A Bad-Block Test Design for Multiple Flash-Memory Chips

CHIN-HSIEN WU
Department of Electronic Engineering
National Taiwan University of Science and Technology
Taipei, 106 Taiwan
E-mail: chwu@mail.ntust.edu.tw

Flash-memory has become a popular alternative in storage systems, due to its characteristics in non-volatility, shock-resistance, and low-power consumption. When a flash-memory chip is created prior to shipping, its initial bad blocks should be identified, where a bad block means that data stored in the block could be unreliable and cannot be used. For safety and reliable reasons, any bad blocks should be efficiently identified by considering the operational model of flash-memory chips. In this paper, we will propose a bad-block test design for multiple flash-memory chips to efficiently identify bad blocks. The objective is to exploit execution parallelism and provide QoS guarantees. We will define a real-time task model and provide the schedulability analysis. We further provide a reasonable execution time setup for real-time tasks and show that the execution time setup can provide a schedulable result. In the experiments, a real case is discussed and measured such that the bad-block test design can be realized by configuring the corresponding computation time, the period, and the longest non-preemption time.

Keywords: design for testability, embedded systems, hardware/software co-design, flash memory, real-time systems

1. INTRODUCTION

Flash-memory has become a popular alternative in storage systems, due to its characteristics in non-volatility, shock-resistance, and low-power consumption. A NAND flash-memory chip consists of multiple blocks, and each block is made up of a fixed number of pages. A block is the smallest unit that erase operations can be applied to, whilst read and write operations can be done in pages. Any updates to existing data on a page are only possible after an erase operation. In particular, there could exist bad blocks that are prone to read and write failures. For safety and reliable reasons, any bad blocks should be efficiently identified by considering the operational model of flash-memory chips. Such an observation motivates this research.

In this paper, we will propose a bad-block test design for multiple flash-memory chips to efficiently identify bad blocks. According to the operation model of flash-memory chips, there are two non-preemption situations: (1) The setup phase or the output phase of write or erase operations; (2) The reading of a page. Any tasks cannot be preempted and should perform entirely when they encounter the situations. However, when a task is coming into the busy phase, the flash-memory chip is working itself such that the task can be preempted by other (higher priority) tasks. It should be noted that a task can’t be preempted during read operations, because their busy phase is relatively short.

Received December 20, 2010; revised September 1, 2011; accepted November 11, 2011.
Communicated by Tyng-Luh Liu.

1091
The objective of the bad-block test design should exploit the execution parallelism of various phases (i.e., busy phases of write and erase operations) and provide QoS guarantees when multiple flash-memory chips are tested simultaneously. We will configure a real-time task model and prove that the real-time tasks can be scheduled for the testing of multiple flash-memory chips by our schedulability analysis. We further provide the reasonable execution time setup for the real-time tasks and show that the execution time setup can provide a schedulable result. We also show that the execution time can be adjusted in a range for varying related real-time tasks’ priorities. In the experiments, we have tested a NAND flash-memory chip (K9G8G08U0M) that is designed by SAMSUNG Corporation. According to the experimental results, the bad-block test design can be realized by configuring the corresponding computation time, the period, and the longest non-preemption time.

The rest of this paper is organized as follows: Section 2 provides an overview of flash-memory characteristics. Section 3 is the related work. Section 4 is the motivation. Section 5 presents the bad-block test design. Section 6 provides the experimental results of a real case. Section 7 is the conclusion.

2. FLASH-MEMORY CHARACTERISTICS

A NAND\(^1\) flash-memory chip consists of multiple blocks, and each block is made up of a fixed number of pages. A block is the smallest unit that erase operations can be applied to, whilst read and write operations can be done in pages. A page contains a user area and a spare area, where the user area is reserved for the storage of a logical block and the spare area stores ECC (Error Correction Code) and other house-keeping information such as the logical block address (LBA). Note that the LBA denotes the logical address of a read/write unit in the flash-memory. For example, a K9G8G08U0M NAND flash-memory chip, the size of the user area and the spare area of a page is 2KB and 64B, respectively. Each block consists of 128 pages and its size is 256KB + 8KB. Flash-memory is write-once, therefore we do not overwrite existing data on each update. Instead, data must be written to free space and the old versions of data are invalidated, or considered as dead. This update strategy is known as “out-place update”, meaning that any existing data on flash-memory can not be over-written (updated) unless it is first erased. The pages that store live data are called “valid (live) pages” and that ones containing dead data are referred to as “invalid (dead) pages”.

An operation on flash-memory chips (i.e., read, write, or erase operations) consists of three phases: the setup phase, the busy phase, and the output phase. As shown in Fig. 1, the control signals are used to control the logic to the functions of read, write, and erase operations. The data signals are used to transfer the required data such as the command to the command latches, the address to the address latches, and the data to/from the I/O buffer. For example, the setup phase of a write operation is used for the command, the address, and accompanying data to the command latches, the address latches, and the I/O buffer, respectively. The busy phase of a write operation is reserved for busy-waiting whilst the data is being flushed from the I/O buffer into the corresponding flash-memory array. The output phase of a write operation is used to check if the operation is complete.

\(^1\)There are two major types of flash-memory in the current market: NAND flash and NOR flash. NAND flash is particularly designed for data storage, and NOR flash is for EEPROM replacement. We will focus our discussions on NAND flash since it is more suited to the requirements of huge-capacity file and storage systems.
A BAD-BLOCK TEST DESIGN FOR MULTIPLE FLASH-MEMORY CHIPS

Fig. 1. A functional block diagram: K9G8G08U0M.

by reading the specific status register. A read operation is similar to that of a write, except that the data transfer and busy-waiting sequences are reversed. The phases of an erase operation are the same as those of a write, except that no data transfer to/from the I/O buffer. In particular, write and erase operations are time consuming, and most of time is spent in the busy phase. However, read operations are faster than write and erase operations, since the busy phase of read operations is relatively short.

After performing several write operations, the amount of free space on flash-memory may be quite low. Activities that consist of a series of read, write, and erase operations that intend on reclaiming free space would then begin. These activities are known as “garbage collection” and are considered as overhead in flash-memory management. The objective of garbage collection is to recycle dead pages scattered throughout blocks, so that they can become free pages after the dead data has been erased. In order to achieve the garbage collection and the out-place update strategy, a flash translation layer (FTL) is proposed to provide the block-device emulation for transparent access from file systems without any modifications to existing file-system implementations. To accomplish the out-place update strategy, FTL usually adopts a RAM-resident translation table to map a given LBA (logical block address) to its physical block address (PBA) where the valid data actually resides. The translation table is indexed by LBA, and each entry of the table contains the PBA of the corresponding LBA. To accomplish the garbage collection, the greedy policy [3] is usually adopted in FTL to handle garbage collection by choosing a block that has the smallest number of live pages to erase. Under the current technology, each flash-memory block has a limitation on the erase cycle count, e.g., 10000 times. A worn-out block could suffer from frequent write errors and is considered as a bad block. Generally speaking, bad blocks could happen in three cases: a read failure, a write failure and an erase failure. While a write or erase operation is executed during its output phase, the status register of the chip can be read to indicate whether the write or erase operation is complete or not. Sometimes a read failure could happen due to some bits error, but ECC (Error Correcting Code) can be used to detect and even correct some bits error. For safety reasons, these blocks that are prone to read failures, write failures, and erase failures should be considered as bad blocks.

3. RELATED WORK

In recent years, issues related to flash-memory management have drawn a lot of at-
tention. Previous research has been reported on performance enhancement, especially on garbage collection, system architecture designs, and flash translation layers [2-14]. In the paper, we will discuss the testing issue about bad blocks and common faults (disturbances) in flash-memory should be introduced. IEEE 1005 Standard Definitions and Characterization of Floating Gate Semiconductor Arrays [15] shows realistic fault models and proposes a March-like test algorithms for the fault models. For flash-memory, there are three fault types: read disturbance, write disturbance, and erase disturbance. A page with read disturbance could not maintain its data after several read operations. Therefore, ECC (Error Correction Code) is used to resolve the problem. A written page with write disturbance could cause other (neighboring) pages to be written or erased. In particular, if data are written to pages in a block with erase disturbance, the pages could remain unchanged. Besides, pages in a block with erase disturbance could result in a leakage current such that erroneous reads could occur. These disturbances could cause bad blocks such that data stored in bad blocks could be unreliable and can not be used. Excellent research results and implementations have been reported on flash-memory testing and diagnostics [16-20]. Their common objective is to design the fastest test pattern to find all faults as much as possible. Regardless of which method, its test pattern must include a sequence of erase, write, and read operations. For example, the length of the test pattern for each block in [20] is $2 \times ET + 2 \times np \times WT + 6 \times np \times RT$, where $ET$ denotes the erase time for a block, $WT$ denotes the write time for a page, $RT$ denotes the read time for a page, and $np$ denotes the number of pages in a block. However, the objective of the paper is not to design a test pattern or diagnostics. We focus our discussion on a bad-block test design to increase execution parallelism and provide QoS guarantees when multiple flash-memory chips are tested simultaneously. Our bad-block test design can cooperate with the test patterns in [16-20] by configuring a proper real-time task model.

4. MOTIVATION

When a flash-memory chip is created prior to shipping, its initial bad blocks should be identified, where a bad block means that data stored in the block could be unreliable and can not be used. All bad blocks should be located such that systems can mask out the bad blocks via address mapping. For example, Samsung company guarantees that all blocks in flash-memory chips will be erased to FFh prior to shipping, except for initial bad blocks. Samsung company also makes sure that the last byte of the last page of each initial bad block has non-FFh data. As a result, when a Samsung flash-memory chip is tested, a loop procedure can be used to check the last byte of the last page of all blocks for searching the initial bad blocks. However, one situation is that the initial bad blocks can be erased such that the last byte of the last page of the blocks could be FFh. Another situation is that some blocks might have been bad blocks but the last byte of the last page of the blocks doesn’t reveal non-FFh data. Therefore, whether flash-memory chip vendors or flash-memory storage vendors should need a sophisticated bad-block test to identify the possible bad blocks when they first time use the chips. When chips are identified and used in some emergency or real-time systems, bad blocks in the chips could occur and cause unreliability after a certain amount of operations such as reads, writes and erases. Therefore, a bad-block test mechanism might be combined with flash-memory
controllers and is executed in a regular period. Such an observation motivates this re-
search. We want to propose a bad-block test design for multiple flash-memory chips such
that bad blocks can be efficiently identified by considering the operational model of
flash-memory chips. A real-time task model and schedulability analysis should be de-
vided for the tasks that are testing flash-memory chips simultaneously.

5. A BAD-BLOCK TEST DESIGN

5.1 Overview

In the section, a bad-block test design is proposed for multiple flash-memory chips
to efficiently identify bad blocks. Since a flash-memory chip consists of multiple blocks,
each test unit in our design is a block. A test procedure for each test unit includes a se-
quence of erase, write, and read operations. For the sake of simplicity, assume that the
test procedure for each test unit is one block erase, all page writes in the block, and all
page reads in the block. The test procedure can be adjusted dynamically according to the
test pattern. During the test procedure, any failures can help the identification of bad
blocks. In section 5.2, the operation model of a flash-memory chip is introduced. Since
the operation model of a flash-memory chip consists of three phases: the setup phase, the
busy phase, and the output phase, a bad-block test design should exploit the execution
parallelism of various phases and provide QoS guarantees when multiple flash-memory
chips are tested simultaneously. In section 5.3, since a real-time task is created to test a
flash-memory chip, each real-time task is defined as as a triple \( <c, p, npt> \), where \( c \) de-
notes the computation time, \( p \) denotes the period, and \( npt \) denotes the longest time that
can not be preempted. \( npt \) is required for those real-time tasks that share non-preemption
resources. We show that there are two non-preemption situations: (1) The setup phase or
the output phase of write or erase operations; (2) The reading of a page. \( npt \) can be cal-
culated according to the non-preemption situations. In section 5.4, we propose a sched-
ulability analysis for the real-time task model by revising the basic priority ceiling pro-
tocol. We prove that the real-time tasks for the testing of multiple flash-memory chips
can be scheduled by the rate monotonic algorithm, where the non-preemption situations
have been considered. In section 5.5, we further provide the reasonable execution time
setup for the real-time tasks and show that the execution time setup can provide a sched-
ulable result. We also show that the execution time can be adjusted in a range for varying
related real-time tasks’ priorities.

5.2 An Operation Model of a Flash-Memory Chip

The operation model of a flash-memory chip generally consists of three phases: the
setup phase, the busy phase, and the output phase, as shown in Fig. 2. For example, the
operation model of a read operation consists of “Command”, “Address”, “Busy-waiting”,
and “Data Out”. The setup phase includes “Command” and “Address”, the busy phase
includes “Busy-waiting”, and the output phase includes “Data Out”. During the setup
phase of a read operation, the command and the address are written to the data registers
of flash-memory in order. The busy phase of a read operation is reserved for busy wait-
ing while the data is being read to appropriate registers of flash-memory. The busy phase of a read operation is relatively short due to the characteristics of flash-memory. The output phase of a read operation is to deliver data from registers of flash-memory to the host system. For the operation model of a write operation, the setup phase includes “Command”, “Address”, and “Data In”; the busy phase includes “Busy-waiting”, and the output phase includes “Status”. During the setup phase of a write operation, the command, the address, and accompanying data are written to the data registers of flash-memory in order. The busy phase of a write operation is reserved for busy waiting while the data is being flushed into the appropriate flash-memory cell from data registers. The output phase of a write operation is to detect if the write operation is complete. The operation model of an erase operation is the same as those of a write operation, except that no “Data In” occurs during the setup phase. The busy phase of an erase operation is to charge the block such that the block can be rewritten. Note that write and erase operations are time consuming, and most of time is spent in the busy phase.

### 5.3 A Real-Time Task Model

According to the operation model of a flash-memory chip when a real-time task is executed during one of two non-preemption situations: (1) The setup phase or the output phase of write or erase operations; (2) The reading of a page, and the task can not be preempted and should perform entirely. However, when the task is coming into the busy phase, the flash-memory chip is working itself such that the task can be preempted by other (higher priority) tasks. It should be noted that a task can’t be preempted during read operations, because their busy phase is too short to benefit the proposed method. For example, in the datasheet of Samsung K9F6408U0A NAND flash memory, the busy phase of reading of a page is about 25us, but we have measured that the overhead of suspension and resumption time for a real-time task was about 30-50us in a Linux system (i.e., CPU is AMD-Duron 750MHZ and DRAM is 320MB) [12].

A real-time task model is defined as follows: assume that a system has $n$ real-time tasks ($\tau_1 \sim \tau_n$) for testing $n$ flash-memory chips and one processor (i.e., a testing controller)
to execute the real-time tasks. Each real-time task $\tau_i$ is defined as a triple $\langle c_i, p_i, npt_i \rangle$, where $c_i$ denotes the computation time, $p_i$ denotes the period, and $npt_i$ denotes the longest time for the setup or output phases that cannot be preempted. As a result, $c_i$ should be greater than or equal to $npt_i$. If $npt_i$ is 0, it means that the corresponding real-time task is not for the testing of a flash-memory chip. When $\tau_i$ is ready to test a specific flash-memory chip, we should measure the reading time of a page, the setup phase time for write and erase operations, and the output phase time for write and erase operations. $npt_i$ is the maximum one that could be among the reading time of a page, the setup phase time, or the output phase time. According to the experimental results, the variance of the access time is negligible such that the setup phase time and the output phase time is highly predictable. As a result, $npt_i$ can be quickly calculated by accessing a few pages and blocks. The pseudo code of calculating $npt_i$ is shown in Algorithm 1. Assume that $R_i^{setup}$, $R_i^{busy}$, and $R_i^{output}$ denote the setup phase time, the busy phase time, and the output phase time for a read operation, respectively. $W_i^{setup}$, $W_i^{busy}$, $W_i^{output}$, $E_i^{setup}$, $E_i^{busy}$, and $E_i^{output}$ have the similar definitions for write and erase operations. Let $RT_i$, $WT_i$, and $ET_i$ denote $(R_i^{setup} + R_i^{busy} + R_i^{output})$, $W_i^{setup} + W_i^{busy} + W_i^{output}$, and $E_i^{setup} + E_i^{busy} + E_i^{output}$, respectively.

Assume that there are total execution time $T$ that can be used and $\tau_i$ is to test a flash-memory chip that consists of $B_i$ blocks, $p_i$ can be defined as $T/B_i$. Since each test unit includes one block erase, all page writes in the block, and all page reads in the block, $T/B_i$ should be greater than $((ET_i + npt_i \times (WT_i + RT_i))$, where $np$ is the number of pages in a block. Furthermore, $c_i$ at least is defined as $((E_i^{setup} + E_i^{output}) + npt_i \times ((W_i^{setup} + W_i^{output}) + RT_i))$. As a result, each real-time task $\tau_i$ for testing a flash-memory chip is defined as a triple $\langle c_i = ((E_i^{setup} + E_i^{output}) + npt_i \times ((W_i^{setup} + W_i^{output}) + RT_i)), p_i = T/B_i, npt_i \rangle$.

### 5.4 A Schedulability Analysis

A higher priority real-time task could be blocked by a lower priority real-time task when the lower priority task is testing a flash-memory chip and can not be preempted. This is the well-known priority inversion problem [22]. We propose to revise the basic priority ceiling protocol [22] to solve the problem. In the basic priority ceiling protocol, a semaphore is used to guard a critical section and the priority ceiling of the semaphore is the priority of the highest priority task may lock the semaphore. The basic priority ceiling protocol also includes the basic priority inheritance protocol: if a lower priority real-time task blocks a higher priority real-time task, the lower priority task will inherit the priority of the higher task and the priority inheritance is transitive. Under the basic priority ceiling protocol, a task $\tau$ can lock a semaphore if the semaphore is available and the priority

---

2 A higher priority real-task must wait for the execution of a lower priority real-time task.
of \( \tau \) is higher than the highest priority ceiling of all semaphores that currently are locked by tasks except for \( \tau \). Since the non-preemption situations for the testing of flash-memory chips are one kind of critical section, a semaphore is used to protect the non-preemption situations such that a resource management (e.g., the basic priority ceiling protocol) is required. Assume that \( \text{Pri}(\cdot) \) denotes the priority of \( \tau \), \( \text{Priority}_\text{Ceiling}(s) \) denotes the priority ceiling of a semaphore \( s \), and \( \text{Max} \) is a maximum function. We will show the related properties, lemmas, and theorems in the following:

**Lemma 5.4.1** A real-time task \( \tau_i \) can be blocked by a lower priority real-time task \( \tau_j \) only if \( \tau_j \) is testing a flash-memory chip during one of two non-preemption situations when \( \tau_i \) arrives.

**Proof:** Similar proof can be found in [22].

**Lemma 5.4.2** No real-time task can be blocked for more than one critical section of any lower priority real-time task that is testing a flash-memory chip.

**Proof:** Similar proof can be found in [22].

**Proposition 5.4.3** Let \( S_j \) be the set of semaphores that \( \tau_j \) may lock, where \( 1 \leq j \leq n \) and \( j \neq i \).

**Proposition 5.4.4** \( M_i = \{ \tau_j | \text{Pri}(\tau_j) > \text{Pri}(\tau_i) \text{ and } \text{Max}_{\tau_j \in S}(\text{Priority}_\text{Ceiling}(s)) \geq \text{Pri}(\tau_j) \} \).

**Lemma 5.4.5** The worst-case blocking time for \( \tau_i \) is \( \text{Max}_{\tau_j \in M}(\text{npt}_j) \).

**Proof:** According to Lemmas 5.4.1 and 5.4.2, \( \tau_i \) can not be blocked for more than one critical section of any lower priority real-time task that is testing a flash-memory chip. In addition, Propositions 5.4.3 and 5.4.4 suggest that \( \tau_i \) may be blocked by \( \tau_j \) in \( M_i \) such that the worst-case blocking time must be the largest \( \text{npt}_j \).

**Theorem 5.4.6** Let a set of \( n \) real-time tasks \( (\tau_1~\tau_n) \) be sorted in rate monotonic order. Under the revised priority ceiling protocol, the \( i \)th task can be scheduled by the rate monotonic algorithm if \( \forall i, 1 \leq i \leq n, \sum_{j=1}^{i-1} \frac{c_j}{p_j} + \frac{\text{Max}_{\tau_j \in M_i}(\text{npt}_j)}{p_j} \leq i(2^{1/i} - 1) \).

**Proof:** Since \( \text{Max}_{\tau_j \in M}(\text{npt}_j) \) is the worst-case blocking time for \( \tau_i \), it can be considered as an additional computation requirement for \( \tau_i \).

**5.5 An Execution Time Analysis**

When a real-time task \( \tau_i \) is created to test a flash-memory chip that consists of \( B_i \) blocks, its period can be defined as \( T/B_i \), where \( T \) denotes the available execution time. According to the schedulability analysis, \( T \) can be derived such that all real-time tasks can be scheduled. Assume that there are \( n \) real-time tasks \( (\tau_1~\tau_n) \) for testing \( n \) flash-memory chips, the execution time for \( n \) flash-memory chips can be set as \( T = \text{Max}_{i=1,1\leq i \leq n} \).
A BAD-BLOCK TEST DESIGN FOR MULTIPLE FLASH-MEMORY CHIPS

899

After the execution time \( T \) is determined, the periods of all real-time tasks can be calculated. The configuration of real-time tasks is shown in Algorithm 2 under the rate monotonic algorithm.

**Algorithm 2**  Configuration of real-time tasks

1: Let \( T \) be \( \max_{1 \leq i \leq n} ((\sum_{j=1}^{i-1} (c_j \cdot B_j) + (c_i + \max_{r \neq i} (npt_r)) \cdot B_i) / i(2^{1/i} - 1)) \).

2: Let a real-time task \( \tau_i \) be defined as \( <c_i = ((E_{\text{setup}} + E_{\text{output}}) + np*(W_{\text{setup}} + W_{\text{output}}) + RT)_i, p_i = T/B_i, npt_i> \).

3: Sort these real-time tasks (\( \tau_1 \sim \tau_n \)) in a rate monotonic order (i.e., the smallest period has the largest priority).

**Theorem 5.5.1**  When \( T \) is set as \( \max_{1 \leq i \leq n} ((\sum_{j=1}^{i-1} (c_j \cdot B_j) + (c_i + \max_{r \neq i} (npt_r)) \cdot B_i) / i(2^{1/i} - 1)) \), the real-time tasks are scheduled by the rate monotonic algorithm.

**Proof:** According to Theorem 5.4.6, if \( \forall i, 1 \leq i \leq n, \sum_{j=1}^{i-1} \frac{c_j + \max_{r \neq i} (npt_r)}{p_j} \leq i(2^{1/i} - 1) \), the \( i \)th task can be scheduled. Let \( p_i \) and \( p_j \) be \( T/B_i \) and \( T/B_j \), respectively. The above inequality can be written as follows:

\[
\sum_{j=1}^{i-1} \frac{c_j + \max_{r \neq i} (npt_r)}{p_j} \leq i(2^{1/i} - 1)
\]

\[
\Rightarrow \sum_{j=1}^{i-1} (c_j \cdot B_j) + (c_i + \max_{r \neq i} (npt_r)) \cdot B_i \leq T \cdot i(2^{1/i} - 1)
\]

\[
\Rightarrow (\sum_{j=1}^{i-1} (c_j \cdot B_j) + (c_i + \max_{r \neq i} (npt_r)) \cdot B_i) / i(2^{1/i} - 1) \leq T.
\]

As a result, when \( T \) is set as \( \max_{1 \leq i \leq n} ((\sum_{j=1}^{i-1} (c_j \cdot B_j) + (c_i + \max_{r \neq i} (npt_r)) \cdot B_i) / i(2^{1/i} - 1)) \), the real-time tasks (\( \tau_1 \sim \tau_n \)) are schedulable by the rate monotonic algorithm.

Since there are different types of flash-memory chips, they may have different size and access performance. Every real-time task (i.e., \( \tau_i \)) for testing a flash-memory chip may have different configuration (i.e., \( <c_i, p_i, npt_i> \)). When \( T \) is derived, \( T \) denotes the execution time for the testing of flash-memory chips under the rate monotonic algorithm. In a real-time system, \( T \) can be increased to decrease the priorities of real-time tasks for the testing of flash-memory chips. Similarly, \( T \) can be decreased to increase the priorities of real-time tasks and the schedulability analysis in Theorem 5.4.6 can be used for any specific \( T \). However, \( T \) must have a minimum execution time.

**Lemma 5.5.2**  When \( n \) flash-memory chips are tested, the minimum execution time for \( T \) is \( \sum_{i=1}^{n} (B_i \cdot c_i) \).
Proof: Since $c_i$ is the required execution time for testing a block, $\sum_{i=1}^{n} (B_i \cdot c_i)$ will be the required execution time for testing $n$ flash-memory chips. As a result, $T$ cannot be smaller than it.

Clearly, when $T$ will be decreased to a value, the value must be between $\max_{i=1, \ldots, n} ((\sum_{j=1}^{i} c_j \cdot B_j) + (c_j + \max_{i=1}^{\min_{k}} (npt_k)) \cdot B_j) / i(2^{i-1} - 1)$ and $\sum_{i=1}^{n} (B_i \cdot c_i)$.

<table>
<thead>
<tr>
<th>Table 1. Features of a Samsung K9G8G08U0M NAND flash-memory chip.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Voltage Supply</td>
</tr>
<tr>
<td>Memory Cell Array</td>
</tr>
<tr>
<td>Memory Cell</td>
</tr>
<tr>
<td>Number of Blocks</td>
</tr>
<tr>
<td>Number of Pages in a Block</td>
</tr>
<tr>
<td>Data Register</td>
</tr>
<tr>
<td>Page Program Time (2K + 64)bytes</td>
</tr>
<tr>
<td>Page Read Time (2K + 64)bytes</td>
</tr>
<tr>
<td>Block Erase Time (256K + 8K)bytes</td>
</tr>
</tbody>
</table>

6. A REAL CASE

We have tested a NAND flash-memory chip (K9G8G08U0M) that is designed by SAMSUNG corporation. We list the features of the chip in Table 1 according to its data-sheet. The flash-memory chip is a kind of MLC (multi-level cell) flash-memory and each memory cell denotes two bits such that its memory size can be large and its cost can be cheap. SLC (single-level cell) flash-memory is another kind of flash-memory and each memory cell can only denote one bit. SLC flash-memory has faster access speed than MLC flash-memory but its cost is more expansive than MLC flash-memory. In the experiments, we used an AVR ATmega16 controller to test one flash-memory chip and also measured the average time of the setup phase, the busy phase, and the output phase for the reading of a page, the writing of a page, and the erasing of a block, respectively. AVR ATmega16 was a kind of 8-bit controller and its execution speed was 16MHz in the experiments. The experimental architecture is shown in Fig. 3, where a LCD module was also used to display the experimental results. Note that in our previous paper [12], we have evaluated the effectiveness of exploiting the execution parallelism when accessing multiple flash memory chips. We demonstrated that the execution parallelism under multiple flash-memory chips can enable more CPU cycles to be used to execute other tasks. That is, some tasks can obtain more CPU cycles by the execution parallelism because CPU is released during the busy phases of write and erase operations.

Since each test unit in our design is a block, our test procedure for each test unit is one block erase, all page writes in the block, and all page reads in the block. The average time of the setup phase, the busy phase, and the output phase for the reading of a page, the writing of a page, and the erasing of a block is shown in Fig. 4. We can observe that the busy phase for write or erase operations was relatively long and the AVR ATmega16 controller was idle during these phases. Assume that multiple flash-memory chips are
A BAD-BLOCK TEST DESIGN FOR MULTIPLE FLASH-MEMORY CHIPS

Fig. 3. Experimental architecture.

![Experimental architecture diagram]

Fig. 4. Experimental results.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Setup Phase</th>
<th>Busy Phase</th>
<th>Output Phase</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reading of a page</td>
<td>8us</td>
<td>59.5us</td>
<td>40.5us</td>
</tr>
<tr>
<td>Writing of a page</td>
<td>6.5us+64.5us</td>
<td>993us</td>
<td>3us</td>
</tr>
<tr>
<td>Erasing of a block</td>
<td>6.5us</td>
<td>1,233us</td>
<td>4us</td>
</tr>
</tbody>
</table>

tested simultaneously, when one flash-memory chip is coming into its busy phase for write or erase operations, the AVR ATmega16 controller can test another chip to increase the test parallelism. Since the clock frequency of the AVR ATmega16 controller was 16MHZ and it was not a fast processor, many execution time will be needed when many instruction cycles were executed for the I/O transfer. For example, when (2k + 64) bytes were transferred during “Data Out” for read operations and “Data In” for write operations, the number of instruction cycles was 2,112 and the execution time was about 40–60us! This was because the AVR ATmega16 controller will be busy-waiting during the “Data Out” for read operations and “Data In” for write operations such that the test parallelism will not be favored.

Assume that a real-time task \( \tau_i \) will test the flash-memory chip, \( \tau_i \) can be defined as a tuple \(<c_i> = ((E_{setup} + E_{output}) + np^i((W_{setup} + W_{output}) + RT_i)), p_i = T/B_i, npt_i>\), where \( npt_i \) is the maximum of \( W_{setup}, W_{output}, RT_i, E_{setup}, \) and \( E_{output} \). According to Fig. 4, \( \tau_i \) is defined as \(<c_i> = 9,583.8us, p_i = T/4,096, npt_i = 108us>\). Note that the setup of \( T \) can be seen in section 5.5.
7. CONCLUSION

When a flash-memory chip is created prior to shipping, its initial bad blocks should be identified, where a bad block means that data stored in the block could be unreliable and can not be used. For safety reasons, any bad blocks should be efficiently identified by considering the operational model of flash-memory chips. As a result, we propose a bad-block test design for multiple flash-memory chips to efficiently identify bad blocks. The bad-block test design should exploit the execution parallelism of various phases (i.e., busy phases of write and erase operations) and provide QoS guarantees when multiple flash-memory chips are tested simultaneously. The major contributions of this paper are summarized as follows:

- We introduce the operation model of a flash-memory chip. When a real-time task is executed during one of two non-preemption situations: (1) The setup phase or the output phase of write or erase operations; (2) The reading of a page, the task can not be preempted and should perform entirely. However, when the task is coming into the busy phase of write or erased operations, the flash-memory chip is working itself such that the task can be preempted by other higher priority tasks.
- We propose a real-time task model for the testing of multiple flash-memory chips. Each real-time task $\tau_i$ is defined as a triple $<c_i, p_i, npt_i>$, where $c_i$ denotes the computation time, $p_i$ denotes the period, and $npt_i$ denotes the longest time for the setup or output phases that can not be preempted.
- We propose a schedulability analysis for the real-time task model by revising the basic priority ceiling protocol. We prove that real-time tasks can be scheduled by the rate monotonic algorithm, where the non-preemption situations have been considered.
- We provide a reasonable execution time setup for the real-time tasks and show that the execution time setup can provide a schedulable result. We also show that the execution time can be adjusted in a range for varying related real-time tasks’ priorities.
- In the experiments, we have tested a NAND flash-memory chip (K9G8G08U0M) that is designed by SAMSUNG corporation. According to the experimental results, the configuration $<c_i, p_i, npt_i>$ of a real-time task can be realized.

Future research should involve further examination of the characteristics of flash-memory, especially when different flash-memory chips are tested. With a more considered approach incorporating application designs and flash-memory characteristics, a real prototype will be designed and manufactured. The real prototype can provide a fast test and analysis of multiple flash-memory chips.

REFERENCES


Chin-Hsien Wu (吳晉賢) received his B.S. degree in computer science from National Chung Cheng University in 1999. He received his M.S. and Ph.D. degree in computer science from National Taiwan University in 2001 and 2006, respectively. Now, he is an Associate Professor at the Department of Electronic Engineering in National Taiwan University of Science and Technology. He is also a member of ACM and IEEE. His research interests include embedded systems, real-time systems, ubiquitous computing, and flash-memory storage systems.