고성능 플래시 메모리 솔리드 스테이트 디스크

(A High Performance Flash Memory Solid State Disk)

윤 진 혁 † 남 이 현 †† 성 윤 제 ††

(Jin Hyuk Yoon) (Eyee Hyun Nam) (Yoon Jae Seong)

김 홍 석 †† 민 상 렬 ††† 조 유 근 †††

(Hongseok Kim) (Sang Lyul Min) (Yookun Cho)

요 약 플래시 메모리는 전력 소모가 작고 충격과 진동에 강하며 크기가 작다는 특성 때문에 최근 노트북이나 UMPC(Ultra Mobile PC)와 같은 이동 컴퓨팅 시스템에서 하드디스크를 대체할 대용량 저장 매체로서 주목 받고 있다. 플래시 메모리에 기반한 저장 장치는 일반적으로 랜덤 읽기 성능이나 순차 읽기, 순차 쓰기 성능이 매우 좋은데 비해, 덮어쓰기가 불가능한 플래시 메모리의 물리적인 제약으로 인하여 소량의 랜덤 쓰기 성능은 떨어진다. 본 논문은 이 문제를 해결하기 위한 두 가지 중요한 특징을 갖는 SSD (Solid State Disk) 아키텍처를 제안하였다. 첫 번째로 비휘발성 이면서도 SRAM과 동일한 인터페이스로 덮어쓰기가 가능한 작은 크기의 FRAM(Ferroelectric RAM)을 NAND 플래시 메모리와 함께 사용하여 소량 쓰기 오버헤드를 최소화하였다. 두 번째, 호스트 쓰기 요청들도 소량 랜덤 쓰기와 대량 순차 쓰기로 분류하여 각각에 대해 최적의 쓰기 버퍼 관리 방법을 적용하였다. 평가 보드 상에서 SSD 프로토타입을 구현하고 PC 사용 환경의 워크로드에 기반한 벤치마크를 이용하여 성능을 평가해 본 결과 랜덤 패턴을 보이는 워크로드에서는 하드디스크나 기존의 상용 SSD들에 비해 처리율(throughput) 측면에서 3배 이상의 성능을 보였다.

키워드 : 플래시 메모리, SSD(Solid State Disk), 플래시 사상 계층(FTL), 플래시 메모리 기반 저장 장치 플랫폼

Abstract Flash memory has been attracting attention as the next mass storage media for mobile computing systems such as notebook computers and UMPC(Ultra Mobile PC)s due to its low power consumption, high shock and vibration resistance, and small size. A storage system with flash memory excels in random read, sequential read, and sequential write. However, it comes short in random write because of flash memory's physical inability to overwrite data, unless first erased. To overcome this shortcoming, we propose an SSD (Solid State Disk) architecture with two novel features. First, we utilize non-volatile FRAM (Ferroelectric RAM) in conjunction with NAND flash memory, and produce a synergy of FRAM's fast access speed and ability to overwrite, and NAND flash memory's low and affordable price. Second, the architecture categorizes host write requests into small random writes and

· 본 연구는 정보통신부 및 정보통신연구진흥원의 IT신성장동력핵심기술개발사업의 일환으로 수행하였음 [2006-S-040-01, Flash Memory 기반 임베디드 멀티미디어 소프트웨어 기술 개발]

† 정 회 원 : 엠트론스토리지테크놀로지(주)
jhyoon@mtron.net
†† 학생회원 : 서울대학교 컴퓨터공학부
ehnam@archi.snu.ac.kr
yjsung@archi.snu.ac.kr
hskim@archi.snu.ac.kr
††† 종신회원 : 서울대학교 컴퓨터공학부 교수
symin@archi.snu.ac.kr
ykcho@snu.ac.kr

논문접수 : 2007년 9월 10일
심사완료 : 2008년 5월 9일

정보과학회논문지: 컴퓨팅의 실제 및 레터 제14권 제4호(2008.6)

large sequential writes, and processes them with two different buffer management, optimized for each type of write request. This scheme has been implemented into an SSD prototype and evaluated with a standard PC environment benchmark. The result reveals that our architecture outperforms conventional HDD and other commercial SSDs by more than three times in the throughput for random access workloads.

Key words : Flash memory, SSD(Solid State Disk), flash translation layer(FTL), flash memory storage device platform

1. 서 론

플래시 메모리는 EEPROM(Electrically Erasable/Programmable ROM)의 일종으로 바이트 단위로 지우기(erase) 작업을 수행하는 기존 EEPROM과는 달리, 수십에서 수백 킬로바이트 정도의 큰 단위를 한 번에 지울 수 있는 비휘발성 메모리이다. 플래시 메모리는 데이타를 저장하는 셀의 물리적인 구조에 따라 크게 NOR 형과 NAND 형으로 나뉜다. 이중 NAND 플래시 메모리는 쓰기 단위가 커서 쓰기 속도가 빠르며, 집적도가 높아 가격이 싸고 대용량의 칩을 만드는데 유리하여 주로 CF, SD, MMC 메모리 카드나 USB 드라이브와 같은 휴대형 저장 장치의 데이타 저장 매체로 많이 사용된다. 최근에는 노트북, UMPC(Ultra Mobile PC) 등의 이동 컴퓨팅 시스템에서 하드디스크를 대체하기 위한 용도로도 주목 받고 있다.

그러나 플래시 메모리에는 덮어쓰기(overwrite)가 불가능하다는 물리적인 제약이 존재한다. 한번 쓰기 작업이 수행된 영역에는 임의의 다른 값을 갖는 데이타를 덮어쓰지 못하며, 새로운 데이타를 기록하기 위해서는 먼저 실제 기록할 데이타보다 훨씬 큰 영역에 대해 지우기 작업을 수행해야 한다. NAND 플래시 메모리의 쓰기 단위는 페이지(page)라고 부르며, 보통 2KB의 크기를 갖는다. 지우기 단위는 블록(block)이라고 부르며, 보통 64개의 페이지로 구성되어 128KB의 크기를 갖는다. 페이지 쓰기 시에는 200μs, 블록 지우기 시에는 2ms가 소요되고 페이지 단위의 읽기 시에는 20μs가 소요된다[1].

덮어쓰기가 불가능하다는 문제를 해결하기 위해 플래시 메모리 기반 저장 장치에서는 FTL(Flash Translation Layer)이라고 불리는 소프트웨어 계층을 사용하는 것이 일반적이다[2,3]. FTL은 외부로는 플래시 메모리 저장 장치를 디스크와 동일하게 섹터들의 집합으로 추상화시켜 보여주면서 내부적으로는 논리적인 섹터와 플래시 메모리와의 사상을 담당한다.

현재의 하드디스크를 그대로 대체하기 위해 개발된 SSD(Solid State Disk)는 호스트 인터페이스로서 ATA, SCSI와 같은 전통적인 디스크 인터페이스를 가지며 물리적인 폼펙터에 있어서도 디스크와 동일한 플래시 메모리 저장 장치이다. 그림 1은 SSD의 전체 개요를 보여주고 있다. 호스트에서는 기존의 하드디스크를 위한 장치 드라이버를 포함하여 파일 시스템과 가상 메모리 시스템을 그대로 사용한다. SSD는 호스트와는 독립된 시스템으로 동작하며 플래시 메모리 칩들과 이들을 제어하기 위한 제어기, FTL 소프트웨어로 구성된다.

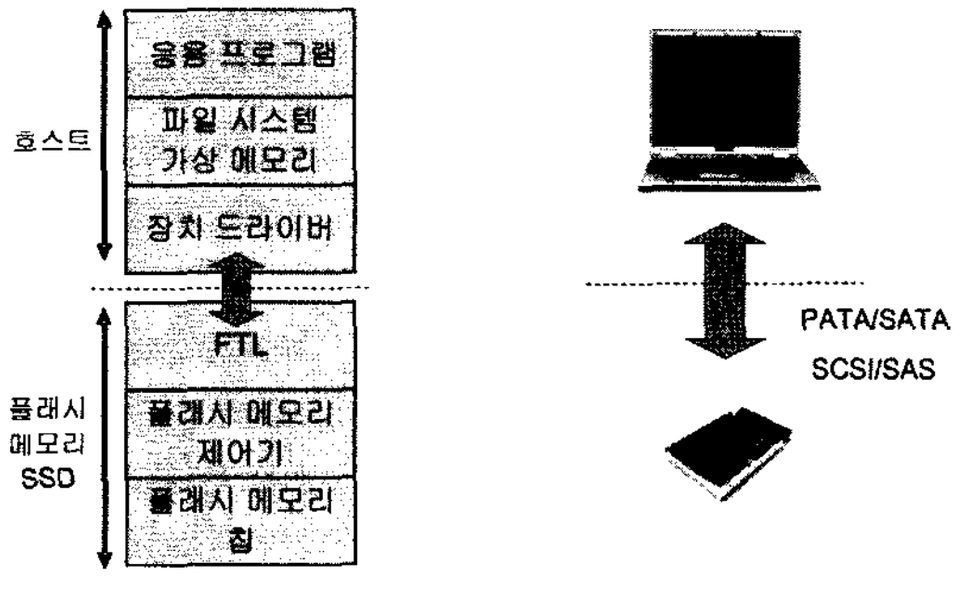


그림 1 SSD 개요

본 논문에서 다루고 있는 또 하나의 비휘발성 저장 매체는 FRAM(Ferroelectric RAM)이다. FRAM은 SRAM과 유사한 인터페이스와 성능을 가지는 차세대 비휘발성 메모리로서 전하를 보유하는 커패시터로 강유전성을 갖는 물질을 사용하여 전원 공급이 중단되어도 데이타를 유지할 수 있다[4]. FRAM은 SRAM과 같이 바이트 단위의 읽기, 쓰기 기능을 제공하며 접근 시간이 100ns 정도로 빠르다. 또한, FRAM은 생산 공정 상 일반 논리 회로 칩을 만드는 공정과의 호환성이 뛰어나 쉽게 프로세서에 내장할 수 있다는 특징이 있다. SSD 제어 ASIC 칩을 제조하는 과정에서 원하는 만큼의 FRAM을 내장하는 것이 가능할 것으로 예상되며 논리 회로 공정의 발전과 함께 내장된 FRAM 용량도 자연스럽게 증가할 것으로 예상된다.

플래시 메모리를 이용한 저장 장치는 플래시 메모리의 특성에 의해, 일반적으로 다음과 같은 성능 특성을 갖는다. 먼저, 랜덤 읽기 성능의 경우 물리적인 헤드와 스핀들 등을 동작시켜야 하는 하드디스크에 비해 월등한 성능을 보여준다. 또한, 순차 읽기 및 순차 쓰기 성능에서도 단일 플래시 메모리 칩 처리율(throughput)

측면에서는 디스크에 비해 떨어지지만, 여러 칩을 동시에 구동하는 방법을 통해 원하는 만큼의 충분한 처리율을 얻을 수 있다. 그러나, 랜덤 쓰기 특히 소량의 랜덤 쓰기 성능에 있어서는 덮어쓰기가 불가능하다는 물리적인 제약으로 인해 디스크 랜덤 쓰기와 비슷하거나 오히려 더 낮은 성능을 보이기도 한다. 따라서, 소량 랜덤 쓰기 요구를 얼마나 효율적으로 처리할 수 있는지는 플래시 메모리 기반 저장 장치 설계에 있어 주요한 고려사항의 하나이다.

이 문제를 해결하기 위해 본 논문에서는 플래시 메모리 저장 장치에서 비휘발성을 요구하는 데이타를 분류하여 각각의 특성에 가장 적합한 처리 방법을 적용하였다. 사상 테이블을 포함한 FTL 메타데이타는 전체 크기가 작고 빈번한 소량 갱신이 이루어지므로 소량 갱신 측면에서 유리한 FRAM에 유지하고 호스트 쓰기 요청에 의한 호스트 데이타는 최소한 512B 이상의 섹터 단위의 쓰기 작업이 수행되므로 저비용, 대용량의 NAND 플래시 메모리에 유지한다. 또한, 플래시 메모리에 저장하는 호스트 데이타도 쓰기 요청 패턴에 따라 소량 랜덤 쓰기와 대량 순차 쓰기로 나누어 각 패턴을 처리하는데 적합한 페이지 단위 쓰기 버퍼와 블록 단위 쓰기 버퍼에 나누어 처리하였다. 이러한 FRAM 활용과 쓰기 버퍼 관리 기법을 통해 소량 랜덤 쓰기 성능 저하를 최소화한 고성능 SSD 아키텍처를 제안하고 프로토타입 구현을 통해 그 실효성을 검증하였다.

이후의 논문 구성은 다음과 같다. 2장에서는 플래시 메모리와 FRAM과 관련하여 진행된 연구들을 정리하였다. 3장에서는 본 논문에서 제안하는 아키텍처의 주요 특징에 대해 기술하고 4장에서 프로토타입의 구현에 대해 간략히 설명하고 성능 결과를 제시하였다. 마지막으로 5장에서 결론을 도출하고 향후 연구 계획에 대해 기술하였다.

2. 관련 연구

플래시 메모리에서는 덮어쓰기가 불가능하다는 물리적 제약을 극복하기 위해 소프트웨어적인 지원이 필수적이며, 이에 대한 연구가 활발히 진행되었다[1]. FTL의 사상 방법은 크게 페이지 단위 사상과 블록 단위 사상으로 나뉜다. 페이지 단위 사상의 경우 쓰기 단위인 페이지를 기준으로 사상하여, 논리 섹터들이 임의의 플래시 메모리 페이지에 기록될 수 있으며, 로그 구조 파일 시스템(Log Structured File System)[5]과 비슷한 방법으로 운영된다. 논리 섹터를 임의의 플래시 페이지에 기록할 수 있다는 유연성 때문에 쓰기 작업 시 호스트의 접근 패턴과는 상관없이 항상 플래시 메모리의 연속적인 페이지들에 순차적으로 쓸 수 있다는 장점이 있다. 페이지 단위 사상 기법에서는 비용/이득 분석(cost/benefit analysis)에 기반한 세그먼트 정리 정책(segment cleaning policy)과 데이타 접근 패턴에 따른 데이타 분류 (data clustering) 정책에 관한 연구가 진행되었다[6-8].

그러나 페이지 단위 사상은 사상 정보의 양이 매우 크다는 단점이 있고, 일단 플래시 메모리 전체를 한 번 쓰고 난 뒤에는 여전히 쓰레기 수집 작업을 통해 지워진 공간을 확보하는 오버헤드가 발생한다. 특히, 사상 정보의 양으로 인해 대용량 저장 장치 플랫폼에서는 실질적으로 채택하기 어려운 방법이다.

블록 단위의 사상은 사상 정보의 양이 페이지 단위 사상에 비해 현저히 작아서 실제 대용량의 플래시 메모리 저장 장치에서 많이 사용되는 방법으로 지우기 단위인 블록을 기준으로 사상을 하며 블록 내에서는 논리 섹터와 물리 섹터의 위치가 동일하게 고정된다. 블록 단위 사상을 위해 논리 섹터 주소 공간은 물리적 블록 크기 또는 그의 배수의 크기를 갖는 논리 블록으로 구분되며, 특정 시점에서 호스트로 제공되는 모든 논리 블록에 대해 각각에 사상된 물리 블록 또는 물리 블록 집합이 존재한다. 그러나, 데이타 블록이라고 부르는 이러한 블록들에는 이미 현재의 데이타가 쓰여져 있으며, 덮어쓰기가 불가능하다. 쓰기 처리를 위해서는 몇 개의 물리 블록을 논리 블록 사상에서 제외해 두고 일단 이러한 블록들에 쓰기를 처리한 뒤 기존 데이타 블록의 변경되지 않은 섹터들과 새로 쓰여진 섹터들을 모아 새로운 데이타 블록을 구성하는 작업이 필요하다. 쓰기 작업을 위해 논리 블록 사상에서 제외해 둔 블록을 쓰기 버퍼 블록(write buffer block)이라고 부르며 쓰기 버퍼 블록과 기존 데이타 블록 내의 유효한 페이지들을 모아 새로운 물리 블록을 구성하는 작업을 블록 병합(block merge)이라고 한다.

블록 단위 사상 기법과 관련해서는 대체 블록(replacement block), 로그 블록 등으로 불리는 쓰기 버퍼 블록의 관리 정책에 대한 연구가 진행되었다[9,10]. 또한, 두 가지 기법의 장단점을 모두 취할 수 있는 복합적인 관리 기법에 대한 연구[11]와 사상 정보량 및 사상 오버헤드를 줄이기 위한 연구[12] 등이 진행되었다. 본 논문에서도 기본적으로는 블록 단위 사상을 이용하고[9] 쓰기 버퍼 관리 방식에서는 [10,11]과 유사한 복합적인 관리 정책을 사용하였다.

그러나, 기존 연구들은 대부분 내장형 시스템에서 직접 플래시 메모리를 관리하는 경우에 관한 연구로서 일반적인 내장형 시스템보다 더욱 심한 자원 제약을 갖는 SSD와 같은 독립적인 대용량 저장 장치에 관한 연구는 많이 이루어지지 않았다. 기존 연구들에서는 대부분 사

상 테이블을 전부 SRAM에 유지하고 부팅 시 전체 플래시 메모리를 검색하여 사상 테이블을 재구성하는 방식을 가정하고 있는데, SSD 플랫폼에서는 실질적으로 사용할 수 없는 방법이다. 즉, 전체 사상 테이블을 비휘발성 저장 매체에 별도로 유지하고 실행 시 전체 테이블에서 현재 필요로 하는 일부만을 메인 메모리로 읽어들여 작업하는 방식이 필요하다. 이러한 경우 변경된 사상 정보를 플래시 메모리에 갱신하는 작업의 오버헤드에 의해 성능이 많이 저하되는데 이를 FRAM을 활용하여 해결하였다.

FRAM을 포함한 차세대 비휘발성 메모리와 관련된 연구로서는 파일 시스템 수준에서의 메타데이타 관리 방법에 대한 연구가 수행되었다[13-15]. 디스크를 기반으로 MRAM을 메타데이타 저장소와 쓰기 버퍼로 사용하는 파일 시스템(HeRMES: High-Performance Reliable MRAM-Enabled Storage)에 대한 아이디어가 제안되었으며[13] 배터리 지원 RAM을 이용한 대량의 비휘발성 버퍼를 사용하는 Conquest 파일 시스템 연구가 진행되었다[14]. 플래시 메모리 기반 파일 시스템과 관련되어서는 대표적인 플래시 메모리 파일 시스템인 YAFFS[16]를 개선하여 비휘발성 메모리를 활용할 수 있도록 한 MiNV 파일 시스템에 대한 연구가 수행되었다[15]. 이들 연구에서도 메타데이타나 작은 파일들에 대한 소량의 빈번한 변경 처리를 효율적으로 하기 위해 비휘발성 메모리를 사용하였으나 저장 장치 수준에서의 활용을 다루는 본 연구와는 달리 파일 시스템 수준에서의 비휘발성 메모리 활용을 다루고 있다.

3. 아키텍처 설계

본 논문에서 제안하는 아키텍처는 다음과 같은 특징을 갖는다. 전체적인 구조 측면에서, 플래시 메모리와는 달리 작은 쓰기 단위를 가지고 덮어쓰기 제약이 없는 FRAM과 대용량의 가격이 싼 NAND 플래시 메모리를 함께 사용하여 효율성과 경제성 측면에서 양쪽의 장점을 취하였다. FTL 사상 측면에서는 호스트로부터의 쓰기 요구를 소량의 랜덤 쓰기와 대량의 순차 쓰기로 구분하고 각각에 적합한 쓰기 버퍼 관리 방법을 적용하여 랜덤 쓰기에 유리한 페이지 단위 사상과 순차 쓰기에 유리한 블록 단위 사상 양쪽의 장점을 활용하였다. 하드웨어적인 구조는 단일 NAND 플래시 메모리 칩의 읽기, 쓰기 성능 제한을 해결하기 위해 칩 단위, 버스 단위로 인터리빙(interleaving) 기법을 적용하였고 호스트 인터페이스와 플래시 인터페이스 간의 데이타 전송을 위해 독립적인 데이타 전송 경로를 추가하였다.

3.1 FRAM 활용

3.1.1 비휘발성 데이타 및 저장 매체 특성 분석

플래시 메모리 저장 장치에서 비휘발성을 갖도록 유지되어야 하는 데이타들은 크게 호스트에서 읽기, 쓰기 요청을 통해 섹터 단위로 접근하는 호스트 데이타와 이들 호스트 데이타에 대한 정보 즉, 저장 장치 내부적으로 유지하는 메타데이타로 나눌 수 있다. 호스트 데이타는 전체 저장 장치의 현재 데이타에 해당하는 영구적인(permanent) 영역과 최근 쓰기 작업이 수행된 섹터들에 해당하는 일시적인 (transient) 영역으로 나눌 수 있다. 일시적인 영역의 섹터들은 쓰기 버퍼에 기록되었다가 이후의 쓰기 작업이 진행되면서 블록 병합 작업에 의해 블록 단위로 영구적인 영역으로 이동하게 된다.

FTL 메타데이타는 읽기 전용인 데이타와 동작 중 변경되는 데이타로 나눌 수 있다. 읽기 전용 정보로는 플래시 메모리 칩 수와 종류, 호스트에 제공되는 전체 용량 등의 저장 장치의 구성 정보가 있다. 변경되는 메타데이타로는 일시적인 영역에 속하는 쓰기 버퍼들의 상태를 나타내는 정보와 영구적인 영역에 해당하는 전체 논리 블록에 대한 블록 사상 테이블이 있다. 표 1은 이러한 비휘발성 데이타들을 정리한 것이다.

비휘발성 데이타에 대한 쓰기 요구들을 발생 빈도가 높은 것부터 살펴보면, 가장 빈번하게 변경되는 정보는 매 호스트 쓰기 요구 시마다 변경되는 쓰기 버퍼 상태

표 1 비휘발성 데이타 분류

데이타 종류			전체 크기	변경 빈도	변경 크기
호스트 섹터 데이타	영구적인 영역	데이타 블록	저장 장치 용량	블록 병합 시	블록 크기
	일시적인 영역	블록 단위 쓰기 버퍼 블록	설정 가능 (다수 개의 블록)	호스트 쓰기 요청 시 블록 병합 시	다수 개의 섹터
		페이지 단위 쓰기 버퍼 블록	설정 가능 (다수 개의 블록)	호스트 쓰기 요청 시 블록 병합 시 쓰레기 수집 시	
FTL 메타 데이타	변경 가능 데이타	블록 사상 정보	저장 장치 용량 GB 당 < 32KB	블록 병합 시	< 64B
		쓰기 버퍼 정보	< 4KB	호스트 쓰기 요청 시 블록 병합 시 쓰레기 수집 시	< 16B
	읽기 전용 데이타	저장 장치 구성 정보	< 2KB	해당 없음	해당 없음

정보이다. 모든 쓰기 버퍼 블록들의 상태를 담고 있는 자료구조의 크기는 4KB 정도이지만 실제 변경되는 정보는 호스트 쓰기를 처리한 해당 쓰기 버퍼 블록에 관련된 정보로서 16B 이하이다. 쓰기 버퍼 정보의 양은 플래시 메모리 크기와는 상관없이 쓰기 버퍼 블록의 수에 따라 결정되며 본 연구에서는 전체 SRAM 사용량을 고려하여 쓰기 버퍼 정보의 양을 4KB 이내로 제한하였다. 몇 번의 쓰기 요청을 처리하여 쓰기 버퍼내의 가용 페이지가 모두 사용된 경우 블록 병합 작업이 수행되며 이때 변경되는 데이타는 해당 블록에 대한 64B 이하의 사상 정보이다. 호스트 쓰기 요청 시 전송되는 데이타는 보통 소량 랜덤 쓰기의 경우는 8 섹터(4KB) 이내, 대량 순차 쓰기의 경우는 128 섹터(64KB) 이상이다[12].

비휘발성 쓰기 요구를 처리하기 위한 저장 매체로는 FRAM과 NAND 플래시 메모리가 있으며 쓰기 처리와 관련된 특성을 비교해 보면 표 2와 같다[1,4]. 쓰기 단위와 소요 시간 면에서 FRAM이 소량의 빈번한 변경에 유리함을 알 수 있다. 플래시 메모리의 경우는 기본적인 쓰기 단위가 클 뿐 아니라 덮어쓰기가 불가능한 제약으로 인해 소량 쓰기에는 비효율적이다. 그러나, 한번에 기록하는 데이타 양이 커질수록 지우기 오버헤드가 분산되어 효율성이 높아지며, 한번에 블록 크기만큼 쓰기 작업을 수행하는 경우에는 평균적인 오버헤드가 최소화된다.

표 2 FRAM, NAND 플래시 메모리 쓰기 특성 비교

	FRAM	NAND 플래시 메모리
쓰기 단위	바이트, 워드	2KB
소요 시간	100ns	200us (데이타 전송: 25ns/B)

3.1.2 비휘발성 데이타 처리

다양한 비휘발성 데이타 및 FRAM, NAND 플래시 메모리의 특성을 고려하여 각각의 데이타를 다음과 같이 처리한다. FTL 메타데이타 중 변경이 되지 않는 정보는 NAND 플래시 메모리에 기록해 두고 부팅 시 SRAM에 복사하여 사용한다. 빈번히 변경되면서 매 변경 시 소량의 데이타만이 변경되는 쓰기 버퍼 정보 및 블록 사상 정보는 FRAM에 유지한다. FTL 메타데이타에 대한 쓰기 작업을 FRAM에서 처리함으로써 NAND 플래시 메모리에서 처리할 때 필요한 지우기 작업 오버헤드가 없어지며, 플래시 메모리의 페이지 크기와는 무관하게 실제로 변경된 데이타만을 FRAM 상에서 덮어쓸 수 있다.

512B 섹터 단위로 기록되는 호스트 데이타는 NAND 플래시 메모리에 저장하며 쓰기 요청 처리 시에는 일단 작은 수의 쓰기 버퍼 블록에서 이를 처리한 뒤 이후에 블록 병합 작업을 수행한다. 쓰기 버퍼는 호스트 쓰기 패턴에 따라 소량 랜덤 쓰기와 대량 순차 쓰기를 각각 페이지 단위 쓰기 버퍼와 블록 단위 쓰기 버퍼로 나누어 처리한다. 각 쓰기 버퍼의 관리 방법에 대해서는 다음 절에서 자세히 설명한다. 그림 2는 전체적인 쓰기 요청 처리 절차를 나타낸다.

3.1.3 FRAM 사용 시의 고려 사항

SRAM과 달리 비휘발성을 갖는 FRAM에 대해 쓰기 작업을 수행할 때는 예상 외의 전원 중단 시의 복구 방법에 대한 고려가 필요하다. 워드 단위의 쓰기에 대해서는 FRAM 칩 내부적으로 원자성(atomicity)을 보장해 주지만, 여러 워드의 쓰기를 수행하던 중에 전원 공급이 중단되는 경우에는 실제 쓰기 작업이 완료된 워드와 완료되지 않고 이전 데이타를 가지고 있는 워드를 구분할 수 없게 된다. 본 논문에서는 FRAM 변경 시 변경 전, 후 데이타를 먼저 로그로 남기고 실제 변경 작업을 수행하도록 설계하였다. 전원 중단이 발생하면 다음 번 부팅 시 로그 내의 데이타와 현재 FRAM 상의 데이타를 비교하여 완전히 변경되지 않은 경우는 로그를 취소 또는 재실행시켜 주도록 하였다.

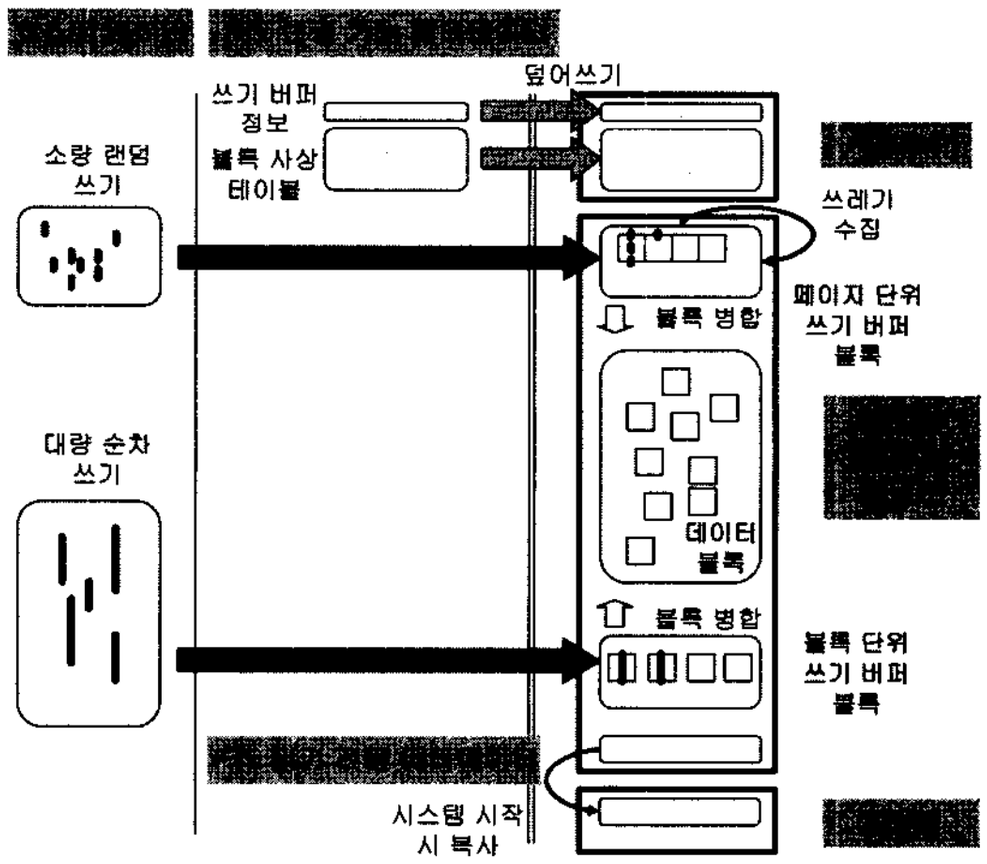


그림 2 쓰기 요청 처리 절차

3.2 쓰기 버퍼 관리 방법

호스트 쓰기 요구를 처리하는 쓰기 버퍼는 쓰기 요청 패턴에 따라 페이지 단위 사상에 기반한 쓰기 버퍼와 블록 단위 사상에 기반한 쓰기 버퍼로 구분하여 관리한다. 소량 랜덤 쓰기와 대량 순차 쓰기를 구분하는 기준으로는 하나의 쓰기 요청이 정해진 수 이상의 섹터를 쓰는 경우나 대량 순차 쓰기 패턴으로 판단된 기존의 쓰기 열(write stream)에 이어지는 경우 대량 순차 쓰기로 판단하고 그 외에는 소량 랜덤 쓰기로 판단하는

단순한 방법을 사용하였다. 그러나, 실제 쓰기 처리 후 이어진 쓰기 요청들의 패턴이 판단과 다른 경우에는 다른 쓰기 버퍼로 이동하여 처리할 수 있도록 구현하였다.

3.2.1 블록 단위 쓰기 버퍼

블록 단위 쓰기 버퍼에는 대량 순차 쓰기 요청에 의한 데이타 섹터들이 논리 섹터 번호와 동일한 물리 섹터에 쓰여지며, 블록의 중간 섹터부터 쓰여지는 경우는 쓰여지지 않은 앞부분의 섹터들은 기존의 데이타 블록에서 복사된다. 요청된 섹터가 모두 쓰여진 후에는 추가로 다음 요청에 의해 나머지 섹터들이 쓰여질 수 있으므로 아직 쓰여지지 않은 섹터들에 대한 복사 작업을 수행하지는 않는다.

블록 전체가 다 쓰여진 경우 또는 쓰기 버퍼 블록이 모두 사용되고 있을 때 새로운 대량 순차 쓰기가 발생하는 경우에는 그림 3과 같이 블록 병합 작업을 수행한다. 이미 블록 전체가 쓰여진 경우에는 논리 블록에 대한 사상을 변경하는 작업만 수행하면 되지만 아직 전체가 모두 쓰여지지는 않은 경우에는 기존 데이타 블록에서 섹터들을 복사해 오는 작업을 먼저 수행해야 한다.

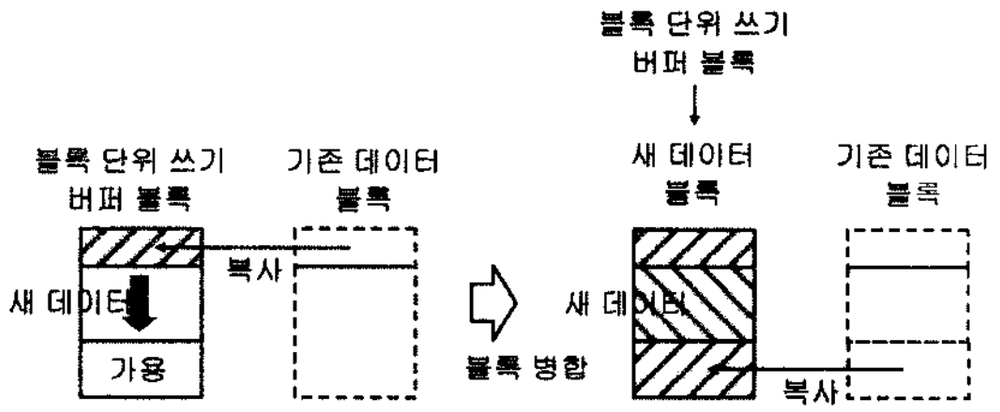


그림 3 블록 단위 쓰기 버퍼 관리(블록 병합)

블록 단위 쓰기 버퍼의 효율성은 얼마나 많은 섹터를 기존 데이타 블록에서 복사해 오는가에 의해 좌우된다. 전체 블록이 한번에 쓰여지거나 연속된 쓰기 요청에 의해 전체 블록이 새로 쓰여지는 경우 기존 데이타 블록에서 섹터를 복사해 올 필요가 없으므로 매우 효율적으로 처리된다. 반면, 일부 섹터만을 쓰게 되는 경우 또는 동일 섹터에 대한 쓰기 요청이 자주 들어오는 경우에는 대부분의 섹터를 복사해 와야 하므로 효율성이 급격히 저하된다. 이러한 상황이 발생하는 경우에는 해당 블록을 페이지 단위 쓰기 버퍼로 이동시킨다.

3.2.2 페이지 단위 쓰기 버퍼

페이지 단위 쓰기 버퍼에는 논리 블록의 구분과는 상관없이 데이타를 쓰기 요청이 들어오는 순서대로 저장한다. 이때 새로 쓰여지는 페이지가 이미 쓰기 버퍼 내에 존재하는 경우에는 기존의 페이지는 무효화 된다. 페이지 단위 쓰기 버퍼 블록들은 하나의 커다란 로그로 간주되어 가용 페이지를 순차적으로 사용하며 가용 페이지의 수가 줄어들면 빈 공간을 확보하기 위해 쓰기

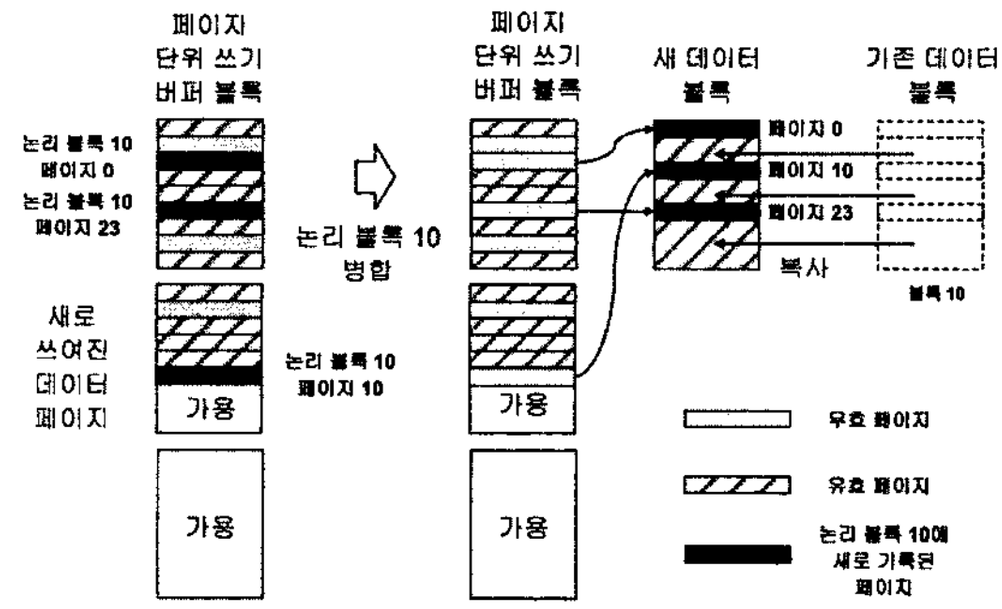


그림 4 페이지 단위 쓰기 버퍼 관리 (블록 병합)

버퍼 상태에 따라 블록 병합과 쓰레기 수집 작업을 수행한다.

페이지 단위 쓰기 버퍼 내의 유효 페이지 수가 일정 비율 이상인 경우에는 그림 4에서와 같이, 유효 페이지 수를 줄이기 위해 유효 페이지를 가장 많이 가지고 있는 블록을 선택하여 블록 병합을 수행한다. 예를 들어, 왼쪽 그림과 같은 상태에서 논리 블록 10번이 병합 대상으로 선택되었다면 10번 논리 블록의 유효 페이지들인 0번, 10번, 23번 페이지들을 페이지 단위 쓰기 버퍼로부터 나머지 페이지들은 데이타 블록으로부터 복사하여 새로운 데이타 블록을 구성한다. 블록 단위 쓰기 버퍼에서의 병합 작업과 마찬가지로 복사 작업이 모두 완료된 이후에는 블록 사상 테이블을 갱신한다. 병합이 수행되고 나면 해당 페이지들은 페이지 단위 쓰기 버퍼 내에서는 무효 상태가 된다. 병합 작업을 수행해 줌으로써 전체적인 유효 페이지 비율을 낮추어 쓰레기 수집 효율을 원하는 수준으로 유지할 수 있다. 또한, 대량 순차 쓰기 작업들을 소량 랜덤 쓰기로 잘못 판단했을 경우에는 병합 작업에 의해 페이지 단위 쓰기 버퍼에서 제외된다.

가용 페이지 수가 일정 비율 이하로 떨어지는 경우에는 그림 5와 같이 페이지 단위 쓰기 버퍼 내에서 유효 페이지가 가장 작은 블록을 선택하여 쓰레기 수집 작업을 수행한다. 예를 들어, 그림 왼쪽에서와 같이 쓰레기 수집 대상 물리 블록에 10번 논리 블록의 0번, 23번 페이지를 포함한 4개의 유효 페이지가 존재하는 경우 그림 오른쪽과 같이 가용 페이지들에 복사하고 유효 페이지가 남지 않은 페이지 단위 쓰기 버퍼 블록을 가용 블록 리스트에 반환한다.

페이지 단위 쓰기 버퍼는 유효 페이지 비율이 낮게 유지되면서 한번 쓰여진 논리 섹터들이 반복적으로 다시 쓰여져 오래된 섹터들은 대부분 무효화되는 경우에 가장 효율적으로 동작한다. 이런 경우에는 병합 작업 없이 쓰레기 수집만 수행하면 되고 쓰레기 수집 시에도

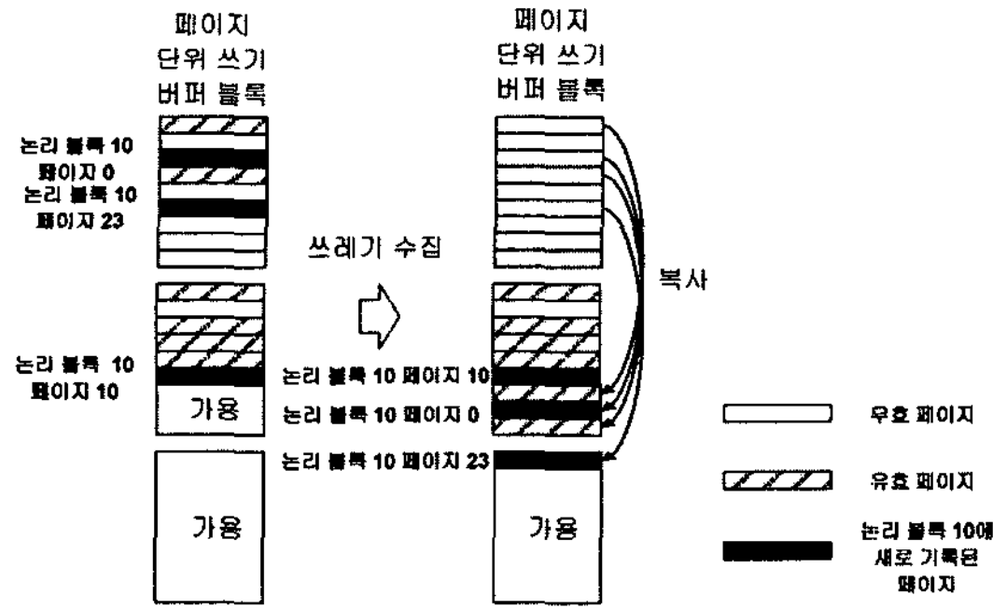


그림 5 페이지 단위 쓰기 버퍼 관리(쓰레기 수집)

대상 물리 블록 내에는 유효한 페이지가 존재하지 않아 그냥 블록 지우기만 수행되며 쓰기 처리 비용이 최소화된다. 그러나 소량 랜덤 쓰기가 이루어지는 섹터들의 집합이 커지는 경우에는 유효 페이지 비율이 높아져 오버헤드가 큰 블록 병합 작업이 수행되어야 하고 쓰레기 수집 시 복사해야 하는 유효 페이지 수도 증가하여 효율이 떨어진다.

3.2.3 읽기 요청 처리

앞에서 살펴본 바와 같은 방식으로 쓰기 요청을 처리하므로, 읽기 요청이 들어오는 경우에는 최근에 쓰여진 데이타가 블록 단위 쓰기 버퍼나 페이지 단위 쓰기 버퍼에 존재하는지 먼저 검색해야 한다. 블록 단위 쓰기 버퍼의 경우 각 쓰기 버퍼 블록들이 논리 블록 하나씩을 처리하고 있으므로 각 쓰기 버퍼 정보를 찾아보면 된다. 페이지 단위 쓰기 버퍼는 쓰기 버퍼 블록에 여러 논리 블록이 섞여 있을 수 있으므로 별도의 논리 블록 리스트를 유지하며 일단 이 리스트에서 해당 논리 블록을 찾아본 뒤 논리 블록이 페이지 단위 쓰기 버퍼에 존재한다면 해당 블록의 페이지 링크 리스트를 검색하여 필요한 페이지가 존재하는지 찾는다. 어느 쓰기 버퍼에도 존재하지 않는 것으로 판명된 페이지들은 블록 사상 테이블을 참조하여 해당 논리 블록에 사상되어 있는 데이타 블록에서 읽는다.

3.3 전체 시스템 구성

그림 6은 제안된 아키텍처의 전체적인 시스템 구성을 나타낸다. 다수의 버스를 제어하는 플래시 메모리 제어기 모듈과 호스트 인터페이스 모듈이 있고, 이 두 모듈 사이를 확장 DMA(Extended DMA) 제어 모듈이 연결한다. 시스템 버스에는 실행 코드와 데이타를 담는 SRAM과 비휘발성 저장 매체로 사용되는 FRAM이 연결된다.

플래시 메모리 제어기는 물리적 페이지 단위의 읽기/쓰기, 블록 단위의 지우기와 같은 단순한 연산이 아닌 여러 물리 블록으로 구성된 가상 블록 내의 연속된 섹

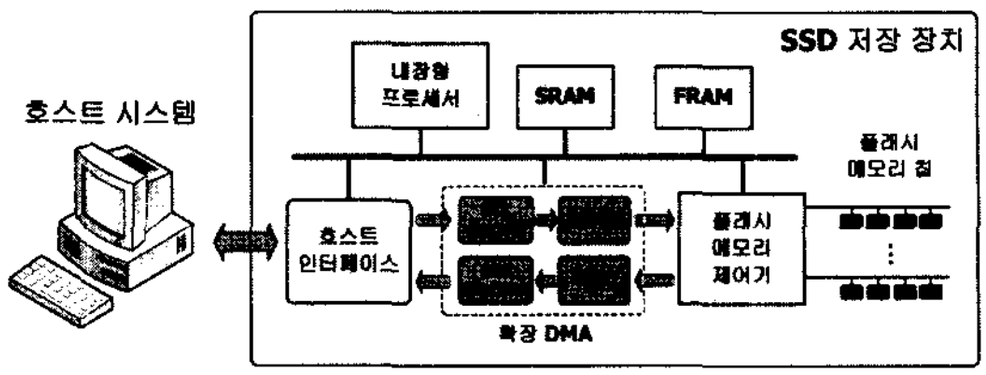


그림 6 전체 시스템 구성

터들에 대한 읽기, 쓰기 및 가상 블록 지우기 연산을 제공하는 진보된 플래시 메모리 제어기이다. 프로세서의 개입 없이 데이타를 전송하는 확장 DMA 제어기는 플래시 메모리에서 발생할 수 있는 비트 반전(bit flip) 에러를 처리할 수 있도록 에러 수정(ECC: Error Correction Code) 기능 또한 구현하고 있다.

이러한 모듈들을 통해, 프로세서는 플래시 메모리 칩의 제어, 데이타 전송과 같은 작업에 관여하지 않고 데이타 전송도 시스템 버스와는 무관하게 별도의 전송 경로를 이용하므로, 느린 속도로 동작하는 프로세서와 시스템 버스로 인한 병목 현상을 막을 수 있다. 대역폭 측면에서, 플래시 메모리 버스를 다수 사용함으로써 단일 칩 처리율 제한을 극복하여 버스 수만큼의 대역폭을 얻을 수 있다.

FTL은 기본 기능인 사상 관리뿐 아니라 몇 가지 플래시 메모리 특성에 따른 추가 기능들을 필요로 한다. 덮어쓰기가 불가능하다는 제약과 함께 NAND 플래시 메모리의 큰 제약은 블록마다 수행할 수 있는 지우기 작업 회수가 최대 10^6번 정도로 제한된다는 점이다. 이 회수를 초과하거나 또는 때에 따라서 초과하지 않더라도 지우기나 쓰기 작업이 정상적으로 수행되지 않는 배드 블록(bad block)이 될 수 있다. 이러한 문제를 해결하기 위해 특정 블록에 지우기가 집중되는 것을 막는 마모 균등화(wear leveling)와 배드 블록이 발생하는 경우 이를 처리할 수 있는 기능이 필요하다.

본 논문에서는 마모 균등화를 위해 두 가지 방법을 함께 사용하였다. 호스트 쓰기 요청 처리 시에는 가장 지우기가 적게 수행된 쓰기 버퍼 블록을 선택하여 사용함으로써, 쓰기 버퍼 블록들과 호스트 쓰기가 활발하게 진행되는 데이타 블록들 간의 지우기 회수를 균등화하였다. 또한, 유휴(idle) 시간을 이용, 호스트 쓰기 요청이 들어오지 않아서 지우기가 거의 수행되지 않은 데이타 블록을 지우기가 많이 수행된 쓰기 버퍼 블록과 교환하는 방법을 사용하여 전체적인 지우기 회수를 균등화하였다. 배드 블록 처리를 위해서는 지우기나 쓰기 에러 발생 시 해당 블록을 시스템 초기화 시 미리 확보해 두었던 정상 블록과 교체하는 작업을 수행한다.

4. 구현 및 성능 평가

제안된 아키텍처를 FPGA 기반 프로토타입 평가 플랫폼을 이용하여 구현하였으며, 성능 평가를 위해 PC 환경에서 사용되는 대표적인 벤치마크 프로그램인 PCMark04 HDD 벤치마크[17]를 사용하였다.

4.1 프로토타입 구현

프로토타입 구현을 위해 PowerPC 프로세서를 내장한 Xilinx Virtex IV FPGA 칩에 기반한 그림 7과 같은 평가 보드를 제작하였다. 보드 상에는 각 플래시 메모리 버스에 해당하는 플래시 메모리 모듈을 위한 4개의 슬롯이 있으며, 아래쪽으로 FRAM 모듈을 위한 슬롯이 있고 왼쪽으로는 PATA 호스트 인터페이스 커넥터가 있다. 사용한 플래시 메모리 모듈에는 1GB(8Gb)의 플래시 메모리 칩이 4개 장착되며 4개의 모듈을 사용하여 동시에 동작할 수 있는 플래시 메모리 칩의 수는 16개, 전체 플래시 메모리 용량은 16GB이다. FRAM 모듈은 1MB(8Mb) 용량의 FRAM 칩이 2개 장착되어 총 2MB이다. PATA 인터페이스는 UDMA100(100MB/s) 모드까지 지원한다.

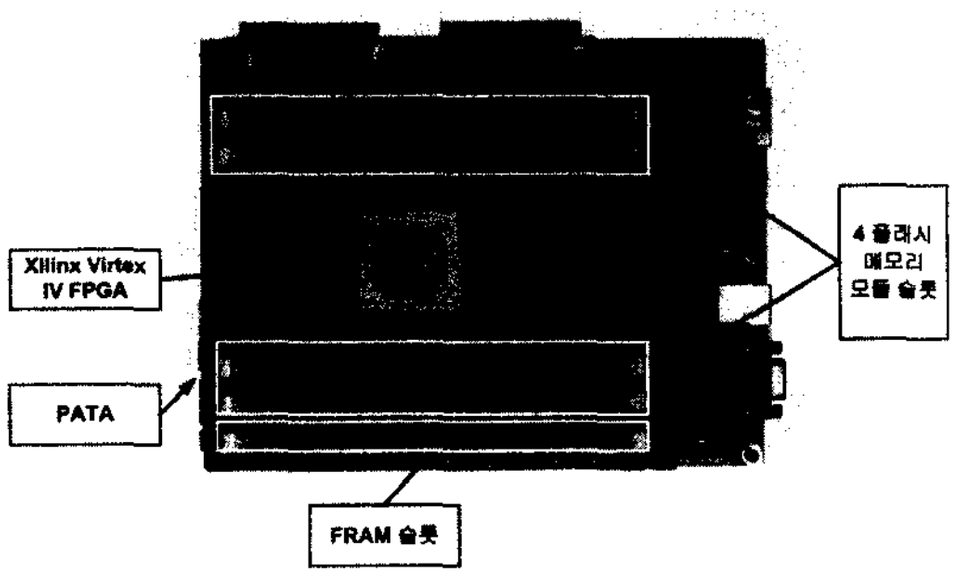


그림 7 평가 보드

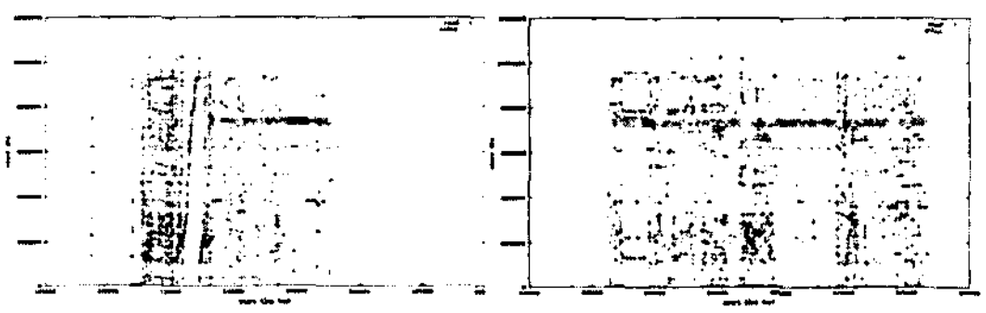
플래시 메모리 제어기 모듈과 호스트 인터페이스 모듈, 확장 DMA 모듈 등은 FPGA 상에서 구현하였으며, FTL 소프트웨어가 사용하는 SRAM 주 메모리는 64KB이다. 보드에 장착된 FRAM 2MB 중 실제 사용된 FRAM은 328KB이다. 블록 사상 테이블을 저장하기 위해 사용된 FRAM은 320KB(사상 정보 256KB, 지우기 회수 64KB)이며 이 값은 전체 플래시 메모리 용량에 비례한다. 즉, 본 연구에서 제안한 아키텍처에서는 플래시 메모리 16GB당 FRAM 320KB를 필요로 하며 용량이 증가하면 FRAM 요구량은 따라서 증가한다.

쓰기 버퍼와 관련하여 대량 순차 쓰기 분류 기준으로는 128섹터(64KB)를 사용하였다. 또한, 블록 단위 쓰기 버퍼로는 8MB(쓰기 열 개수 4), 페이지 단위 쓰기 버퍼로는 32MB를 사용하였다.

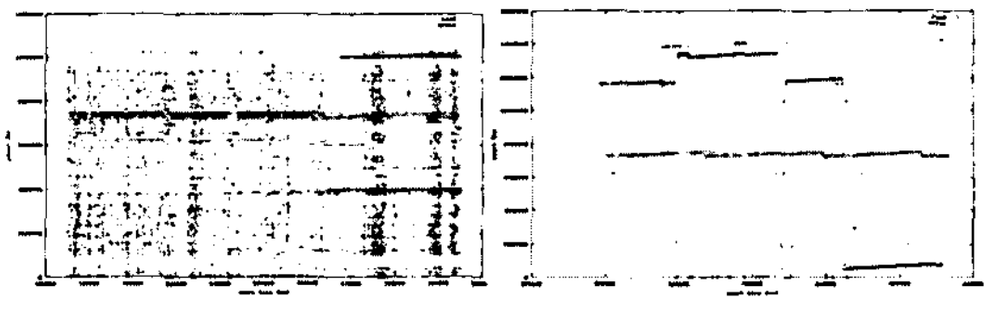
4.2 성능 평가 방법

PCMark04 벤치마크 프로그램의 워크로드는 XP Startup, Application Loading, General HDD Usage, File Copying의 네 종류로 구성되어 있다. XP Startup은 Windows XP 운영체제가 부팅될 때 발생하는 워크로드이고 Application Loading은 Microsoft Word, Acrobat Reader, Windows Media Player 등의 응용 프로그램을 시작 시키거나 종료할 때 발생하는 워크로드, General HDD Usage는 Word, WinZip, WinAmp, Internet Explorer, Picture Viewer 등의 응용 프로그램을 실행하는 과정에서 발생하는 워크로드, File Copying은 400MB 정도의 파일들을 복사할 때 발생하는 워크로드이다. 각 워크로드의 접근 패턴을 그래프로 나타내면 그림 8과 같다. 각 그래프에서 X축은 요청 번호를 Y축은 논리 섹터 번호를 나타낸다. 전체적으로 XP Startup, Application Loading, General HDD Usage에서는 랜덤 패턴이 대부분이며, 반대로 File Copying에서는 순차 패턴을 보이고 있다.

성능 비교 대상으로는 7200RPM의 회전 속도를 갖는 SATA2 인터페이스(300MB/s)의 하드디스크(Seagate Barracuda 7200.10 ST3250620AS)와 상용으로 판매되는 PATA 인터페이스(UDMA100 전송 모드: 100MB/s)를 갖는 2가지 종류의 SSD(Adtron I35FB, M-Systems FFD UATA)를 선정 하였다. 벤치마크를 수행한 PC 시스템은 표 3과 같다.



(a) XP Startup (b) Application Loading



(c) General HDD Usage (d) File Copying

그림 8 벤치마크 워크로드 패턴

표 3 성능 측정 환경

CPU	Intel Pentium 4 (2.4GHz)
주 메모리	1GB
메인 보드 칩셋	Intel i945P
ATA 제어기	Intel ICH7R
운영체제	Windows XP Professional (Version 5.1.2600)

4.3 결과

4.3.1 하드디스크 및 SSD와의 성능 비교

그림 9의 그래프는 각 워크로드에 대해 워크로드의 요청들을 처리하는데 소요된 시간과 전송된 데이타 양을 이용해 처리율로 계산한 결과를 보여주고 있다. X 축은 워크로드 및 비교 대상들을 Y 축은 MB/s 단위의 처리율을 나타낸다. 결과를 보면 랜덤 요청이 많은 워크로드에서는 구현된 프로토타입의 성능이 비교 대상들에 비해 3배 이상 높은 것을 볼 수 있다. 순차 요청이 많은 File Copying 워크로드에서는 하드디스크와 대등한 성능을 보였으나 비교 대상들의 성능은 File Copying시의 전송률에 비해 다른 워크로드에서는 크게 저하된다. 이에 비하여 프로토타입의 경우는 패턴 별로 최적화된 쓰기 버퍼를 두는 방식에 의해 워크로드에 의한 영향이 크지 않고 거의 비슷한 성능을 나타내는 것을 볼 수 있다.

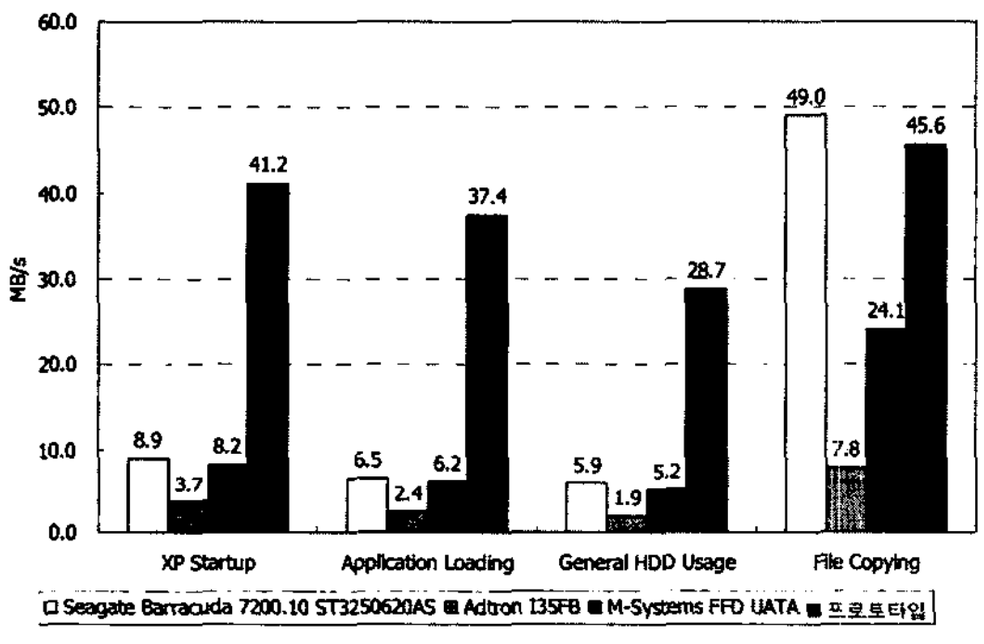


그림 9 성능 측정 결과

4.3.2 FRAM 사용 여부에 따른 성능 분석

FRAM을 사용하는 것이 성능 향상에 얼마나 기여하였는지를 분석하기 위해 FRAM을 사용하지 않고 NAND 플래시 메모리로 FRAM 메타데이타 읽기, 쓰기를 대신 처리하는 FTL을 구현하였다. 표 4는 각 경우의 실험 결과를 보여준다. FRAM 사용에 따른 성능 향상은 워크로드마다 조금씩 차이가 나긴 하지만, 전체적으로 18% 정도의 성능 향상이 있었음을 볼 수 있다. File Copying 워크로드의 경우에는 요청 당 섹터 수가 커서 메타데이타 접근 회수가 상대적으로 적으므로 FRAM에 의한 성능 향상이 다른 워크로드에 비해 적었다.

표 4 FRAM 사용 여부가 성능에 미치는 영향

	FRAM 미사용	FRAM 사용	성능 향상
XP Startup	34.7MB/s	41.2MB/s	18.7%
Application Loading	31.6MB/s	37.4MB/s	18.4%
General HDD Usage	24.0MB/s	28.7MB/s	19.6%
File Copying	41.5MB/s	45.6MB/s	9.9%

5. 결 론

본 논문에서는 플래시 메모리 기반 저장 장치에서 비휘발성을 갖도록 유지해야 하는 정보들을 분류하여, FRAM과 NAND 플래시 메모리에 나누어 유지함으로써 성능과 경제성 양쪽의 장점을 함께 갖춘 새로운 비휘발성 저장 장치 계층 구조를 제안하였다. 또한, 호스트 쓰기 요청을 패턴에 따라 소량 랜덤 쓰기와 대량 순차 쓰기로 구분하여 각각에 가장 적합한 관리 방법을 적용함으로써 양쪽 쓰기 패턴 모두를 효율적으로 처리할 수 있는 쓰기 버퍼 관리 방법을 제안하였다.

평가 보드에서 프로토타입을 구현하여 성능을 평가해 본 결과 매 호스트 요청 처리 시마다 접근하게 되는 FTL 메타데이타 관리 오버헤드를 현저하게 줄일 수 있었고, 쓰기 요청 패턴에 무관하게 특히 소량 랜덤 쓰기에 있어서도 순차 쓰기 시의 성능에 근접한 성능을 얻을 수 있음을 보였다.

향후 연구로서 플래시 메모리 용량과는 상관없이 사용 가능한 FRAM의 크기가 작아지거나 커지는 경우 FRAM의 활용 방법에 대한 연구를 계획하고 있다. 본 연구에서 제안한 아키텍처는 사상 테이블 전체를 FRAM에 항상 유지해야 하는 구조이므로, FRAM의 크기가 사상 테이블보다 작은 경우에는 본 연구의 아키텍처를 적용할 수 없다. 따라서, 전체 사상 테이블을 FRAM에 유지하지 않고 일부 또는 전체 테이블을 NAND 플래시 메모리에 유지하고 그 위치 정보만을 FRAM에 유지하는 아키텍처에 대한 연구가 필요하다. 이를 통해 FRAM 요구량은 최소화하면서도 충분한 성능 향상 효과를 얻을 수 있는 최적의 FRAM 요구량을 도출할 수 있다. 더 나아가 여분의 FRAM을 호스트 데이타를 저장하는 용도로 활용함으로써 쓰기 성능을 더욱 향상 시킬 수 있는 아키텍처에 대한 연구도 필요하다.

참 고 문 헌

[1] Samsung Electronics, NAND flash memory datasheets, http://www.samsung.com/.

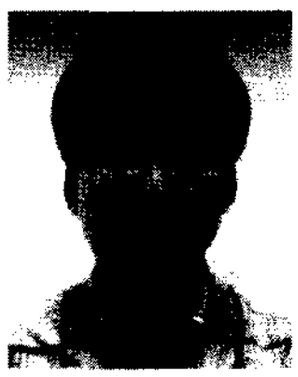
[2] Gal, E. and Toledo, S., "Algorithms and data structures for flash memories," ACM Computing Surveys, Vol.37, No.2, pp. 138-163, 2005.

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

[4] Ramtron, FRAM datasheets, http://www.ramtron.com/.

[5] Rosenblum, M. and Ousterhout, J., "The design and implementation of a log-structured file system," ACM Transactions on Computer Systems,

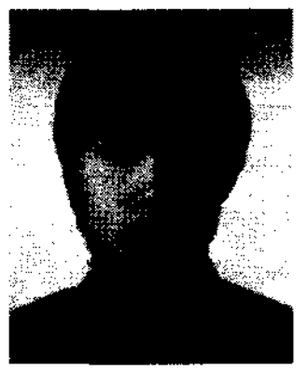
Vol.10, No.1, pp. 26-52, 1992.
[6] Wu, M. and Zwaenepoel, W., "eNVy: a non-volatile, main memory storage system," in Proceedings of the Sixth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-6), pp. 86-97, 1994.
[7] Kawaguchi, A., Nishioka, S., and Motoda, H., "A flash-memory based file system," in Proceedings of the USENIX 1995 Winter Technical Conference, pp. 155-164, 1995.
[8] Chiang, M.-L., Lee, P. C. H., and Chang, R.-C., "Using data clustering to improve cleaning performance for flash memory," Software: Practice and Experience, Vol.29, No.3, pp. 267-290, 1999.
[9] Kim, J. M., Kim, J. M., Noh, S. H., Min, S. L., and Cho, Y., "A space-efficient flash translation layer for CompactFlash systems," IEEE Transactions on Consumer Electronics, Vol.48, No.2, pp. 366-375, 2002.
[10] Lee, S.-W., Park, D.-J., Chung, T.-S., Lee, D.-H., Park, S., and Song, H.-J., "A log buffer-based flash translation layer using fully associative sector translation," ACM Transactions on Embedded Computing Systems, Vol.6, No.3, 2007.
[11] Wu, C.-H. and Kuo, T.-W., "An adaptive two-level management for the flash translation layer in embedded systems," in Proceedings of the 2006 IEEE/ACM International Conference on Computer Aided Design (ICCAD '2006), pp. 601-606, 2006.
[12] Chang, L.-P. and Kuo, T.-W., "Efficient management for large-scale flash-memory storage systems with resource conservation," ACM Transactions on Storage, Vol.1, No.4, pp. 381-418, 2005.
[13] Miller, E. L., Brandt, S. A., and Long, D. D. E., "HeRMES: High Performance Reliable MRAM-Enabled Storage," In Proceedings of the 8th IEEE Workshop on Hot Topics in Operating Systems (HotOS-VIII), pp. 83-87, 2001.
[14] Wang, A. A., Kuenning, G. H., Reiher, P., and Popek, G. J., "Conquest: better performance through a disk/persistent-RAM hybrid design," ACM Transactions on Storage, Vol.2, No.3, pp. 309-348, 2006.
[15] Doh, I. H., Choi, J., Lee, D., and Noh, S. H., "Exploiting non-volatile RAM to enhance flash file system performance," in Proceedings of the 7th ACM & IEEE International Conference on Embedded Software, pp. 164-173, 2007.
[16] Aleph One, "YAFFS: Yet another Flash file system", http://www.aleph1.co.uk/yaffs/.
[17] Futuremark Corporation, "PCMark04 white paper," http://www.futuremark.com/.



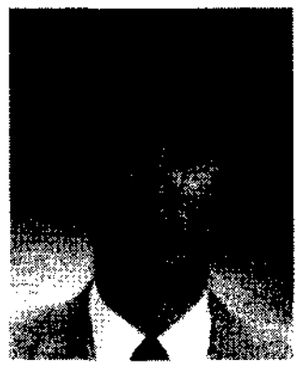
윤 진 혁
1999년 서울대학교 컴퓨터공학부 졸업(학사). 2001년 서울대학교 전기컴퓨터공학부 졸업(석사). 2008년 서울대학교 전기컴퓨터공학부 졸업(박사). 2008년~ 엠트론스토리지테크놀로지㈜. 관심분야는 운영체제, 내장형 시스템, 플래시 메모리 기반 저장장치 시스템

남 이 현
1998년 서울대학교 전기공학부 공학사 1998년~2002년 콤텍 시스템 기술연구소. 2002년~2004년 퓨처 시스템 기술연구소 2005년~현재 서울대학교 전기컴퓨터 공학부 석박사 통합과정. 관심분야는 OS, Computer Architecture, HW/SW 통합 설계 등

성 윤 재
2005년 서울대학교 컴퓨터공학부 졸업(학사). 2007년 서울대학교 전기컴퓨터공학부 졸업(석사). 2007년~서울대학교 전기컴퓨터공학부 박사과정. 관심분야는 컴퓨터구조, 내장형 시스템, 플래시 메모리 기반 저장장치 시스템



김 홍 석
2005년 서울대학교 전기컴퓨터공학부 학사 졸업. 2008년 현재 서울대학교 전기컴퓨터공학부 석박사 통합과정. 관심분야는 임베디드 시스템, 저장장치 시스템, 플래시 메모리 소프트웨어



민 상 렬
1983년 서울대학교 전자계산기공학과 공학사. 1985년 서울대학교 전자계산기공학과 공학석사. 1989년 워싱턴대학교 전산학과 박사. 1989년~1990년 IBM Watson 연구소 객원 연구원. 1990년~1992년 부산대학교 컴퓨터공학과 조교수. 1992년~현재 서울대학교 컴퓨터공학부 교수. 관심분야는 컴퓨터구조, 컴퓨터성능평가, 실시간시스템, 내장형시스템 등

조 유 근

1971년 서울대학교 졸업(학사). 1978년 미국 미네소타 대학교 컴퓨터 과학 박사 졸업. 1985년 미국 미네소타 대학교 방문 교수 2001년 한국 정보과학회 회장. 1979년~현재 서울대학교 컴퓨터공학부 교수. 관심분야는 운영체제, 알고리즘, 시스템 보안, 결함 허용 컴퓨팅