國立臺灣科技大學
電子工程系

碩士學位論文

學號 M9902138

一個基於固態硬碟之可平行化的多控制器設計

PMCD: A Parallel Multi-Controller Design for Solid-State Drives

研 究 生：周茂儀
指導教授：吳晉賢 博士

中華民國 一百零二 年一月三十日

碩士學位論文指導教授推薦書

本校　電子工程系　　周茂儀(CHOU, MAW-YI)　君

所提之論文：

一個基於固態硬碟之可平行化的多控制器設計

係由本人指導撰述，同意提付審查。

指導教授：吳晉賢

指導教授 吳晉賢

102 年 1 月 29 日

碩士學位考試委員審定書

M9902138

指導教授：吳晉賢

本校　電子工程系　　周茂儀　君

所提之論文：

一個基於固態硬碟之可平行化的多控制器設計

經本委員會審定通過，特此證明。

學校考試委員會

委　　員：許孟超

陳維美

陳秋男

林昌鴻

吳晉賢

指導教授：吳晉賢

學程主任：

系(學程)主任、所長：方文賢

中華民國 102 年 1 月 29 日

一個基於固態硬碟之可平行化的多控制器設計

研 究 生 ： 周茂儀
指 導 老 師 ： 吳 晉 賢
時 間：102 年 01 月

中文摘要

NAND 型快閃記憶體已經應用在許多消費性電子產品及嵌入式系統中，因為它有非揮發、低功率、存取速度快和抗震等等的優點。 近幾十年來，以 NAND 型快閃記憶體所構成 SSDs(固態硬碟)已應用在許多方面，取代原本的 HDDs (傳統硬碟)。 然而一些 NAND 型快閃記憶體的特性(如非對稱的存取時間及存取基本單位、out-place update 及有限的清除次數)會對效能產生影響。 目前在 SSD 的內部架構中，須要多個控制器來掌控和管理 NAND 型快閃記憶體晶片。 一個控制器只能存取其專屬匯流排上的 NAND 型快閃記憶體晶片，而無法存取其他控制器所屬的 NAND 型快閃記憶體晶片。 我們把這種情形稱為匯流排限制。 當多個請求須存取同一條匯流排上的多個 NAND 型快閃記憶體晶片時，匯流排限制會減低控制器處理的平行度。在本論文中我們提出一個可平行化的多控制器設計(PMCD)來解決這個問題。 我們會把 PMCD 和傳統的多控制器設計(TMCD)實作在 FPGA 的開發板上(Altera DE2)。 根據我們的實驗結果，比起 TMCD，PMCD 增加了 27.3%的效能，而且 overhead 也在合理的範圍內。

關鍵字：NAND 型快閃記憶體、固態硬碟、控制器

Abstract

NAND flash memory has advanced along with the wave of consumer electronics and embedded systems, because of its advantages of non-volatility, lower-power consumption, faster access, and shock-resistance. In recent decades, SSDs (Solid-State Drives) which use NAND flash memory have replaced HDDs (Hard-Disk Drives) in many applications. However, NAND flash memory that has special characteristics (e.g., unsymmetrical operations, out-place updates and, erase limitation) could have performance impacts on SSDs. Currently, the internal architecture of a SSD needs multiple controllers to handle and manage NAND flash chips. However, one controller is in charge of some specific NAND flash chips on its own bus and can't access other NAND flash chips that owned by other controllers. We call the situation as the bus constraint. The bus constraint will reduce the controllers' execution parallelism when multiple requests will access those chips on the same bus. To overcome this problem, we will propose a parallel multi-controller design (PMCD) in this thesis. We will implement PMCD and the traditional multi-controller design (TMCD) on an FPGA-based development board (i.e., Altera DE2). According to our experimental results, PMCD can increase about 27.3% performance when compared to TMCD, and the overhead is acceptable.

Keywords: NAND flash memory, Solid-State Drives (SSD) , Controller

Index

Index of Figures

Index of tables

Chapter 1 Introduction

SSDs (Solid-State Drives) are a popular alternative to replace HDDs (Hard-Disk Drives) in some applications. Many kinds of non-volatile memories can be used in SSDs, such as NAND flash memory [1], NOR flash memory, PCM (Phase-Change Memory), STTM (Spin-Torque Transfer Memory) and so on. Today, SSDs mostly use NAND flash memory because of the advantages of non-volatility, lower-power consumption, faster access, and shock-resistance. However, NAND flash memory has special characteristics such as (1) unsymmetrical operations, (2) out-place updates and, (3)erase limitation. Reads and writes on NAND flash memory are unsymmetrical operations. Reads on NAND flash memory have faster access speed and less energy consumption than writes on NAND flash memory. Out-place updates mean that updates to existing data on a page are only possible after an erase operation. A read or write unit is a page (e.g., 512KB [2] or 4KB [3]) but an erase unit is a block (e.g., 16 or 32 pages). Erase limitation means that each block in NAND flash memory has a limited number of erasure cycles, e.g., 1,000~10,000 times. A worn-out block could suffer from frequent write errors or read errors. Therefore, the management of SSDs must consider the characteristics of NAND flash memory.

In nowadays, controllers connect to NAND flash chips in a static way on traditional SSDs, which means that a controller will access specific NAND flash chips on its own bus [23],[24], [25], [26], [27], [28], [29], [30], [32], [31]. A controller can't access those NAND flash chips that don't belong to it. We call the situation as the bus constraint. Simply speaking, it is forbidden to access more than one chip on the same bus concurrently. The bus constraint will reduce the controllers' execution parallelism and reduce the performance harmfully. In the thesis, we will propose a parallel multi-controller design (PMCD) for solid-state drives. Any NAND flash chip will not be dominated by a specific controller. We will propose a topology interface between controllers and NAND flash chips to achieve this goal. We will also show how controllers communicate with each other and how requests can access NAND flash chips simultaneously without the bus constraint. Both the proposed design and the traditional design will be implemented on an FPGA-based development board (i.e, Altera DE2) and we can compare the two designs on the access time under some extreme cases, hardware resource, and power consumption. The experimental results shows that the proposed design has about 27.3% decreases on the access time, a slight increase on the usage of hardware resource and almost the similar power

consumption, when compared to the traditional design.

The rest of this thesis is organized as follows: Chapter 2 shows a parallel multi-controller design for solid-state drives. The experiments are presented in Chapter 3. Chapter 4 is the conclusion.

Chapter 2 A PARALLEL MULTI-CONTROLLER DESIGN FOR SOLID-STATE DRIVES

2.1 Overview

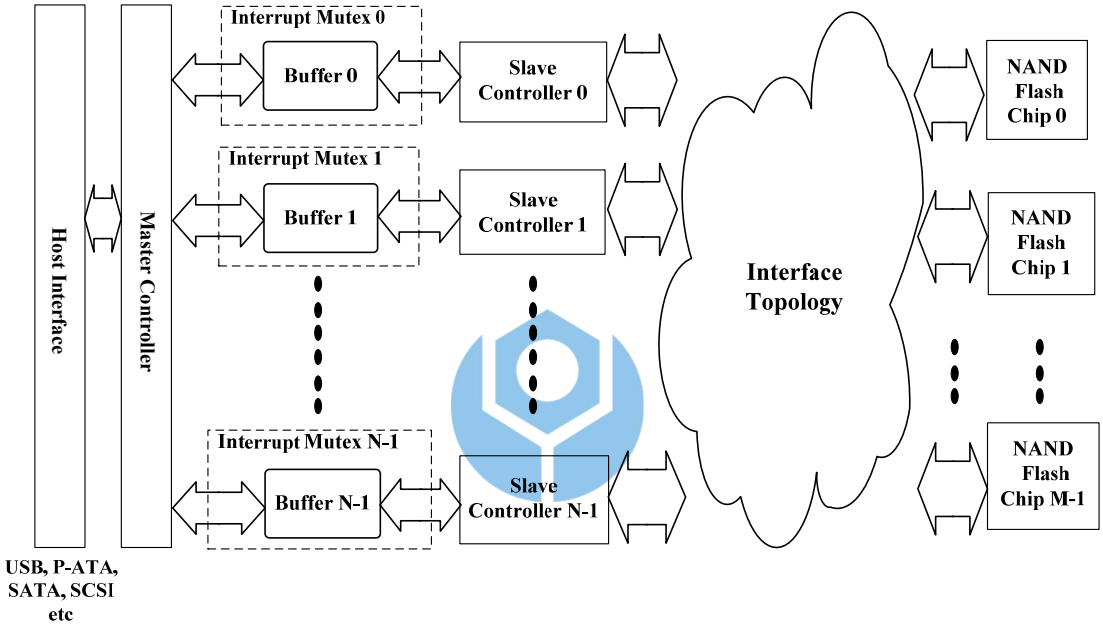


Fig. 1: the Concept of a Multi-Controller Design for Solid-State Drives

We will propose a Parallel Multi-Controller Design for solid-state drives, referred to as PMCD, as shown in Fig.2. The master controller is responsible for receiving requests from the host side (i.e., the USB Bus in our design), assigning tasks to the slave controllers and maintaining the occupation of the NAND flash chips. The slave controllers engage in accessing the NAND flash chips. For communication between the master controller and the slave controllers, we will propose an Interrupt Mutex module for each master-slave pair. The interrupt mutex module can store data and exchange message and its function is similar to the shared-memory multi-processor architecture [33] or POSIX threads (pthread) programming [34]. Note that the master controller does not perform any real operations (e.g., read, program, and erase) on the NAND flash chips and only the slave controllers will handle the real operations. Therefore, the master controller can be fast to handle a lot of requests. We

also propose a N x M router interface between the slave controllers and NAND flash chips, where N denotes the number of slave controllers and M denotes the number of NAND flash chips. The purpose of the N x M router interface is to provide multiple paths from slave controllers to different NAND flash chips simultaneously. In the thesis, we assume that N is 4 and M is 16.

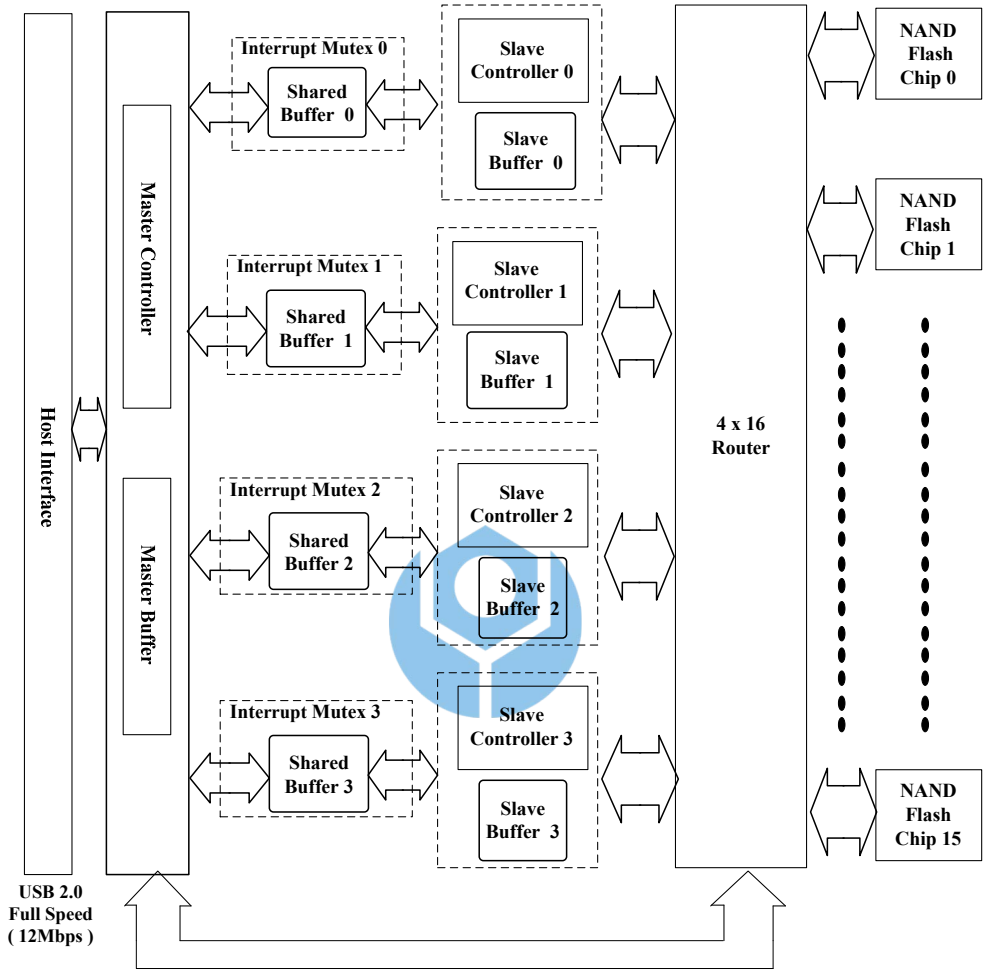


Fig. 2: PMCD: A parallel multi-controller design for solid-state drives.

2.2 A Parallel Multi-Controller Design

Assume that there are one master controller, 4 slave controllers, 16 NAND flash chips, and one 4x16 router in the design, as shown in Fig. 2. We will implement the design on DE2 [4] and describe the interrupt mutex module and the NxM router interface in the following.

2.2.1 An Interrupt Mutex Module

In the shared-memory multiprocessor architecture [33], many processors have dedicated memories and share a public memory that provides the communication among processors. Since each processor can access the public memory, the data consistence in the public memory must be considered. To maintain the data consistence, there should be a shield to protect the shared memory and also a passport into this protected area if someone will access the public memory. In the pthread programming [34], the Mutex plays a role as the shield. If someone will access the critical area (e.g., the shared memory), the Mutex must be locked first. Only one can lock the Mutex at the same time and it is lockable again until it is unlocked. With the same concept, we use an interrupt mutex module to guard the individual shared buffer for each master-slave pair. Each shared buffer contains the data written to the NAND flash chips, the data transmitted to the host side, and the exchanging messages between the master controller and the relative slave controller. In practice, a dead-lock situation may occurs. That is the master controller wants to transfer data into the shared buffer, but it can't lock the corresponding interrupt mutex for a long waiting time. The reason is that the relative slave controller locks the interrupt mutex and is waiting for the master controller to put data in the shared buffer. To solve the problem, we add the "Interrupt" feature and some control pins in the interrupt mutex module, as shown in Fig. 3. With the "Interrupt" feature, the master controller can send an interrupt signal to the relative slave controller for the notification of a new task transmitted by the master controller. The slave controller can also send an interrupt signal to the master controller to inform the execution status of the task.

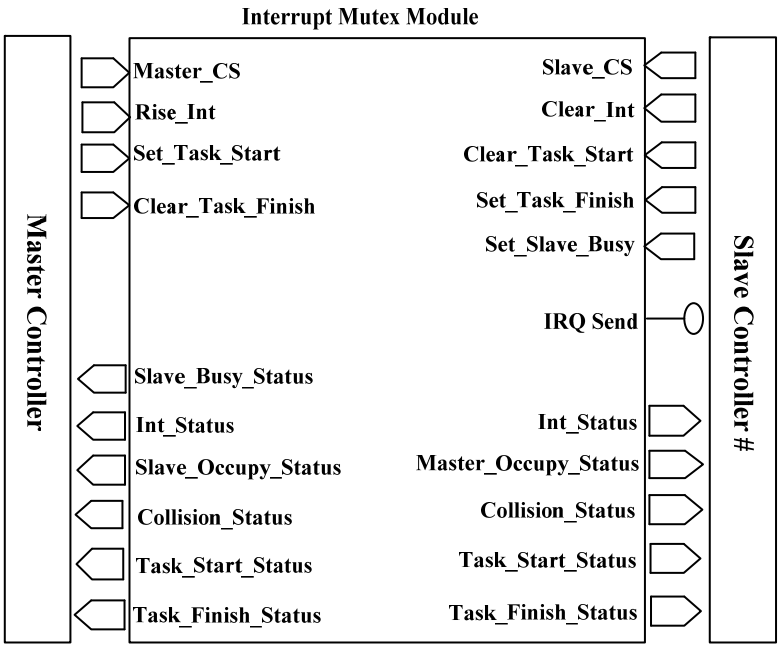


Fig. 3: An interrupt mutex module.

The signals that are writable in Fig. 3 including Master_CS, Rise_Int, Set_Task_Start, Clear_Task_Finish, Slave_CS, Clear_Int, Clear_Task_Start, Set_Task_Finish, Set_Slave_Busy. The signals that are read-only status include Int_Status, Collision_Status, Task_Start_Status, Task_Finish_Status, Slave_Occupy_Status, Slave_Busy_Status, and Master_Occupy_Status. Both the master controller and the slave controllers can grab the value of the read-only signals at any time without locking the corresponding interrupt mutex. If the master controller or the slave controller will access the shared buffer in an interrupt mutex, they need to confirm whether the corresponding interrupt mutex is locked by checking Slave_Occupy_Status or Master_Occupy_Status. If Slave_Occupy_Status is enable, it means that the interrupt mutex is locked by the slave controller. On the other hand, if Master_Occupy_Status is enable, it means that the interrupt mutex is locked by the master controller. If Master_CS or Slave_CS are enable, it means that the master controller or the relative slave controller wants to lock the interrupt mutex. If Collision_Status is enable, it means that both the master controller and the relative slave controller want to lock the interrupt mutex at the same time. To solve the problem, either the master controller or the relative slave controller should give up the locking and set Master_CS or Slave_CS unable. If the master controller successfully locks the interrupt mutex, it can set Rise_Int and the relative slave controller will receive an interrupt signal from IRQ_Send. Note that the master controller can't set Rise_Int, when Slave_Busy_Status is enable. This is because the relative slave

controller is busy and can't accept any signal. The master controller can also set Set_Task_Start (i.e., Task_Start_Status will be enable) to inform the relative slave controller of messages about new tasks in the shared buffer.

2.2.2 N x M Router

A N x M router is between N slave controllers and M NAND flash chips. As shown in Fig. 4, there are 4 inputs (i.e., X[0]~X[3]) connecting to 4 slave controllers, 16 outputs (i.e., Y[0] ~Y[15]) connecting to 16 NAND flash chips, 4 sets of control signals (i.e., C[0] ~C[3])attaching to the master controller to set up 4 relative slave controllers, and 1 set of information pins to provide the current status of the router (e.g., a path from X[i] to Y[j]). The N x M router not only provides multiple paths from slave controllers to any NAND flash chips but also prevents a NAND flash chip from being occupied by two different slave controllers at the same time. For the purpose, the master controller needs to set the controlling signals of the router properly through C[0]~C[3], as shown in Fig. 4. When the master controller wants to assign a task to a slave controller, it gets the information about chips occupation first to check the status of the required NAND flash chip. If the required NAND flash chip is free, the master controller will allow the dedicated slave controller to access the required NAND flash chip by setting the relative controlling signals (e.g., C[0]~C[3]). Since the NxM router can provide multiple paths from slave controllers to any NAND flash chips, it means that the NxM router interface can let multiple slave controllers access to NAND flash chips simultaneously if the NAND flash chips are mutually exclusive. For example, assume that the master controller will assign 4 individual tasks (task 0~task 3) to 4 idle slave controllers (Slave Controller 0~Slave Controller 3), as shown in Fig. 5. First, the master controller sets a control signal C[2] to indicate a path from X[2] (i.e., Slave Controller 2) to Y[12] (i.e., NAND flash chip 12) for task 0. Next, it sets a control signal C[0] to indicate a path from X[0] to Y[9] and so on.

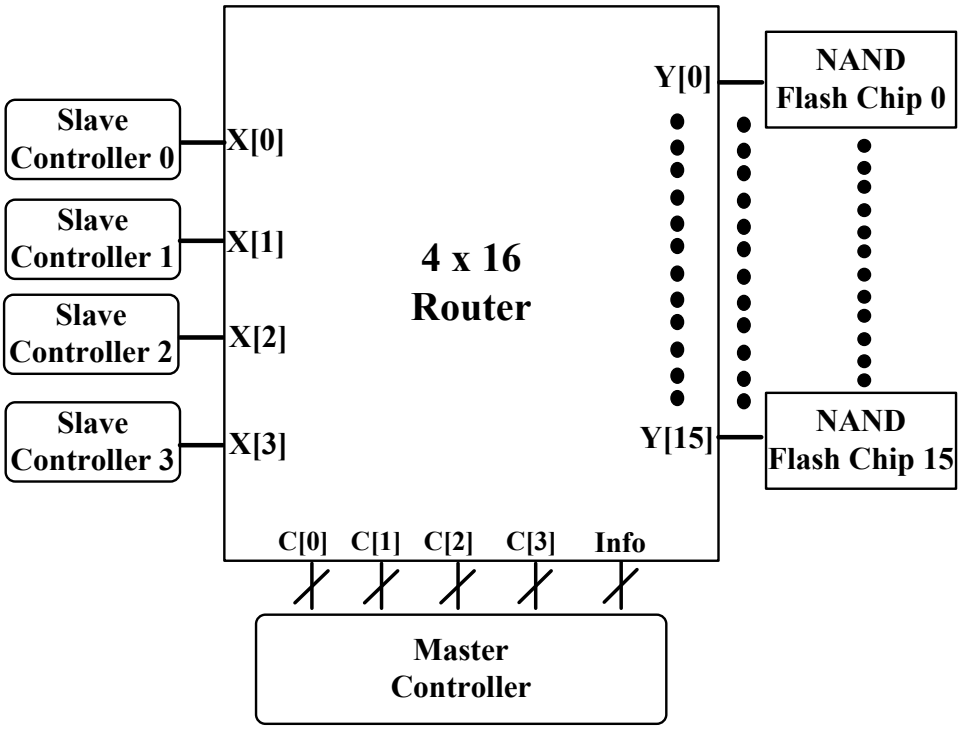


Fig. 4: A 4x16 router interface.

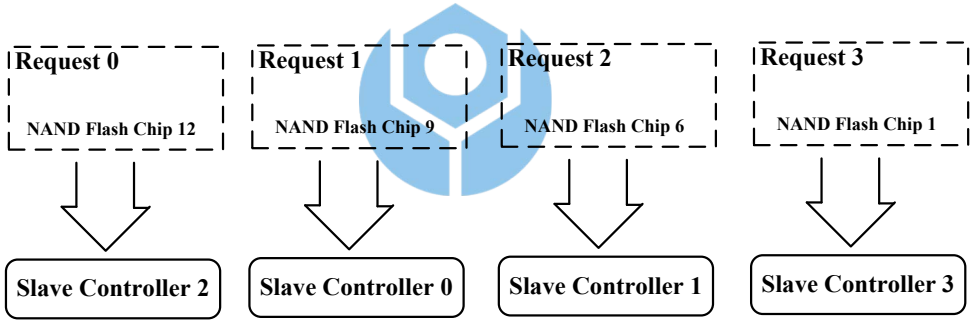


Fig. 5: 4 tasks.

2.3 Comparison with Previous Methods

There are many papers discussing the architectures of SSDs such as [23], [24], [25], [26], [27], [28], [29], [30], [32], and [31].We call the architectures of SSDs as TMCD (Traditional Multi-Controller Design) in the rest of thesis to compare with the proposed method (i.e., PMCD). Basically the architectures of [23], [24], [25], [26], and [27] are similar to Fig. 6(a). [25] and [26] discuss the performance with different numbers of buses (i.e., bus scalability). [27] discusses about how reordering the request sequence to improve the performance.

Although [23] and [24] emphasize on other topics, their architectures are also

similar to Fig. 6(a).

There are many slave controllers in the designs of [28],[29],[30], [31], and [32]. [28],[29], and [30] use BEE3 [18] as their basic hardware platforms whose architecture is shown in Fig. 6(d). The 4 FPGA chips on BEE3 play roles as one master controller and 3 slave controllers. These 4 controllers interconnect through the ring bus. Only the master controller interfaces with the host. Each controller has two DDR2 channels with two DIMMs (Dual In-line Memory Module) per channel. BEE3 uses the FDIMM (Flash Dual In-line Memory Module) which is a custom printed circuit with a CPLD (Complex programmable logic device) and 8 flash chips. A FDIMM that is just like a DIMM can be inserted into a DIMM slot. A FDIMM provides 8 independent flash channels. The architecture of [31] is implemented by the development board [21], as shown in Fig. 6(c). They use one master controller and 4 slave controllers. Each slave controller is in charge of specific flash chips. For example, in Fig. 6(c), each slave controller will handle 4 flash chips. [32] adopts the HAPS52 [20] FPGA hardware platform and its design is shown in Fig.6(b). There are one master controller, N slave controllers, and N flash chips in the design. The master controller can communicate with N slave controllers by a ring bus. The topology between N slave controllers and N flash chips is a reconfigurable switch. However, authors don't reveal too much details in the paper.

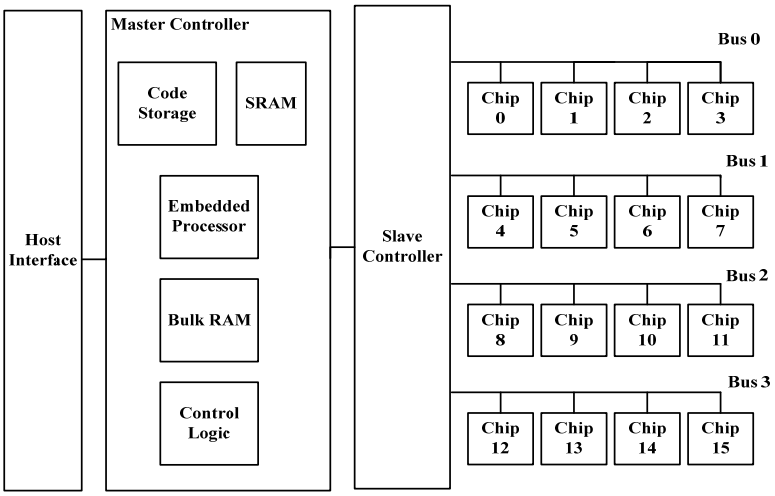


Fig.6 (a) Architecture of TMCD (1).

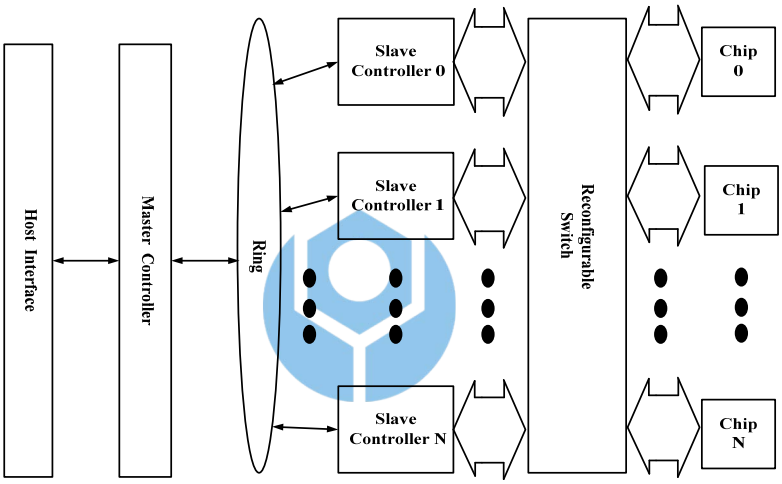


Fig.6 (b) Architecture of TMCD (2).

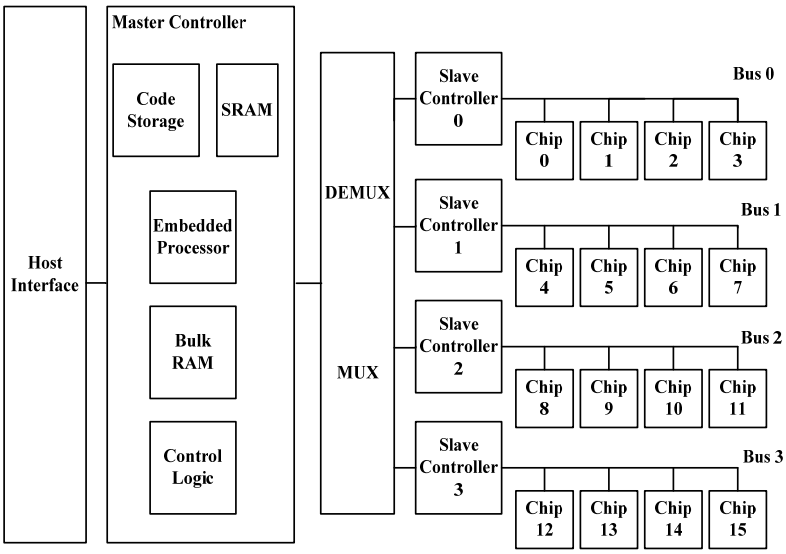


Fig.6 (c) Architecture of TMCD (3).

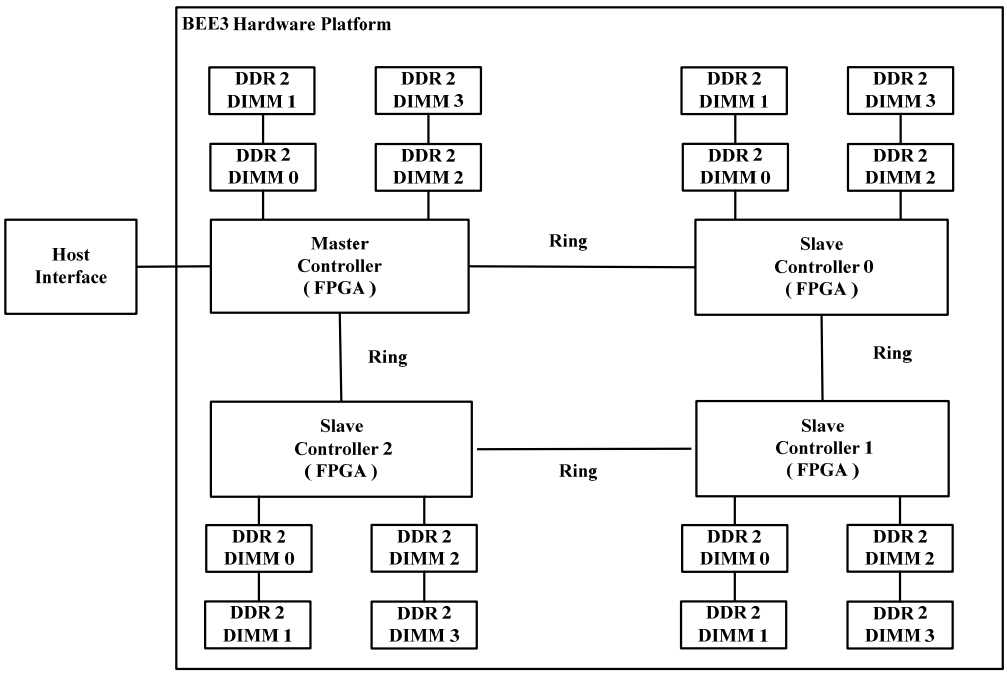


Fig.6 (d) Architecture of TMCD (4).

We need model a general architecture of TMCD to compare with the proposed method (i.e., PMCD). In the thesis, we will take [28], [29], [30], and [31] into consideration. We ignore [32] because of the unknown reconfigurable design. The general architecture of TMCD is modeled as shown in Fig. 7. Compared with PMCD, as shown in Fig. 2, the difference between them is how slave controllers connect to NAND flash chips. For TMCD, each slave controller can only access specific NAND flash chips through a 1x4 mux/demux. For PMCD, we want to build a router such that any NAND flash chip will not be restricted to any specific slave controller and any idle slave controller can access any NAND flash chip. The main difference between PMCD and TMCD is the bus constraint. PMCD doesn't have the bus constraint since two slave controllers can access two different NAND flash chips simultaneously in our design. However, when the same situation happens in TMCD, it is necessary to consider the bus constraint. For example, if the two different NAND flash chips belong to the same bus in TMCD, only one corresponding slave controller can access the two chips sequentially and decrease the parallelism even though there are other idle slave controllers. Therefore, such an observation motivates this research and the proposed PMCD will solve the problem.

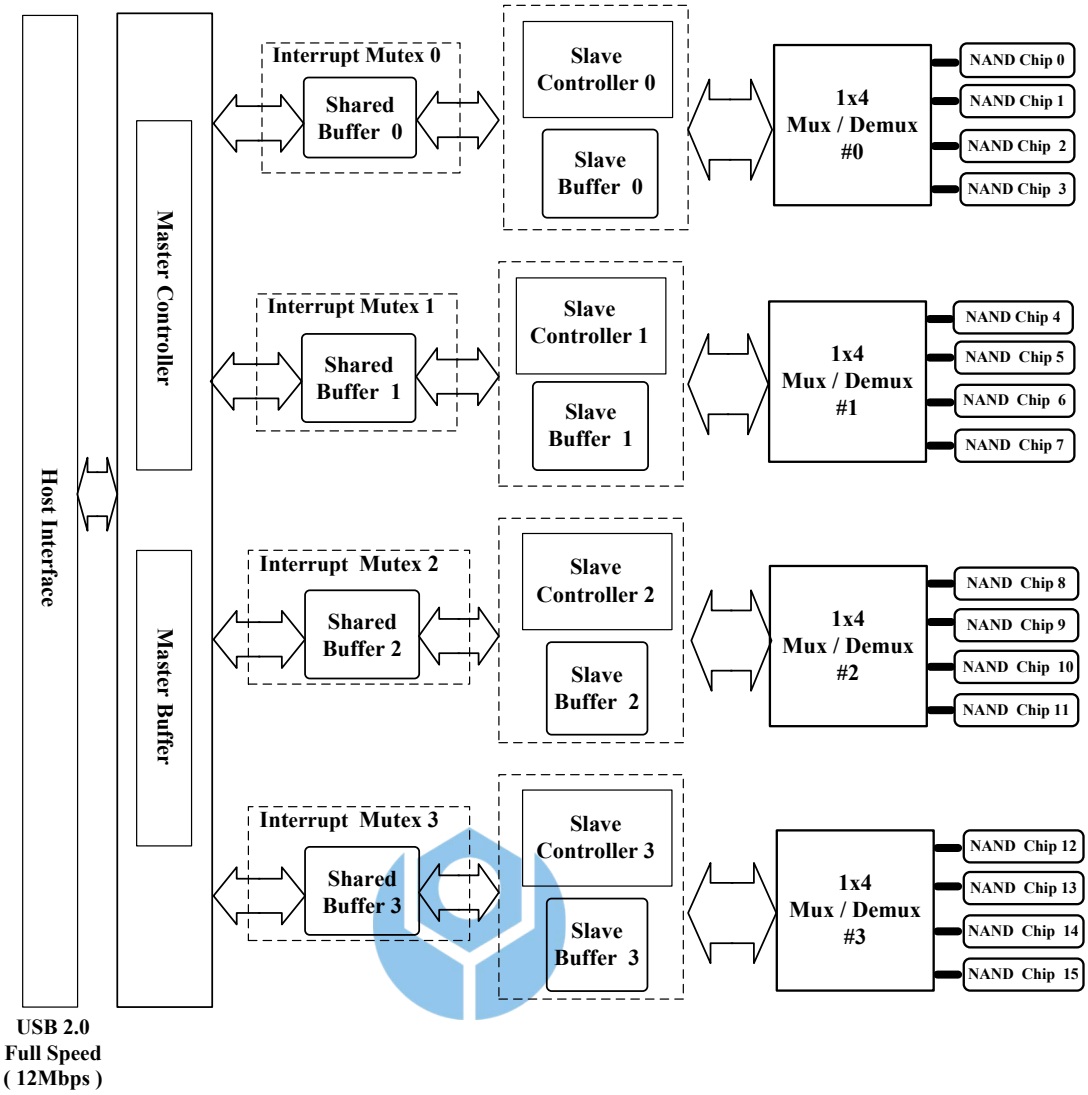


Fig. 7: A Traditional Multi-Controller Design for Solid-State Drives (TMCD).

2.4. Tasks Arrangement

Here we will describe how the master controller assigns tasks to the slave controllers after receiving the requests from the host for the comparison between TMCD and PMCD.

2.4.1 Step 1: Request Queue

After the host transmits the requests to the master controller, the master controller will save them into the request queue. Because of the USB interface

between the host and the master controller, the USB protocol is used and referred as the USB Mass Storage Device Class [35]. For the USB Mass Storage Device Class, there are some kinds of interface subclasses such as Reduced Block Commands (RBC), SFF-8020i, SFF-8070i, QIC-157, USB Floppy Interface (UFI) and SCSI transparent command set [35]. The SCSI transparent command set is usually bound with the USB Mass Storage Device Class and implemented on flash-based devices (e.g., USB thumb drives and SSDs). For the original USB Mass Storage Device Class, the host can only give a request to the device (i.e. the master controller in our design) at a time. The host can't assign a new request until the device finishes the current request and responds to the host. After UAS (USB Attached SCSI [37], [36]) and UASP (USB Attched SCSI Protocol [39]) are announced by USB-IF [38], it is possible that the host can assign multiple requests at a time. Therefore, the host can transmit multiple requests to the master controller at a time and some requests could be buffered in the request queue.

The master controller will analyze those requests in the request queue. The master controller translates a request into a tuple (op, Num Chip, Array Chip, Num Bus, Array Bus), where op denotes an operation (e.g., read, write, or erase), Num Chip denotes the number of chips that the request will access, Array Chip denotes a chip array that the request will access, Num Bus denotes the number of buses that the request will access, and Array Bus denotes a bus array that the request will access. We also check 3 kinds of constraints among the requests in the request queue such as address constraint, bus constraint, and chip constraint. The address constraint means that multiple requests have the overlapping address range. The bus constraint means that multiple requests will access some chips that belong to the same bus. The chip constraint means that multiple requests will access the same chip. Note that the proposed PMCD can resolve the bus constraint because of its design, but TMCD could decrease system performance due to the bus constraint.

2.4.2 Step 2: Reorder Requests

The step will try to reorder the read quests ahead of other requests (e.g., write or erase) unless the address constraint occurs. The purpose of the step is to improve overall throughput since the host could wait for the read requests. To avoid the address constraint in this step is similar to avoid two hazards such as RAW (Read After Write) and WAR (Write After Read) [41], or the conflicts on out-of-order execution [27].

2.4.3 Step 3: Assign Reorder Requests to Issues

An issue consists of several reordered requests. Each top request will be chosen from the request queue and inserted into the current issue if the request will not result in conflicts of the current issue. For example,an issue in TMCD should avoid the address constraint (e.g., RAW and WAR hazards), the bus constraint, and the chip constraint. An issue in PMCD should only avoid the address constraint and the chip constraint. The maximum number of requests in an issue is dependent on the number of slave controllers since every request in an issue can run concurrently to achieve the maximum parallelism. For example, assume that there are only 4 slave controllers. For PMCD, an issue may require 4 slave controllers at most to access 4 different chips. For TMCD, due to the bus constraint, a slave controller can only access specific NAND flash chips that belong to the same bus. Therefore, an issue may require 4 slave controllers at most to access 4 different chips that belong to different buses.

2.4.4 Step 4: Launch an Issue

The master controller will launch an issue at a time and assign requests in the issue to slave controllers through the interrupt mutex module. Once a slave controller finishes a request, it will transmit a finish signal through the corresponding interrupt mutex module to the master controller. The master controller also clears the relative settings of the N x M router for the slave controller. If there are remaining issues, the master controller will still launch an issue at a time.

Chapter 3 PERFORMANCE EVALUATION

3.1 Experimental Setup and Performance Metrics

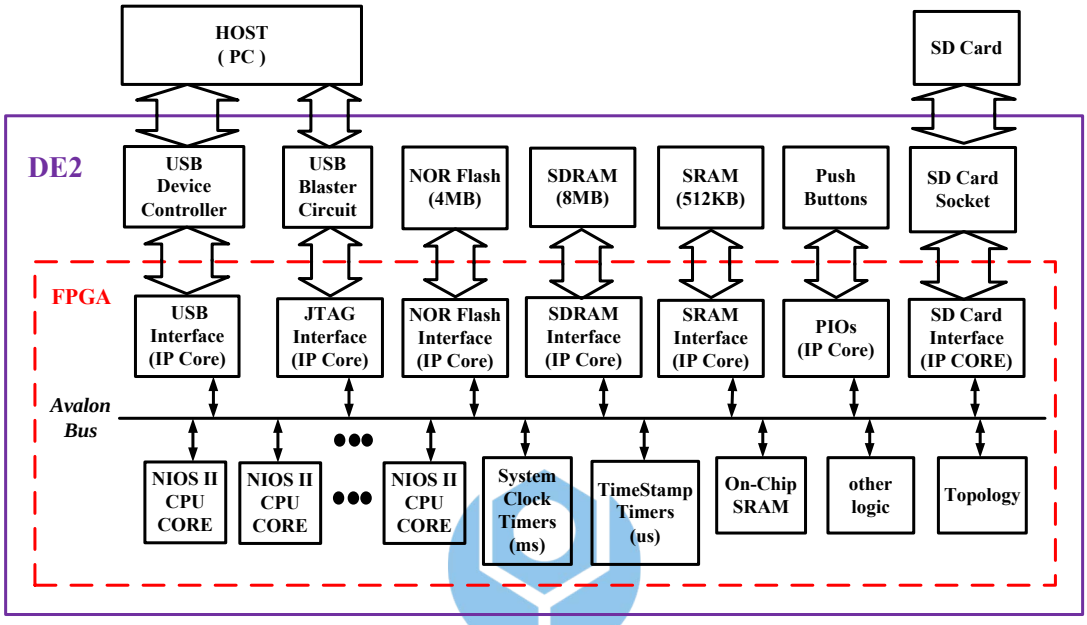


Fig. 8: DE2 Experimental Platform Architecture.

TABLE I: Experimental Environment.

Experimental Platform	Altera DE2
Software Development Tools	Quartus II, SOPC Builder, NIOS2IDE and Active HDL
Host Interface	USB 2.0 Full Speed (12Mbps)
USB Controller	On-Board Hardware(@12MHz)
Master Controller	NIOS II Core x1(@75MHz)
Slave Controllers	NIOS II Core x 4(Each @75MHz)
Master Buffer	64 KB (SDRAM)
4 Slave Buffer(s)	4 x 64 KB (SDRAM)
4 Shared Buffer(s)	4x 128 KB (SRAM)
Operating System	None
UFAT File System	FAT
Storage Device	2G SD Card (1 bit mode @50MHz) (Simulating 16 NAND Flash Chips)
Benchmark	IO Meter

As shown in Fig. 8, we constructed an experimental platform for both PMCD and TMCD by the Altera DE2 [4]. We used the Altera Quartus II [42] and SOPC Builder [43] to synthesize PMCD and TMCD, which generated the FPGA hardware configuration data and the Aldec Active HDL [45] to do the functional and timing verification on modules such as the interrupt mutex module, the NxM router, NAND flash chips and so on. We also edited the firmware code by the Altera NIOSII IDE [44]. Both the FPGA hardware configuration data and the firmware code were downloaded to DE2. In the experimental platform, several NIOS II Cores were used to play roles as controllers. Each NIOS II Core needed a system clock timer. In addition to the system clock timers, we used high resolution timers to measure the specific periods on performance evaluation. We also implemented one 4x16 router on PMCD and four 1x4 mux/demux interfaces on TMCD. There are different kinds of interface IP Cores linking up different off-chip devices on DE2 such as the USB controller, SDRAM, NOR flash memory, SD card and so on. The host transmitted the FPGA hardware configuration data and the firmware code into DE2 through the USB cable connecting to the USB Blaster Circuit binding in JTAG [46].The host received the debug information and the data about measuring specific periods in the same way. Table I shows more details about the platform in our design. We used the USB 2.0 as the interface between the host and DE2. The on-board USB hardware controller, working at 12MHz, will take responsibility for this job. One NIOS II Core played a role as the master controller, and 4 NIOS II Cores played as 4 slave controllers. All of these 5 controllers worked at 75MHz. The local buffers of master and slave controllers consisted of SDRAM. The shared buffers between the master controller and the slave controllers consisted of SRAM. The firmware code was placed in the NOR flash memory in advance and it was loaded into the SDRAM when the system ran. We also used a 2G SD card, with an 1 bit bus at 50MHz, to simulate 16 NAND flash chips. In the following, we will describe the details about the memory arrangement on DE2 and the simulation of 16 NAND flash chips by a 2G SD card.

3.1.1 Memory Arrangement on DE2

TABLE II: Memory Arrangement on DE2

Memory Name	Function	Assignee	Size
NOR Flash (total: 4MB)	Store Firmware Code	Master	1MB
		4 Slave	4 x 768KB
SDRAM (total: 8MB)	Load Firmware Code to Run and as the Buffer (4MB)	Master	1MB
		4 Slave	4 x 768KB
SRAM (total:512KB)	Master and Slave Exchange Message	4 Shared Buffer(s)	4 x 128 KB

The memory arrangement on DE2 is shown in Table II. The master controller is allocated to 1MB NOR flash and 1MB SDRAM, where the firmware code was 274KB and the local master buffer was 64KB in practice. Each slave controller is allocated to 768KB NOR flash and 768KB SDRAM, where the firmware code was 146KB and the local master buffer was 64KB in practice. The 512KB SRAM was for 4 shared buffers and each buffer was 128KB.

3.1.2 Simulation of 16 NAND Flash Chips

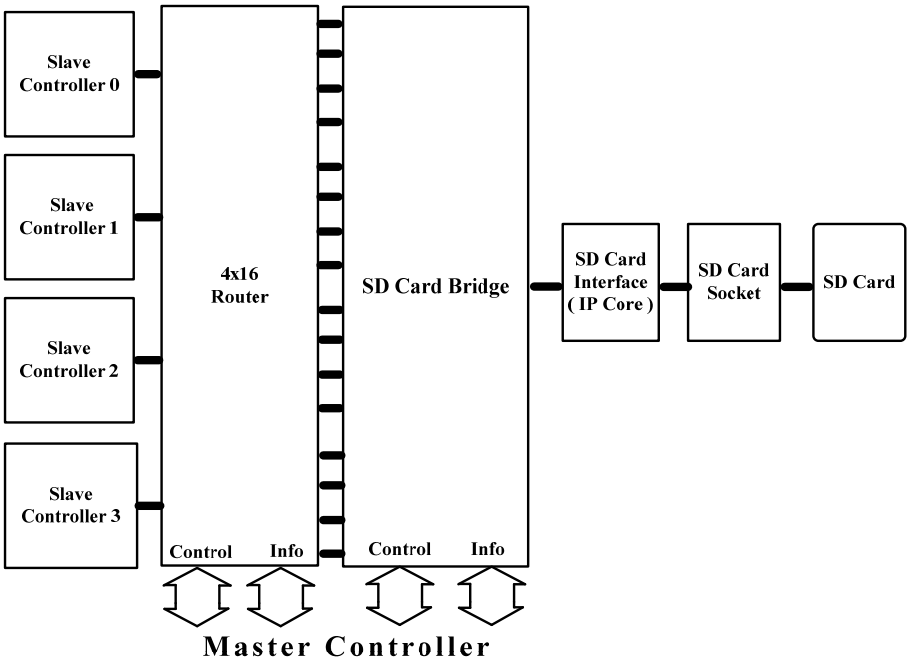


Fig. 9: Simulation of 16 NAND flash chips on PMCD.

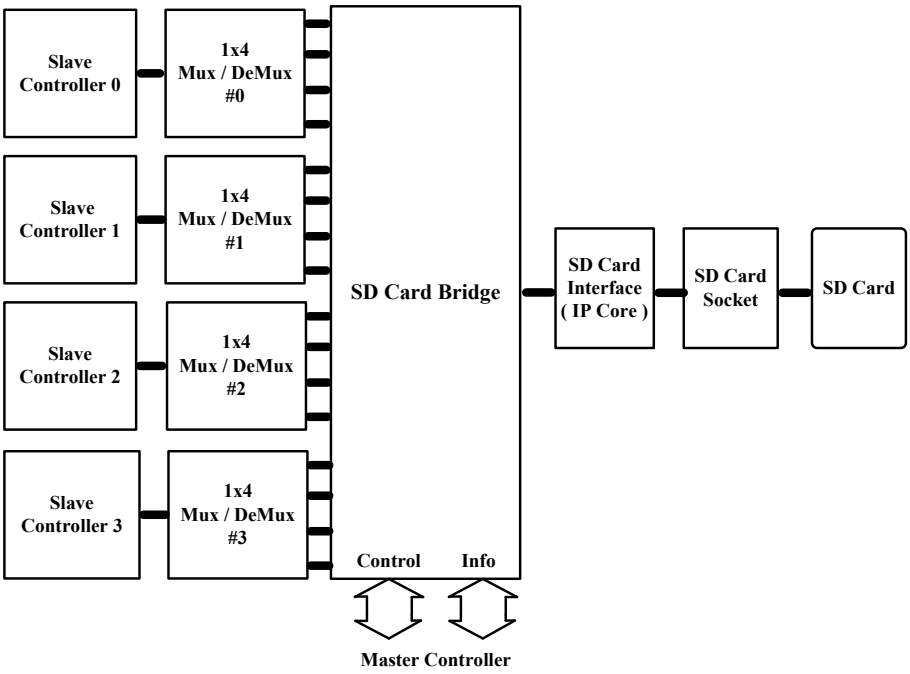


Fig. 10: Simulation 16 NAND flash chips on TMCD.

TABLE III: Simulation of 16 NAND flash chips by a 2G SD card

SD Card Capacity	2GB
Number of Chip	16
Page Size	512 Byte
Block Size	256 pages (128KB)
Chip Size	992 blocks (First 15 Chips)
	547 blocks (Last Chip)
Page Read 512Byte	401.30 µs (Sequential => 397.39 µs) (Random => 405.09 µs)
Page Write 512Byte	2.90 ms (Sequential => 2.35 ms) (Random => 9.16 ms)
Block Erase 128KB	33.84 ms (CMD38)
SD Card Bus	1 bit @50MHz
Ideal Throughput of SD Card Bus	6.25 MB/s
Practical Throughput of SD Card Bus	40.35 KB/s (12.69 µs/page)

Here we state how to simulate 16 NAND flash chips on PMCD and TMCD by a

SD card. As shown in Fig. 9 and Fig. 10, we need to add a SD card bridge between one 4x16 router (or four 1x4 mux/demux interfaces) and the SD Card Interface. The SD card bridge is handled by the master controller on both PMCD and TMCD. Table III shows the details about the simulation by a 2G SD Card. Since the page read time (i.e., 100ms) and the page write time (i.e., 250ms) provided by SD Specification [51] are the worst case, we get more accurate information by implementing the DCD (Duo Cores Design) on DE2 and sending multiple single-page requests (e.g., 512KB) to DCD by IO Meter [47]. There are only one master controller and one slave controller within the DCD. In addition to the page read time and the page write time, we also can measure practical throughput of SD Card bus in the same method. The block erase time can be measured by sending several block erase commands to DCD Each starting address of the erase commands is random. In fact, a block erase operation of the SD card is composed of three commands, CMD32, CMD33 and CMD38 defined in [51]. The first two commands, CMD32 and CMD33, can set the starting page address and the ending page address relatively while the last command, CMD38, erases the pages between them. Therefore, three basic operations (i.e., read, write, and erase) can be measured, as shown in Fig. 11. Someone may wonder why the SD card performance is slow, when compared with the NAND flash chips [1] (Read: 401.3μs v.s. 25 μs ; Write: 2.9 ms v.s. 200 μs ; Erase: 33.84 ms v.s. 1.5 ms). The reason is that the bus width of the SD card is only 1 bit on the DE2.

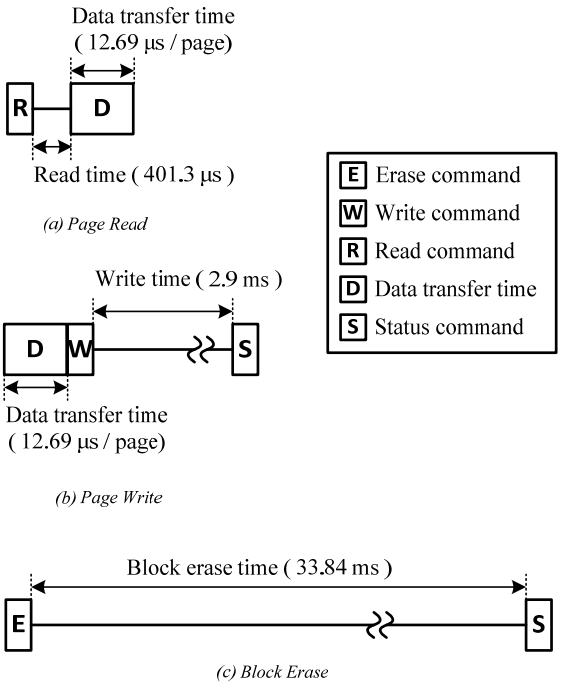


Fig. 11: Timing of three basic operations: read, write, and erase.

3.2 FPGA Synthesization Result

TABLE IV: Synthesization Results of TMCD and PMCD

	TMCD (TMCD/Available)	**TMCD (TMCD/Available)**	**Available**	**Ratio (PMCD / TMCD)**
Total Logic elements	17131 (52%)	20089 (60%)	33216	**1.17**
Total Combinational Functions	15993 (48%)	18923 (57%)	33216	**1.18**
Dedicated Logic Registers	8227 (25%)	9032 (27%)	33216	1.10
Total Registers	8344	9149	N/A	**1.10**
Total Pins	429 (90%)	429 (90%)	475	1.00
Total Memory Bits	316416 (65%)	316416 (65%)	483840	1.00
Embedded Multiplier 9-Bit Element	4 (6%)	4 (6%)	70	1.00
Total PLLs	1 (25%)	1 (25%)	4	1.00

TABLE V: Thermal Power Dissipation of TMCD and PMCD

	TMCD	**TMCD**	**Ratio (PMCD / TMCD)**
Total Thermal Power Dissipation	815.37	771.65	**0.95**
Core Dynamic Thermal Power Dissipation	398.91	437.25	1.10
Core Static Thermal Power Dissipation	80.87	80.8	1.00
IO Thermal Power Dissipation	335.6	253.61	0.76

The synthesization results of TMCD and PMCD are shown in Table IV. Every resource which PMCD uses is a little more than that of TMCD. In particular, the maximum ratio is 1.18. We also use the PowerPlay Power Analyzer [52] builded in

Altera QuartusII [42] to measure the thermal power dissipation, as shown in Table V. Total thermal power dissipation consists of the core dynamic thermal power dissipation, the core static thermal power dissipation, and the IO thermal power dissipation. The “Core Dynamic and Static Thermal Power Dissipation” show the estimated total core dynamic and static power dissipation by each clock domain respectively, which provides estimated power consumption for each clock domain in the design. The“IO Thermal Power Dissipation”denotes the total I/O power. We observe that the total thermal power estimation of PMCD is less than that of TMCD (e.g., the ratio is 0.95).

3.3 Benefit

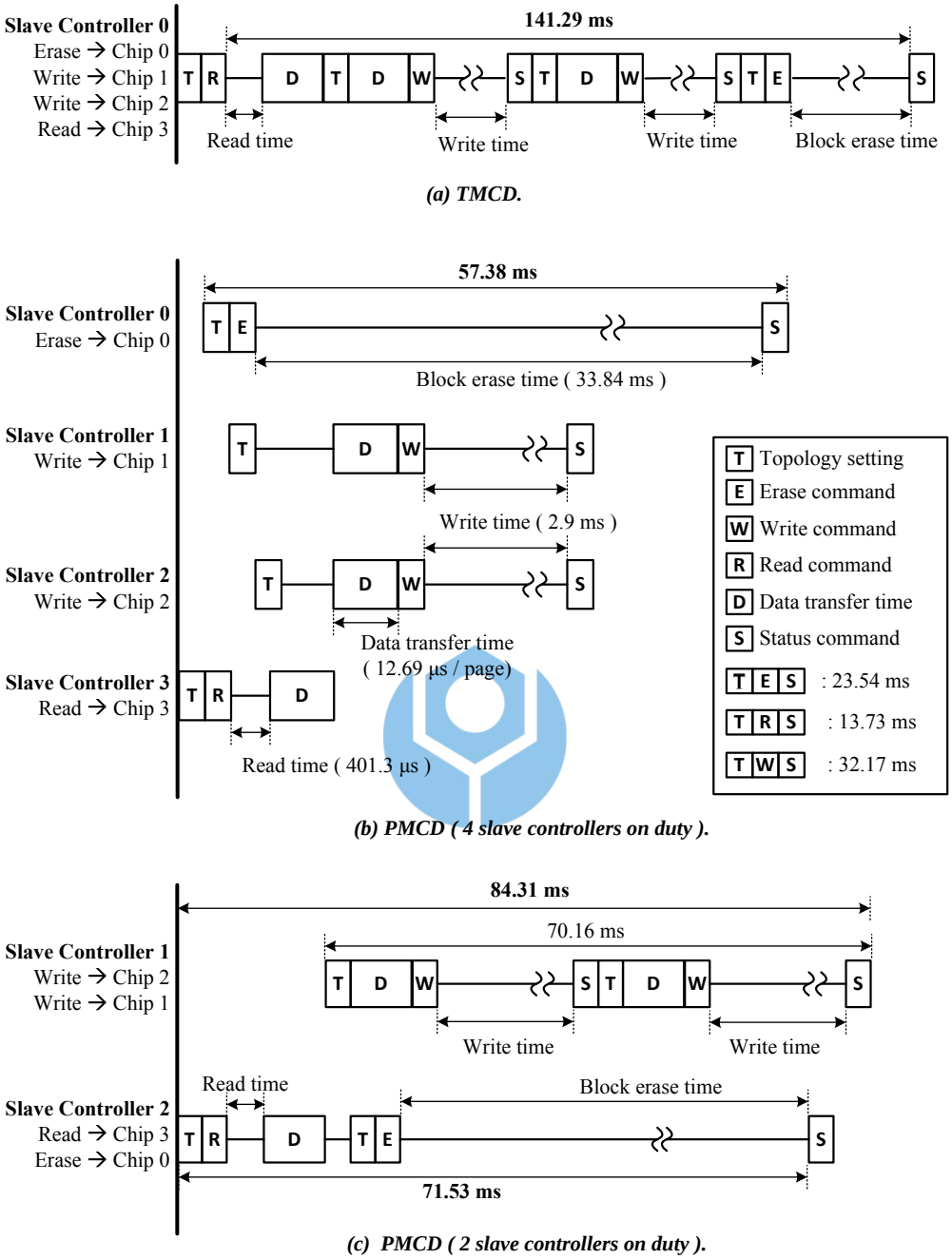


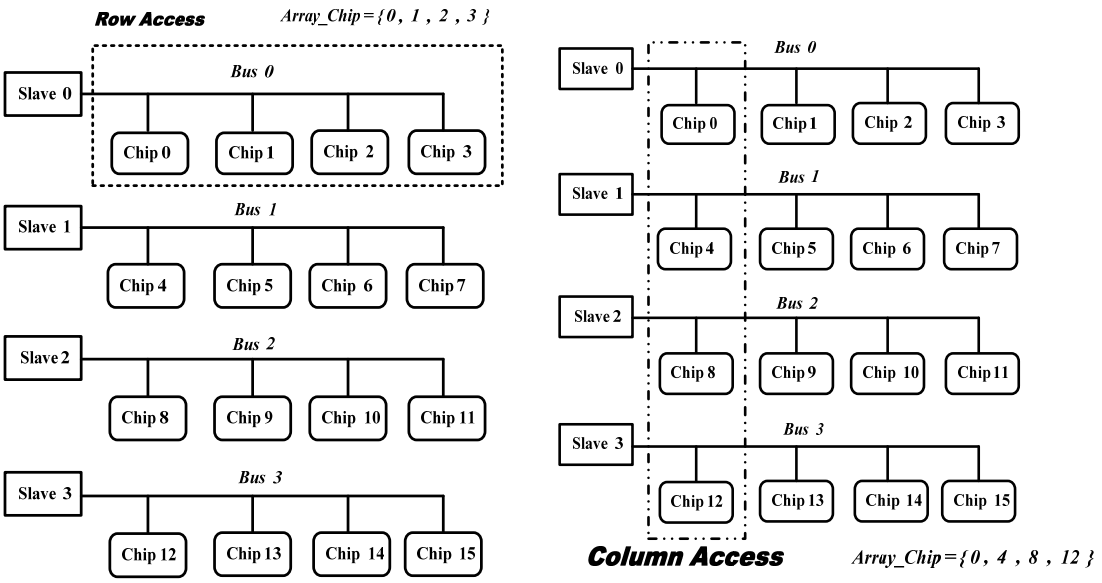
Fig. 12: Timing and assignment of slave controllers.

Since PMCD has no bus constraint when compared with TMCD. Each slave controller can access any NAND flash chips unless the chip constraint occurs. In the following, we will exhibit the advantage of PMCD better than that of TMCD. Assume that there are 4 requests: erase 1 block of Chip 0,write 1 page of Chip 1, write 1 page of Chip 2, and read a page of Chip 3. Fig. 12(a) and Fig. 12 (b) show how TMCD and

PMCD deal with the 4 requests, respectively. We use some symbols to denote specific commands. For example, “T”denotes the time period of topology setting for 1x4 mux/demux interface, 4x16 router, or SD card bridge. The average values of “TES”, “TRS” and “TWS” are measured by running the benchmark of IO Meter on DCD. Due to the bus constraint, the slave controller 0 is in charge of all requests one by one on TMCD. However, 4 slave controllers can run simultaneously on PMCD. Therefore, PMCD can take less accessing time than TMCD (e.g., 57.38ms v.s. 141.29ms). In this example, TMCD has 3 idle slave controllers during the whole procedure. As a result, total idle time of 4 slave controllers on PMCD is shorter than that of TMCD.

On TMCD, since a slave controller is responsible for specific NAND flash chips, TMCD will keep 4 slave controllers on duty, even though they are nothing to do. However, PMCD can turn off some slave controllers according to the current workload to achieve power efficiency. If the 4 requests in this example are considered as the light workload, PMCD can turn off 2 slave controllers and keep slave controller 1 and 2 on duty. As shown in Fig. 12(c), although total accessing time under this setting is more than that with 4 slave controllers on duty (e.g., 84.31ms v.s. 57.38ms), it is still better than that of TMCD (e.g., 84.31ms v.s. 141.29ms). As a result, PMCD can turn off some slave controllers with the acceptable decreasing performance, but TMCD has the fixed configuration to keep 4 slave controllers on duty all the time.

3.4 Simulation about Extreme Cases



(a) (b)

Fig. 13: Row access and column access.

In this Chapter, we will discuss about the impact of some extreme cases on both TMCD and PMCD. As shown in Fig.13, there two types of extreme cases on TMCD. That is the “row access” and the “column access”. For example, the typical “row access” is accessing Chip 0, Chip 1, Chip 2 and Chip 3 simultaneously, as shown in Fig. 13(a). The typical ”column access” is accessing Chip 0, Chip 4, Chip 8 and Chip 12 simultaneously, as shown in Fig. 13(b). Due to the bus constraint, the “row access” on TMCD causes declining performance. On the contrary, the ”column access” on TMCD can improve the performance because 4 slave controllers can run parallelly. Either the “row access” or “the column access” can increase the performance on PMCD since PMCD has no bus constraint.

We will show experimental results about the two extreme cases on TMCD and PMCD. We compare their performance on the access time, the speedup ratio, and the decrease ratio. As shown in Table VI, we define 9 patterns for the ”row access” and the “column access”. Each element within the “Length[ ]” array under each pattern is the access size and is corresponding to each element within the “Array_Chip”. “p”"means the unit of a page and “blk” means the unit of a block. On the “row access”, the performance of PMCD is better than that of TMCD with 27.3% average speedup ratio. On the “column access”, a slight average decrease about 0.08% on PMCD, when compared to TMCD. The reason is that the 4x16 router causes a little delay from the slave controllers to NAND flash chips. Overall, PMCD can improve performance significantly for the “row access” and have almost the same performance for the “column access”, when compared to TMCD.

TABLE VI: Simulation results of row and column access

	Row Access (Array Chip ={0, 1, 2, 3})			**Column Access** (Array Chip ={0, 4, 8, 12})		
	Access Time(μs)		**SpeedUp**	**Access Time(μs)**		**Decrease**
	TMCD	**PMCD**	**(%)**	**TMCD**	**PMCD**	**(%)**
4 Read (length[ ]={8p, 16p, 64p, 128p})	3288.02	1933.78	41.19	1930.45	1933.78	0.17
4 Write (length[ ]= {8p, 16p, 64p,128p})	6996.84	4126.50	41.02	4123.17	4126.50	0.08
4 Erase (length[ ]={4blk, 8blk, 32blk,64blk})	4154.61	2272.89	45.29	2272.89	2272.89	0.00
3 Read, 1 Write (length[ ]={8p, 32p, 128p, 128p})	6669.02	6049.14	9.29	6042.47	6049.14	0.11
2 Read, 2 Write (length[ ]= {8p, 128p, 8p, 128p})	6457.14	6051.62	6.28	6044.95	6051.62	0.11
1 Read, 3 Write (length[ ]= {128p, 8p, 32p, 128p})	7363.35	6054.13	17.78	6047.46	6054.13	0.11
2 Read, 1 Write, 1 Erase (length[ ]= {8p, 128p, 8p, 64blk})	4617.08	2552.44	44.72	2552.23	2552.44	0.01
1 Read, 2 Write, 1 Erase (length[ ]= {8p, 8p, 128p, 64blk})	6802.32	4253.64	37.47	4250.09	4253.64	0.08
1 Read, 1 Write, 2 Erase (length[ ]= {8p, 128p, 8p, 64blk})	2960.81	2552.44	13.79	2552.23	2552.44	0.01

Chapter 4 Conclusion

In the thesis, we propose a parallel multi-controller design (PMCD) for solid-state drives. We implement PMCD and the traditional multi-controller design (TMCD) on a FPGA-based development board (i.e., Altera DE2). We propose some custom modules such as the interrupt mutex module for communication among multiple controllers and the NxM router as the topology interface between the slave controllers and the NAND flash chips. The only difference between TMCD and PMCD is the topology interface. The interface inside TMCD lets a chip be only accessed by a specific slave controller, which is considered as the bus constraint, while that inside PMCD allows every slave controller to access any NAND flash chip unless two slave controllers access the same chip. We also propose a task arrangement which describes how the master controller assigns tasks to the slave controllers and presents how the slave controllers can run parallelly.The experimental results shows the resource utilization and power consumption of PMCD is almost equivalent to that of TMCD. The maximum ratio on the resource utilization is 1.18, and average ratio on the power consumption is 0.95. PMCD has about 27.3% improvement compared with TMCD on the row access (i.e., the bus constraint), and only 0.08% worse than TMCD on the column access. Overall, it proves that PMCD has better performance than TMCD and the overhead is reasonable. For future research, we need to enlarge the capacity of NAND flash chips, and upgrade our USB device and relative firmware compatible with the “USB 2.0 High Speed” or the “USB 3.0 Ultra High Speed” to get better throughput. We should further propose an efficient architecture for different requirements and cost for vendors. As a result, a sophisticated customization of solid-state drives and tool development will become important issues.

Reference

[1] R. Bez, E. Camerlenghi, A. Modelli, and A. Visconti, “Introduction to Flash Memory,” Proceedings of The IEEE, Vol. 91, No. 4, April 2003.

[2] SAMSUNG Electronics, K9F6408U0A-TCB0, K9F6408U0A-TIB0 FLASH MEMORY, http://www.samsung.com/, 2000

[3] SAMSUNG Electronics, 4Gb Flex-OneNAND M-die,http://www.samsung.com/, 2008

[4] Altera DE2 Development and Education Board, http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=53&No=30.

[5] M. Wu and W. Zwaenepoel, “ENVy: A Non-Volatile, Main Memory Storage System,” Proc. Sixth Int. Conf. Architectural Support for Programming Languages and Operating Systems(ASPLOS’94), Nov. 1994, pp. 86-97.

[6] A. Kawaguchi, S. Nishioka, and H. Motoda, “A Flash-Memory Based File System,” Proc. USENIX Ann. Technical Conf. (TCON’95), 1995

[7] A. Ban, “Flash File System Optimized for Page-Mode Flash Technologies,” US Patent no. 5,937,425, Aug. 1999.

[8] A. Gupta, Y. Kim, and B. Urgaonkar, “Dftl: a flash translation layer employing

demand-based selective caching of page-level address map-pings," Proc. the 14th international conference on Architectural support for programming languages and operating systems(ASPLOS'09), 2009, pp. 229-240.

[9] J. Kim, J.M. Kim, S.H. Noh, S.L. Min, and Y. Cho, "A Space-Efficient Flash Translation Layer for Compactflash Systems," IEEE Trans. Consumer Electronics, Vol. 48, No. 2, pp. 366-375, May 2002.

[10] T.S CHUNG, D.J PARK, S.W PARK, D.H. LEE, S.W. LEE, H.J.ANDSONG, "System software for flash memory: a survey," Proc. of the 2006 IFIP International Conference on Embedded And Ubiquitous Computing (EUC 2006), Aug. 2006, pp. 394-404.

[11] J.U. Kang, H. Jo, J.S. Kim, and J. Lee, "A Superblock-Based Flash Translation Layer for NAND Flash Memory," Proc. Sixth ACM and IEEE Int. Conf. Embedded Software (EMSOFT'06), 2006, pp. 161-170.

[12] S.W. Lee, D.J. Park, T.S. Chung, D.H. Lee, S. Park, and H.J. Song, "A Log Buffer-Based Flash Translation Layer Using Fully-Associative Sector Translation," ACM Trans. Embedded Computing Systems (TECS'07)), Vol. 6, No. 3, July 2007.

[13] C.H. Wu and T.W. Kuo, "An Adaptive Two-Level Management for the Flash Translation Layer in Embedded Systems," Proc. IEEE/ACM Int. Conf.

Computer-Aided Design(ICCAD'06), 2006, pp. 601-606.

[14] Y.G. Lee, D. Jung, D. Kang, and J.S. Kim, "μ-ftl: a memory-efficient flash translation layer supporting multiple mapping granularities, "Proc. of the 8th ACM international conference on Embedded software (EMSOFT'08), 2008, pp. 21-30.

[15] M. Rosenblum and J. Ousterhout, "The Design and Implementation of a Log-Structured File System," ACM Trans. Computer Systems, Vol. 10, No. 1, pp. 26-52, Feb. 1992.

[16] Cirrus Logic Inc., "EDB9315A Engineering Development Board," http://www.cirrus.com/en/pubs/manual/EDB9315A_Tech_Ref_ Manual.pdf.

[17] ARM Ltd., "RealView Platform Baseboard for ARM926EJ-S," http://infocenter.arm.com/help/topic/com.arm.doc.dui0224i/DUI0224I_realview_platform_baseboard_for_arm926ej_s_ug.pdf.

[18] BEE3: Revitalizing Computer Architecture Research, http://research.microsoft.com/projects/BEE3/

[19] HASTE (High-performance Advance Storage Technology Emulator).

[20] HAPS52: High-performance ASIC prototyping systems, http://www.synopsys.com/cgi-bin/sld/pdfdla/pdfr1.cgi?file=HAPS52-Virtex5-Motherboard-ds.pdf.

[21] E. H. Nam, K. S. Choi, J. Choi, H. J. Min, and S. L. Min, Hardware Platforms for Flash Memory/NVRAM Software Development, J. Computing Science and Eng. (JCSE), Vol. 3, No. 3, pp. 181-194, Sept. 2009

[22] J. J. Liao and C. H. Wu, "A Multi-Controller Architecture for High-Performance Solid-State Drives," ACM SigAPP Applied Computing Review, Vol. 12, No. 4, 2012

[23] J. H. Kim, D. Jung, J. S. Kim, and J. Huh, " A methodology for extracting performance parameters in solid state disks (SSDs)," Proc. IEEE Int. Symp. on Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS'09), Sept., 2009, pp. 1-10.

[24] S. Jung and Y. Song," Hierarchical Use of Heterogeneous Flash Memories for High Performance and Durability," IEEE Transactions on Consumer Electronics, Vol.55, pp. 1383-1391, Oct. 2009.

[25] N. Agrawal, V. Prabhakaran, T. Wobber, J. D. Davis, M. Manasse, and R. Panigrahy, " Design Tradeoffs for SSD Performance," Proc. USENIX 2008 Technical Conf., 2008, pp. 57-70.

[26] C. Dirik and B. Jacob, "The Performance of PC Solid-State Disks (SSDs) as a Function of Bandwidth, Concurrency, Device Architecture, and System Organization," Proc. ISCA'09, June, 2009, pp. 279-289.

[27] E. h. Nam, B. S. j. Kim, H. Eom, and S. L. Min, “Ozone(O3): An Out-of-Order Flash Memory Controller Architecture,” IEEE Transactons on Computers, Vol. 60, No.5, pp. 653-666, May 2011.

[28] J. D. Davis and L. Zhang, “FRP: a Nonvolatile Memory Research Platform Targeting NAND Flash,” Proc. The First Workshop on Integrating Solid-state Memory into the Storage Hierarchy, 2009

[29] A.M. Caulfield, J. Coburn, T.I. Mollov, A. De, A. Akel, J. He, A. Ja-gatheesan, R.K. Gupta, A. Snavely, and S. Swanson, “Understanding the Impact of Emerging Non-Volatile Memories on High-Performance, IO-Intensive Computing,” Proc. Int. Conf. High Performance Computing, Networking, Storage and Analysis (SC), Nov. 2010, pp. 1-11.

[30] A. M. Caulfield, A. De, J. Coburn, T. I. Mollov, R. K. Gupta, and S. Swanson, “Moneta: A High-performance Storage Array Architecture for Next-generation, Non-volatile Memories,” Proc. the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO’10), Dec. 2010, pp. 385-395.

[31] Y. J. Seong, E. H. Nam, J. H. Yoon, H. Kim, J. Y. Choi, S. Lee, Y. H. Bae, J. Lee, Y. Cho, and S. L. Min, “Hydra: A Block-Mapped Parallel Flash Memory Solid-State Disk Architecture”, IEEE Transactons on Computers, Vol. 59, No.7, pp. 905-921, July 2010.

[32] Y. Cai, E. F. Haratsch, M. McCartney, and K. Mai, “FPGA-Based Solid-State Drive Prototyping Platform,” Proc. IEEE 19th Annu. Int. Symp.on Field-Programmable Custom Computing Machines (FCCM’11), May 2011, pp. 101-104.

[33] M. Dubois, A.T. Laundrie, G.S. Sohi, “Scalable shared-memory multiprocessor architectures,” IEEE Trans. Comput. , Vol. 23, no. 6, pp.71-74, June 1990.

[34] POSIX Threads Programming, https://computing.llnl.gov/tutorials/pthreads/

[35] USB Mass Storage Class Specification Overview, http://www.usb.org/ developers/devclass docs/usb-msc-overview-1.3b.pdf

[36]Technical Committee T10, InterNational Committee on Information Technology Standards (INCITS), http://www.t10.org

[37] Technical Committee T10, “USB Attached SCSI (UAS, T10/2095-D),” http://www.t10.org/drafts.htm#SCSI3 UAS or http://www.t10.org/cgi-bin/ac.pl?t=f& f=uas-r04.pdf

[38] USB Implementers Forum, Inc. (USB-IF), http://www.usb.org

[39] Curtis E. Stevens, “USB Attached SCSI Protocol,” http://www.usb.org/ developers/presentations/pres0410/2-4_SSUSB_DevCon_UASP_Stevens.pdf

[40] Universal Serial Bus Revision 2.0 Specification (USB 2.0), http://www.usb.org/ developers/docs/usb_20_110512.zip

[41] D.A. Patterson and J.L. Hennessy, “Enhancing Performance with Pipelining,” in Computer Organization and Design, 3rd ed. Morgan Kaufmann,2005, ch. 6, pp. 368-453.

[42] Altera QuartusII, http://www.altera.com/

[43] SOPC Builder, http://www.altera.com/

[44] NIOSII IDE, http://www.altera.com/

[45] Active HDL, http://www.aldec.com/en/

[46] Joint Test Action Group(JTAG), IEEE 1149.1-1990

[47] IO Meter Benchmark, http://www.iometer.org/

[48] ATTO Benchmark, http://www.attotech.com/

[49] PCMark05, http://www.futuremark.com/

[50] AS SSD Benchmark, http://www.alex-is.de/PHP/fusion/news.php/

[51] SD Specifications Part 1 Physical Layer Simplified Specification Version 3.01, http://www.sdcard.org/

[52] PowerPlay Power Analyzer, http://www.altera.com/

國立臺灣科技大學博碩士論文授權書

(本授權書裝訂於紙本論文內)

本授權書所授權之論文為周茂儀〔M9902138〕在國立臺灣科技大學電子工程系 101 學年度第 1 學期取得碩士學位之論文。

論文題目：一個基於固態硬碟之可平行化的多控制器設計
指導教授：吳晉賢

茲同意將授權人擁有著作權之上列論文全文〔含摘要〕，依下述授權範圍，以非專屬、無償授權本校圖書館及國家圖書館，不限地域、時間與次數，以紙本、微縮、光碟或其他數位化方式將上列論文重製典藏，並提供讀者基於個人非營利性質之線上檢索書目、館內閱覽、或複印。

授權人	周茂儀	吳晉賢
簽章 (請親筆正楷簽名)	周茂儀	吳晉賢

備註：

1. 授權人不因本授權而喪失上述著作之著作權。
2. 本授權書請授權人簽章後，裝訂於紙本論文內。

中　華　民　國　102　年 2　月 4　日