests are served by buffer block. When the buffer block has garbage to collect valid data are written to the primary block. Thus, by the time primary block becomes buffer block, the buffer block - replacement block and replacement block turn into a primary. The main disadvantage of this works is that they are only considering that write operation can produce affected pages.

There is also research that focuses on the Read Disturb [19]. The researchers are convinced that frequently read data can produce Read-Disturb and effect data, as well as effect number of read cycles that can be produced by the memory unit. They are introducing three improvements among which the

most interesting one is data migration to the Hot or Cold region. The Cold and Hot read data requires different optimal read voltages. Thus the memory space is divided into the Hot and Cold region in such a way that initially all the data is stored in the Cold area and then migrates to the Hot.

However, as FTL due to its nature can violate hard-real time constraints of the system read disturb overcoming technique must be applied as a separate task that can be scheduled alongside with other tasks.

### 3.4 Real-Time Garbage collection

As in our particular case we do not perform write operation frequently only for the relocation, Garbage Collection (GC) may seem not to be a problem, however, with the time the free space on the NAND flash can reach the limit of free pages. That is why we need to consider GC.

The goal of a real-time GC is to bound space and time overheads of memory management. Since many Real-Time applications must operate with limited CPU and memory resources, it is essential that the overhead of the GC to be small enough to fit in that budget, so developers could be able to reason about the impact of selecting a particular GC algorithm on their application.

For this matter regular GC is not an option as it may produce unpredictable delays. That is why we consider adopting Real-Time Garbage Collection (RTGC). Many modifications of GC were suggested in the RT field. We decided to adopt Hard RTGC developed by T. Kalibera [37]. HRTGC thread runs periodically with the highest real-time priority. This means that, at regular intervals, GC will preempt application threads and perform a fixed amount of GC work.

Time predictability is often the main concern when selecting an RTGC. From the point of view of a real-time RT task that must meet a deadline, three things matter: (a) what is the maximum blocking time due to GC, (b) how many times can it be interrupted by GC, and (c) what is the worst-case

slowdown due to the extra barrier checks needed on heap reads and writes? Since the amount of work performed by GC is fixed and the intervals are known a priori, it is possible answer these questions (a), (b) and (c), there are also compiler-inserted barriers because the GC must be incremental.

We assume the GC to be a periodic task with fixed period,  $T_{GC}$ . The period is chosen to be equal to the GC cycle which spans from the time GC activity starts until all unreachable objects have been freed. The work to be done by the GC depends on the memory operations performed by RT tasks: each time the application allocates, loads from, or stores to heap pointer variables, work needs to be done by the GC.

As HRTGC task has a higher priority and calculate its period and schedule its activity based on the already existed tasks in system it will not affect schedulability and HRT execution of the tasks.

### 3.5 Problem description

With the above knowledge and motivation in mind, this dissertation problem can be specifically described as following.

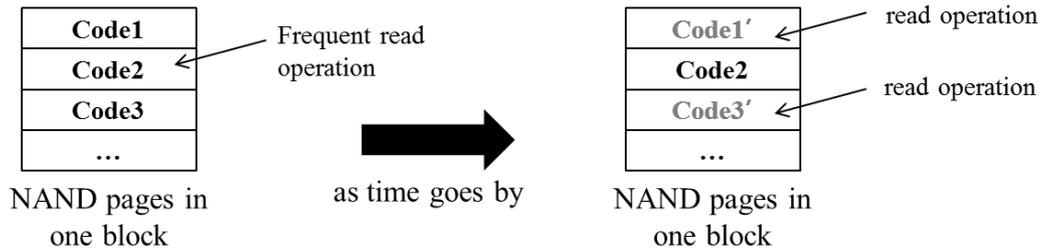


Figure 3-6: Read Disturb

With time, data, due to Read Disturb, code can become effected (Figure 3-6). To avoid read disturb errors, we have to rewrite a block before codes in the block are invalidated (Figure 3-7).

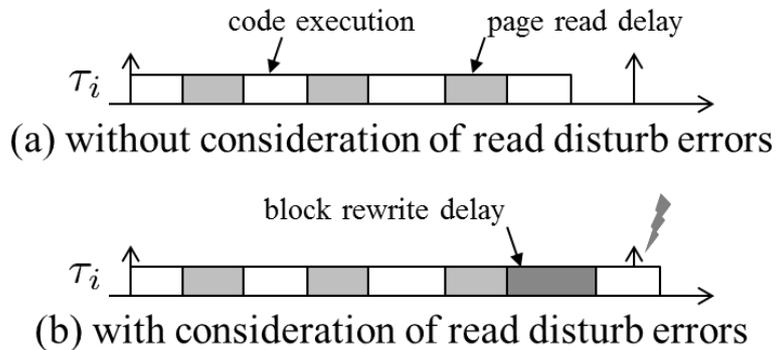


Figure 3-7: Execution sequence (a) without relocation (b) with relocation

With use of NAND page flow graph (Figure 3-8) we can reach the RAM conflict graph (Figure 4-1), and RAM state transition graphs for various

RAM configurations (Figure 4-1). Basing on them the read counts for each NAND page for this configuration and WCET for different RAM configurations can be calculated. Thus, the result of the first step is a function showing “WCET versus allocated RAM size” relation. Further, it is transformed into “Utilization versus allocated RAM size” relation.

In this step Block relocation tasks are also designed.

With use of convex optimization optimal RAM configuration for a task set that includes Block relocation tasks among all found configurations can be chosen.

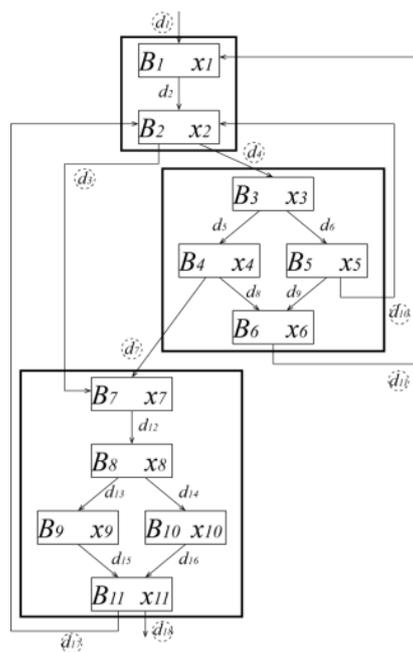


Figure 3-8: NAND page flow graph

### 3.6 System notation

This research problem can be described as follows: for each task  $\tau_i \in \Gamma = \{\tau_1, \tau_2, \dots, \tau_n\}$  where  $\tau_i$  can be represented by tuple  $\tau_i = (C_i, T_i, D_i)$ , where  $C_i$ : number of pages,  $T_i$ : the period,  $D_i$ : the relative deadline.

TH denotes the maximum number of read operations of each NAND page that can guarantee that the stored data is stable. Further this Threshold is used for Block relocation task design.

$S_i$  is a RAM size allocated to task  $\tau_i$ .  $S_i$  also represents RAM paging policy that includes (1) the number of RAM pages used for pinning denoted as  $S_i^{\text{pinning}}$ , (2) the LRU policy which loads non-pinned pages into RAM on-demand while replacing pages based on the Least Recently Used (LRU) policy denoted as  $S_i^{\text{LRU}}$ . Depending on combinations of  $S_i^{\text{LRU}}$  and  $S_i^{\text{pinning}}$  the access pattern can differ, as a page that is in  $S_i^{\text{pinning}}$  doesn't need to be read from NAND.

## **4 Approach**

The overall approach can be divided into two steps: (1) Per-Task Analysis where we analyze each task one by one to find the relation between “allocated RAM size versus WCET”, (2) Convex Optimization which allocates the minimum possible RAM to the tasks meeting hard real-time deadlines.

### **4.1 Per-task analysis**

Adapting ILP methods for cache fault analysis we can construct control flow graph [3] that shows the sequence of code execution for the current program. As it is known which code section of the program is stored on which NAND page physically NAND page flow graph can be constructed (Figure 3-8).

Further, analysis of NAND page flow graph allows constructing RAM conflict graph (Figure 4-1 (a)). It is important to notice that an edge that starts and ends inside the NAND page does not cause the fault. That is why the transitions that are targeted to produce a conflict graph are that those are causing faults.

Further, the access pattern for different RAM configurations (combinations of pinning and LRU) is to be calculated. For that matter for each RAM configuration RAM state transition graph (Figure 4-1 (b)) is constructed.

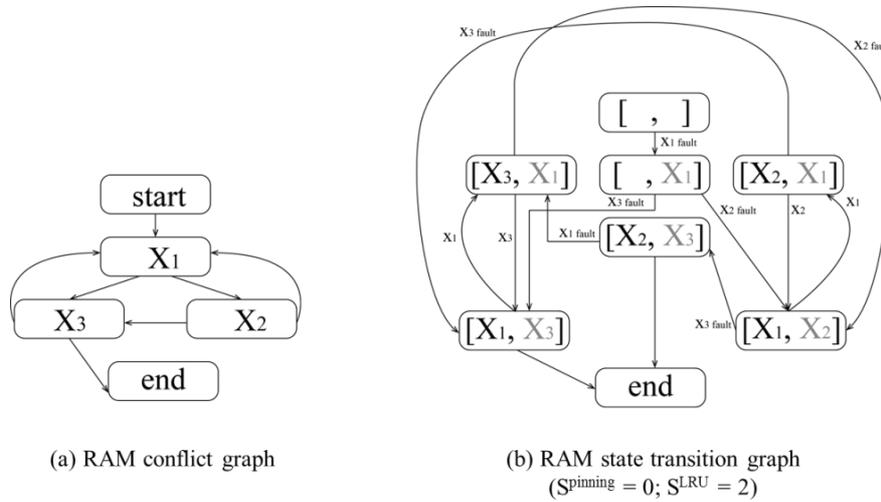


Figure 4-1: (a) RAM conflict graph and (b) RAM state transition graph

For example task  $\tau_1$  has program  $C_1$  and it requires 10 NAND pages. At the starting point the RAM configuration is one RAM page with no pinned pages. Having one RAM page which is pinned will make execution impossible as all needed information from the left nine pages cannot be read. For this configuration with use of RAM conflict graph the RAM state transition graph (Figure 4-1(b)) can be received. Then the additional RAM page is considered on besides of initial RAM configuration. This gives two possible RAM configurations: (1) two RAM pages with no pages pinned, (2) two RAM pages with one pinned page. Further, with number of RAM pages increased the number of RAM configurations will grow until the number of RAM pages is equal to number NAND pages required by a code (code basic blocks size). This configuration was earlier referred as a shadowing – the configuration where number of RAM pages and NAND pages are equal and

all RAM pages are pinned (Figure 4-2). Further increasing of number of the RAM pages (number of pages higher than 10) will not give any improvement in terms of execution time minimization it will not be considered.

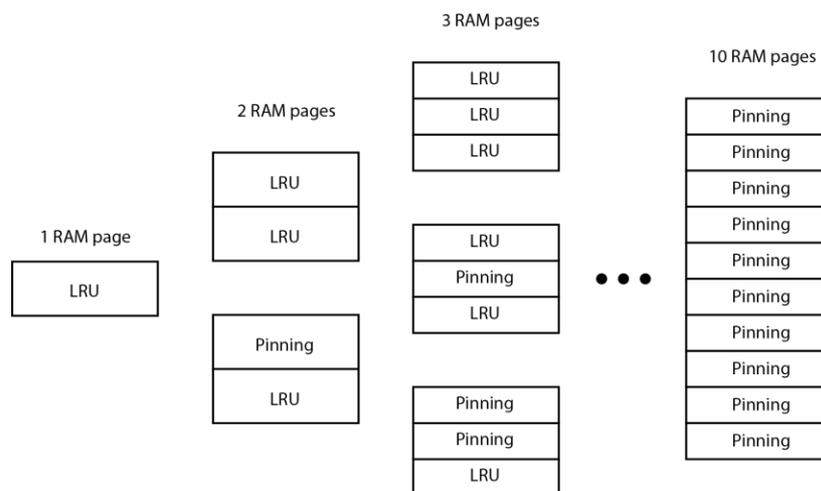
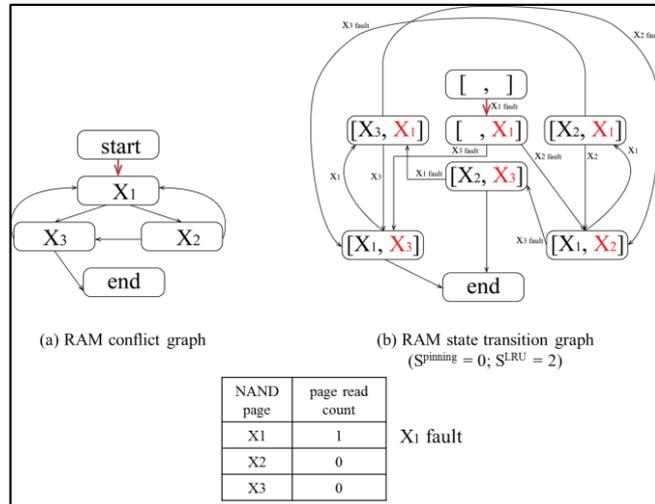


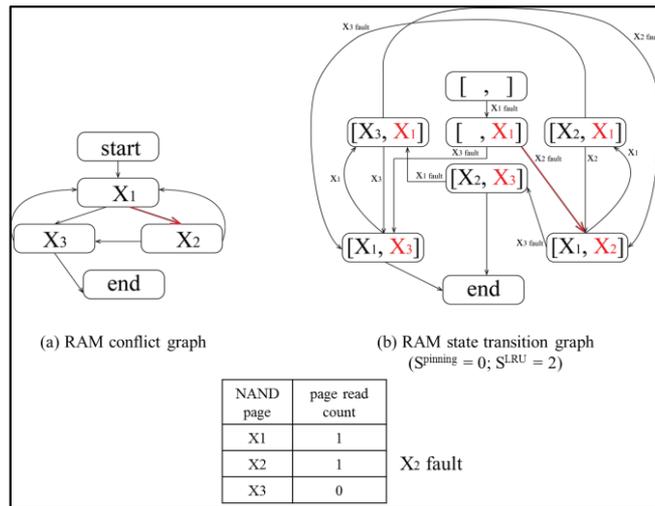
Figure 4-2: RAM configuration

In our example (Figure 4-3) we consider a configuration where number of RAM pages is 2 and number of pinned pages is zero.

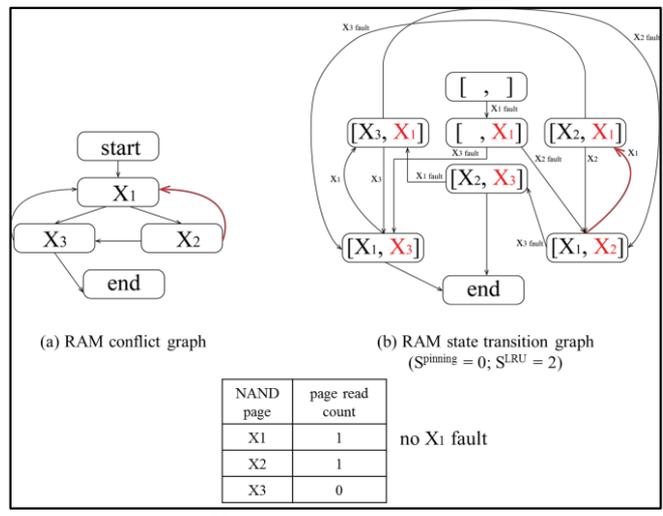
First, X1 read page operation cause fault as X1 page is not loaded in the RAM – the page read count for X1 page is increased (Figure 4-3(a)). Next read operation is read operation of the page X2. As it is also not loaded into the RAM yet it cause fault – the page read of X2 is increased (Figure 4-3(b)). Next read operation due to RAM conflict graph is X1. As X1 has been previously loaded to RAM and has not been pushed out basing on the LRU policy this read operation does not cause any fault. That is why the read count for X1 is not increased (Figure 4-3(c)).



(a) X1 read operation



(b) X2 read operation



(c) X1 read operation repeated  
Figure 4-3: Per-Task analysis

It is important to notice that not all combinations of the configuration are considered, only combinations that can benefit for our approach. That is why complexity of this algorithm is not becoming exponential. For example, in case of configuration with 5 RAM pages where 3 of them are pinning and 2 are LRU not all 10 pages are considered as target of storing them as pinning, but the most accessed ones. The observation of most accessed ones pages is done basing on the results of execution flow analysis. Thus, only top 3 most accessed pages are selected as pinned pages in this example. This observation dramatically reduces the number of cases to be observed.

Moreover, on the last step number of pinned pages is not varying (as you can see on Figure 4-2), it is equal to 10, because all other possible cases will not have a smaller execution time than all pinning case (in this case read overhead is 0 for every page).

With use of per-task analysis relation between allocated RAM configuration (number of pages to be pinned and pages to be managed basing on LRU policy) and worst case execution time (WCET) can be found. In this case beside of actual execution time of a program code WCET read delay that happen when page is LRU and have to be replaced by other page and read to RAM repeatedly.

$$WCET = e + T_{rdelay}$$

$$T_{rdelay} = n' \times (T_{rdpg} + T_{rdoop})$$

$e$  - execution time of the program instructions that are stored on page;  
 $n'$  - number of read operation performed on NAND pages.

Finally, all found WCET for all different CPU configurations are combined in “RAM size versus WCET” function (Figure 4-4).

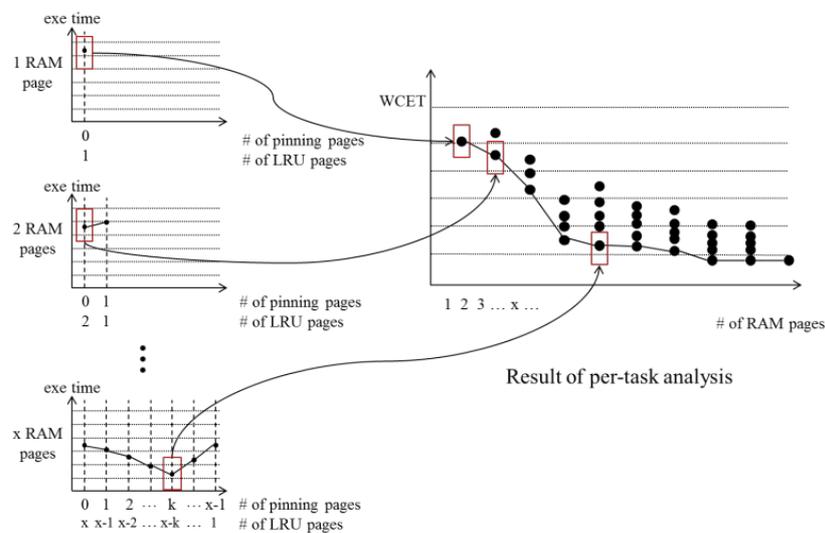


Figure 4-4: Result of Per-Task analysis

Further for convex optimization needs relation between “RAM size versus WCET” transformed to “RAM size versus Utilization”.

## 4.2 Convex optimization

On the second step with the “RAM size vs. WCET” and “RAM size vs. Utilization” relations denoted by  $WCET_i(S_i)$  and  $U_i(S_i)$  a convex optimization is to allocate minimum possible RAM to the tasks under the deterministic schedulability constraints.

At the same time, on the second step the design of the relocation tasks must be done. Number of relocation tasks depends on NAND size required by a program code. The relocation is performed by Blocks.

The relocation task relocates NAND pages on this block when cumulated number of reads for all the pages on the Block reached the TH. This relocation nulls the page count for relocated pages. As code size may be bigger or lower than block size one block may contain more than one tasks code. The priority of the Block relocation task is greater than highest priority of the task which code is stored on the block. The period of the Block relocation task is calculated basing on all period of tasks that it contains. The period calculated as maximum time until cumulated read counts reach TH.

$$\max x \text{ under } \sum_{i=1}^n \left\lceil \frac{x}{p_i} \right\rceil \times rcount_i < TH$$

Equation 3: Relocation task period calculation

Where  $x$  is maximum time until cumulated read counts reach TH,  $p_i$  is

period of  $\tau_i$ ,  $rcount_i$  is read counts of task  $\tau_i$ ,  $n$  is number of tasks.

The time taken for relocation is calculated basing on the constant values specified for NAND memory by the NAND flash memory producers (Table 2).

Characteristics	Samsung 16MB Small Block	Samsung 128MB Large Block
Block size	16384 (bytes)	65536 (bytes)
Page size	512 (bytes)	2048 (bytes)
OOB size	16 (bytes)	64 (bytes)
Read Page	36 (usec)	25 (usec)
Read OOB	10 (usec)	25 (usec)
Write Page	200 (usec)	300 (usec)
Write OOB	200 (usec)	300 (usec)
Erase	2000 (usec)	2000 (usec)

Table 2: NAND flash characteristics [25]

Using this algorithm estimates time taken for relocation of  $n$  pages. Time taken for relocation ( $T_{relocation}$ ) can be calculated as follows:

$$T_{relocation} = n \times (T_{wrpg} + T_{rdpg} + 2T_{rdoop})$$

Equation 4: Relocation time

where (1)  $T_{wrpg}$  – time required to write a page, (2)  $T_{rdpg}$  – time required to read a page, (3)  $T_{rdoop}$  – time required to read 00b area a page (need to be done twice: for read and for write), and (4)  $n$  – number of valid pages on a Block.

Starting from  $S_i = 1$  for all  $\tau_i \in \Gamma$ , our convex optimization tries to optimally increase  $S_i$  values in a way maximally reducing total system utilization such that the task set (that includes relocation tasks) becomes

schedulable. For this we transformed  $WCET_i(S_i)$  relation (Figure 4-5 (a)) to the worst case utilization relation denoted by  $U_i(S_i)$  by dividing  $WCET_i$  by period  $p_i$ . Suggested approach from  $U_i(S_i)$  constructs its convex approximated relation  $\bar{U}_i(S_i)$  consisting from only convex-hull frontiers (Figure 4-5 (b)).

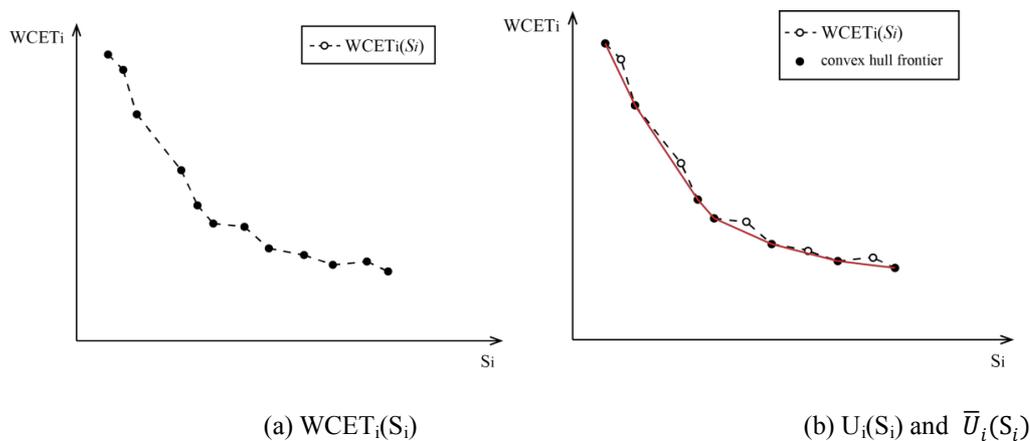


Figure 4-5: Transformation from  $WCET_i(S_i)$  to  $U_i(S_i)$  and  $\bar{U}_i(S_i)$

Starting from  $S_i = 1$  algorithm performs per-task analysis for a task set of original tasks and additional relocation tasks that were designed previously. And if the task set is not schedulable approach gives additional RAM page. As with change of configuration the number of read counts as long as number of referred pages are changed the relocation tasks have to be redesigned. For modified task set with relocation tasks which periods are calculated basing of read counts for this specific RAM configuration response time analysis performed.

The convex optimization result is a specific RAM configuration  $S_i$  with

certain number of pinned and LRU pages, that meet schedulability requirements.

## 5 Evaluation

The testing system configuration was chosen as follows:

- 6 hard real-time periodic tasks;
- 32 pages per block;
- 3 block rewriting tasks, TH = 40,000.

This research is targeting an embedded system that is a part of autonomous vehicle. Tasks that are to be running on the control computer are as following: (1) scan data (SICK sensors, HOKUYO sensors, IBEO sensor data), (2) vision processing, (3) path generation, (4) path following.

$\tau_1$ : path following;

$\tau_2$ : SICK sensors processing;

$\tau_3$ : IBEO sensor data processing;

$\tau_4$ : HOKUYO sensors processing;

$\tau_5$ : path generation;

$\tau_6$ : vision processing.

Characteristics of the tasks used in experiments are summarized in Table 3.

task	code size (pages)	period(ms)
$\tau_1$	11	50
$\tau_2$	9	80
$\tau_3$	10	100
$\tau_4$	14	100
$\tau_5$	19	100
$\tau_6$	21	400

Table 3: Task characteristics

task	period (ms)	execution time (ms)
$\tau_{B1}$	3502	27.2
$\tau_{B2}$	2302	27.2
$\tau_{B3}$	2402	17

Table 4: Relocation task characteristics

Where  $\tau_{B1}$ ,  $\tau_{B2}$ ,  $\tau_{B3}$  are relocations task designed basing on read counts for specific configuration number of RAM pages = 1, number LRU pages = 0. Furter, with algorithm execution relocation tasks characteristics are redesigned. Number of relocation task is calculated basing on the NAND size (in number of pages) required for each tasks:

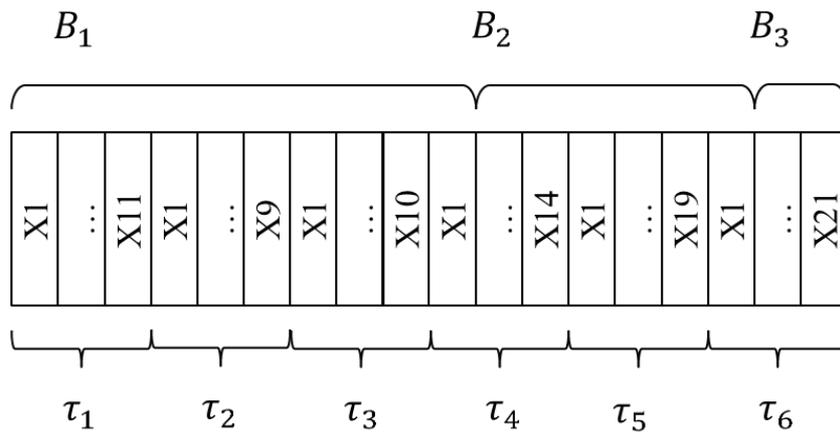
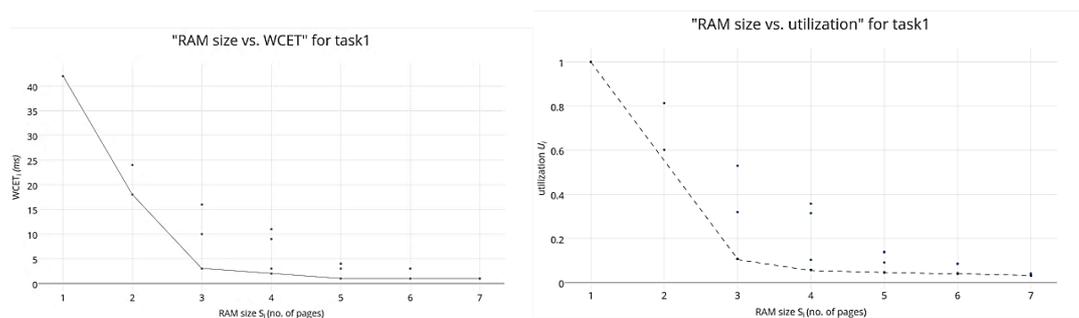


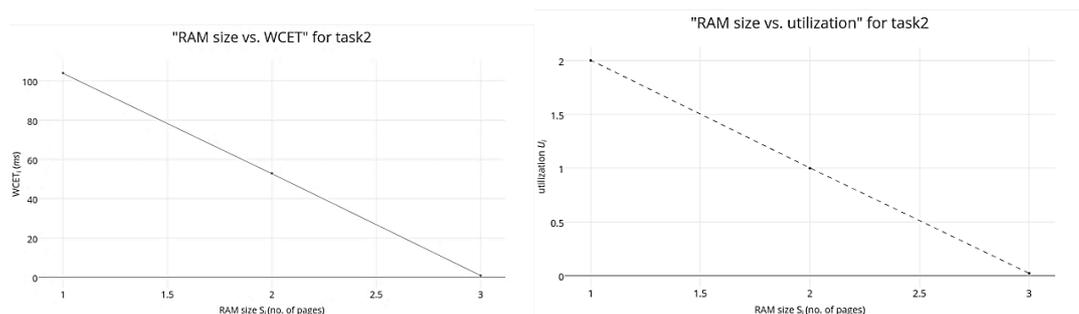
Figure 5-1: Relocation tasks design

Results of the per task analysis are presented on the Figure 5-2. It shows “RAM size  $S_i$  versus WCET” relations, their transformations to “RAM size  $S_i$  versus utilization  $U_i$ ” and also convex approximation for six tasks  $\tau_1$ ,  $\tau_2$ ,  $\tau_3$ ,  $\tau_4$ ,  $\tau_5$  and  $\tau_6$ .

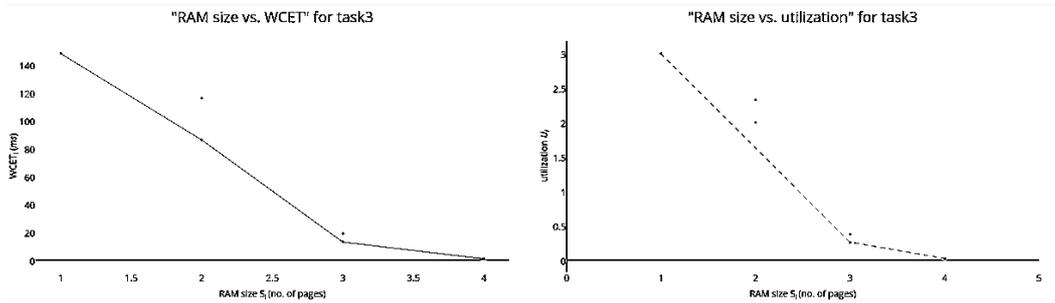
From these results two important observations can be done. Firstly, as it can be seen from “RAM size versus WCET” graphs, the WCET has significant variations for same RAM size depending on LRU/pinning combinations. Thus, optimal combination of pinning and LRU allows to reduce WCET for a given RAM size. Secondly, as it can be seen from “RAM size  $S_i$  versus utilization  $U_i$ ” and convex approximation graphs, the RAM size versus utilization relation closely matches convex approximation. Thus, the convex approximation incurs not much approximation penalty.



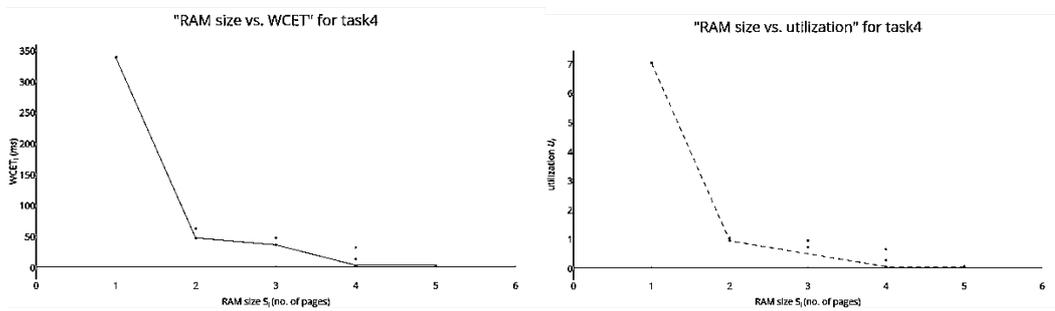
(a) Per-task analysis results for  $\tau_1$



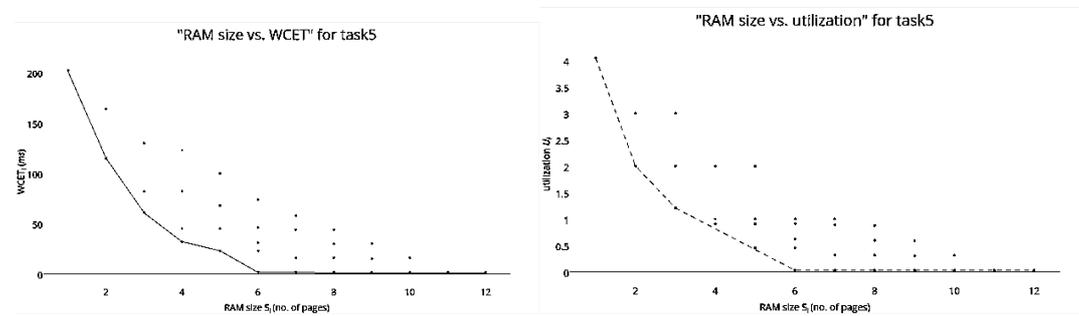
(b) Per-task analysis results for  $\tau_2$



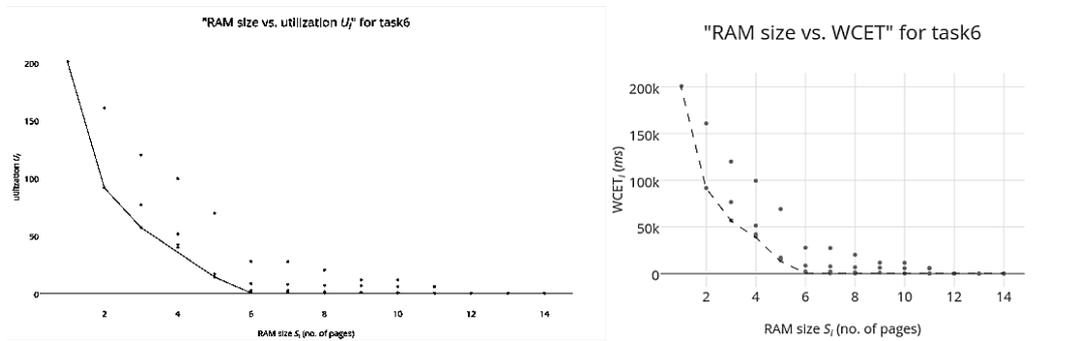
(c) Per-task analysis results for  $\tau_3$



(d) Per-task analysis results for  $\tau_4$



(e) Per-task analysis results for  $\tau_5$



(d) Per-task analysis results for  $\tau_6$

Figure 5-2: Per-task analysis results

Figure 5-3 compares all three approaches that have been overviewed: (1) shadowing approach, (2) baseline approach, and (3) proposed approach.

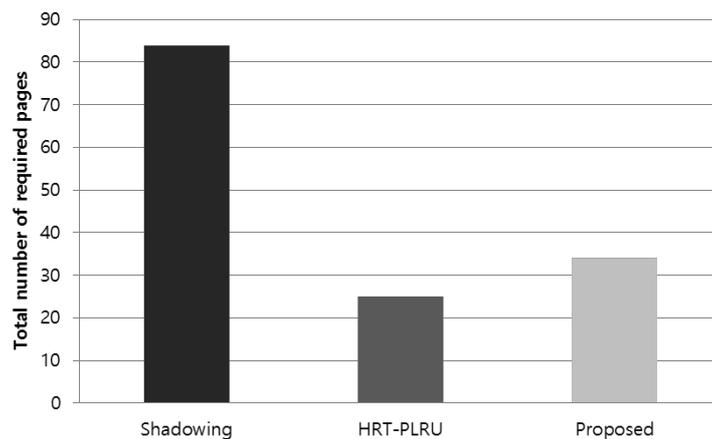


Figure 5-3: Comparison of approaches basing on pages number needed

With the implementation of this technique, the improvement in terms of reliability and RAM size for shadowing, HRT-PLRU and suggested approach were compared it can be seen that: (1) new approach required 36% more RAM pages than HRT-PLRU approach, (2) new approach required 40% less RAM pages than shadowing approach.

## 6 Future works

In future we are planning to extend our limitations and make our approach applicable in real life SSDs. This requires from our approach to consider various but not constant read delay. Now our approach assumes read delay for one page read operation to be a constant value. However, this is not fully true in real life due to SSD I/O buffer latency.

Also, as one of the main goals of the current dissertation is to minimize required RAM size we are planning to consider a system configuration with a cache between RAM and NAND memory in future. This can help to reduce minimal required RAM size and decrease number of read requests to the NAND memory chip. However, it gives a new more complexed research problem.

Another interesting direction in which stated research can go is to extend our approach toward adapting real life CPU features. For now our main consideration is that CPU stalls while one page is read from RAM. One of the probable extensions of the current work is to consider that CPU does not stall while waiting for memory access.

Additionally, in future we need to consider Garbage Collection in our model as a separate task as it has been described in the related work section. For now there is an observation that GC does not affect regular and relocation tasks.

## 7 Conclusion

We investigated issues that can make NAND flash memory unreliable. From existing reliability issue we have chosen an issue those effects on work of developed framework.

For this issue we overviewed existing techniques that allow overcoming reliability issues. With this knowledge we proposed technique that will allow overcoming reliability issue of the embedded system that uses NAND flash memory for program execution. Application of this technique required implementation of per-task analysis and convex optimization.

Also, we took on account several limitations connected with real-time systems constraints, NAND flash memory physical features, and suggested approach features.

In this connection, suggested approach meet above stated requirements to guarantee that with found RAM size for each partition designated for each task all relative deadlines are satisfied.

With the implementation of this technique, the improvement in terms of reliability and RAM size for shadowing, HRT-PLRU and suggested approach were compared it can be seen that: (1) new approach required 36% more RAM pages than HRT-PLRU approach, (2) new approach required 40% less RAM pages than shadowing approach.

## Summary (Korean)

### 요약 (국문초록)

최근 IoT 와 임베디드 시스템에 대한 관심이 급증하면서 NAND 플래시 메모리를 사용하는 장치들 또한 증가하고 있다. 이러한 장치들은 NAND 플래시 메모리를 사용함으로써 큰 이득을 얻을 수 있지만 여전히 신뢰성 측면에서는 해결되지 않은 이슈들이 있다. 본 논문에서는 이러한 신뢰성 문제를 극복할 수 있는 방안에 대해 논의한다. NAND 플래시 메모리는 각 페이지에 대해 읽기 명령을 반복적으로 수행 할 있는 물리적 회수가 한정되어 있기 때문에 읽기 횟수가 한계치에 도달하기 전에 재할당을 해주어야 하는 문제가 있다. 본 논문에서는 프로그램 코드가 저장되어 있는 read-only 페이지를 읽어 코드를 수행하는 실시간 임베디드 시스템에서 실시간 제약 조건을 만족하면서 재할당을 하여 READ DISTURB ERROR 를 줄이는 기법에 대해 제안한다.

본 논문에서 제안하는 기법을 구현하고 실험함으로써, NAND 플래시 메모리의 읽기 한계치에 도달하기 전에 재할당이 보장됨을 보인다. 또한 제안하는 기법을 사용 할 경우 요구되는 RAM 크기가 최대 48% 감소함을 확인한다

**주요어:** 실시간, NAND 플래시 메모리, Read Disturb, Page relocation

**학번:** 2015-23299

## References

- [1]. Jong-Chan Kim, Duhee Lee, Chang-Gun Lee, and Kanghee Kim, “RT-PLRU: A New Paging Scheme for Real-Time Execution of Program Codes on NAND Flash Memory for Portable Media Players”, in IEEE Transactions on Computers, Vol. 60, Issue 8, pp. 8, Aug. 2011
- [2]. Duhee Lee, Jong-Chan Kim, Chang-Gun Lee, and Kanghee Kim, “mRT-PLRU: A General Framework for Real-Time Multi-Task Executions on NAND Flash Memory”, in IEEE Transactions on Computers, Vol. 62, No. 4, Apr. 2013
- [3]. Kyoung-Soo We, Chang-Gun Lee, Kyongsu Yi, Kwei-Jay Lin, and Yun Sang Lee, “HRT-PLRU: A New Paging Scheme for Executing Hard Real-Time Programs on NAND Flash Memory”, in IEEE Transactions on Computers, Vol. 63, No. 4, Apr. 2014
- [4]. Bo-Sung Jung , Cheong-Ghil Kim, and Jung-Hoon Lee, “Fast NAND Flash Memory System for Instruction Code Execution”, in Etri Journal 34(5), Oct. 2012
- [5]. Elena Ionna Vatajelu, Hassen Aziza, Cristian Zambelli, “Nonvolatile Memories: Present and future Challenges”, in 9th International Design and Test Symposium (IDT), 2014
- [6]. Youngwoo Park, Jaeduk Lee, Seong Soon Cho, Gyoyoung Jin, and Eunseung Jung, “Scaling and Reliability of NAND Flash Devices”, in Reliability Physics Symposium, 2014
- [7]. Yu Cai, Y. L., “Read Disturb Errors in MLC NAND Flash Memory: Characterization, Mitigation, and Recovery” in 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, 2015
- [8]. Rino Micheloni, Luca Crippa, Alessia Marelli, “Inside NAND Flash Memories”, Springer, 2010

- [9]. S. Choudhuri and T. Givargis. (n.d.). "Deterministic service guarantees for NAND flash using partial block cleaning". in CODES+ISSS' 08, 2008
- [10]. Min Huang, Y. Wang, Zhaoqing Liu, Liyan Qiao and Z. Shao, "A Garbage Collection Aware Stripping method for Solid-State Drives," The 20th Asia and South Pacific Design Automation Conference, Chiba, 2015
- [11]. J. Ou, J. Shu, Y. Lu, L. Yi and W. Wang, "EDM: An Endurance-Aware Data Migration Scheme for Load Balancing in SSD Storage Clusters," 2014 IEEE 28th International Parallel and Distributed Processing Symposium, Phoenix, AZ, 2014
- [12]. H. T. Lue et al., "Radically extending the cycling endurance of Flash memory (to > 100M Cycles) by using built-in thermal annealing to self-heal the stress-induced damage," 2012 International Electron Devices Meeting, San Francisco, CA, 2012
- [13]. D. h. Lee and W. Sung, "Estimation of NAND Flash Memory Threshold Voltage Distribution for Optimum Soft-Decision Error Correction," in IEEE Transactions on Signal Processing, vol. 61, no. 2, pp. 440-449, Jan.15, 2013
- [14]. S. Jain and Yann-Hang Lee, "Real-time support of flash memory file system for embedded applications," The Fourth IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems, and the Second International Workshop on Collaborative Computing, Integration, and Assurance (SEUS-WCCIA'06), Gyeongju, 2006
- [15]. D. Kang, K. Cho and S. Kang, "A new in-field bad block detection scheme for NAND flash chips," 2015 International SoC Design Conference (ISOCC), Gyungju, 2015
- [16]. S. I. Park, D. Shin and E. G. Han, "Adaptive program verify scheme for improving NAND flash memory performance and lifespan," 2012 IEEE Asian Solid State Circuits Conference (A-SSCC), Kobe, 2012

- [17]. Y. Cai, S. Ghose, Y. Luo, K. Mai, O. Mutlu and E. F. Haratsch, "Vulnerabilities in MLC NAND Flash Memory Programming: Experimental Analysis, Exploits, and Mitigation Techniques," 2017 IEEE International Symposium on High Performance Computer Architecture (HPCA), Austin, TX, USA, 2017
- [18]. Y. Deguchi, T. Tokutomi and K. Takeuchi, "System-level error correction by read-disturb error model of 1Xnm TLC NAND Flash memory for read-intensive enterprise solid-state drives (SSDs)," 2016 IEEE International Reliability Physics Symposium (IRPS), Pasadena, CA, 2016
- [19]. Kobayashi, T. Tokutomi and K. Takeuchi, "Versatile TLC NAND flash memory control to reduce read disturb errors by 85% and extend read cycles by 6.7-times of Read-Hot and Cold data for cloud data centers," 2016 IEEE Symposium on VLSI Circuits (VLSI-Circuits), Honolulu, HI, 2016
- [20]. N. Papandreou et al., "Effect of Read Disturb on Incomplete Blocks in MLC NAND Flash Arrays," 2016 IEEE 8th International Memory Workshop (IMW), Paris, 2016
- [21]. K. Ha, J. Jeong and J. Kim, "An Integrated Approach for Managing Read Disturbs in High-Density NAND Flash Memory," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 35, no. 7, pp. 1079-1091, July 2016
- [22]. Qi Zhang. (n.d.), "Lazy-RTGC: A Real-Time Lazy Garbage Collection Mechanism with Jointly Optimizing Average and Worst Performance for NAND Flash Memory Storage Systems," in ACM Transactions on Design Automation of Electronic Systems vol.20 issue 3, 2015
- [23]. M. Huang; B. Xu; Z. Liu; Y. Xu; D, Wu, "Implicit Programming: A Fast Programming Strategy for NAND Flash Memory Storage Systems Adopting Redundancy Methods," in IEEE Embedded Systems Letters , vol.PP, no.99, 2017

- [24]. Vidyabhushan Mohan, S. G. (n.d.). “Flash Power: detailed power model for NAND flash memory,” in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2013
- [25]. Samsung Electronics, “K9F2G08U0A Datasheet ((K9F2G08UxA) Flash Memory)”, 2007
- [26]. M. Matei, “SSDs to become better & cheaper thanks to 3D NAND flash” in Android Headlines (AH), Jan. 2017, <https://www.androidheadlines.com/2017/01/ssds-become-better-cheaper-thanks-3d-nand-flash.html>
- [27]. Youngwoo Park, Jaeduk Lee, Seong Soon Cho, Gyoyoung Jin, and Eunseung Jung, “Scaling and Reliability of NAND Flash Devices”, in Reliability Physics Symposium, 2014
- [28]. G. Kong, T. K. (n.d.), “Cell-to-Cell Interference Compensation Schemes Using Reduced Symbol Pattern of Interfering Cells for MLC NAND Flash Memory”, in APMRC Digest, 2012
- [29]. Hyunyoung Shim, M. C. (n.d.), “Novel Integration Technologies for Improving Reliability in NAND Flash Memory,” in IEEE International Symposium on Circuits and Systems, 2012
- [30]. Charles Manning. (2012). “Yaffs NAND Flash Failure Mitigation,” <http://www.yaffs.net/sites/yaffs.net/files/YaffsNandFailureMitigation>
- [31]. Zhiwei Qin. (n.d.). “Real-Time Flash Translation Layer for NAND Flash Memory Storage Systems”, in IEEE Transactions on Multi-Scale Computing Systems, vol.2 issue 1
- [32]. Rachana Shelat, K. T., “Analyze the Power Consumption of NAND Flash Memory” in International Journal of Engineering Research and Applications, 2013
- [33]. Duhee Lee, Chang-Gun Lee, and Kanghee Kim, “Real-time code

execution on NAND flash memory supporting wide-spectrum of input data for multiple tasks” in International Workshop on Software Support for Portable Storage, France, Oct. 2009

[34]. M. Joseph and P. Pandya, “Finding response times in a real-time system” in BCS Computer Journal, 29(5), 1986

[35]. A. Bastoni, B. Brandenburg, and J. Anderson, “Cache-related preemption and migration delays: Empirical approximation and impact on schedulability” in Proceedings of Sixth International Workshop on Operating Systems Platforms for Embedded Real-Time Applications, 2010

[36]. RobertN. Charette, “This car runs code” in IEEE Spectrum, Feb. <http://spectrum.ieee.org/transportation/systems/this-car-runs-on-code>

[37]. T. Kalibera, F. Pizlo, A. L. Hosking and J. Vitek, "Scheduling Hard Real-Time Garbage Collection," 2009 30th IEEE Real-Time Systems Symposium, Washington, DC, 2009, pp. 81-92.