바이트 접근성을 가지는 비휘발성 메모리 소자를 이용한 낸드 플래시 파일 시스템의 부팅시간 개선 기법

(Boosting up the Mount Latency of NAND Flash File System using Byte-addressable NVRAM)

전 병 길 † 김 은 기 ††
(Byeonggil Jeon) (Eunki Kim)

신 형 종 † 한 석 희 ††
(Hyungjong Shin) (Seokhee Han)

원 유 집 †††
(Youjip Won)

요 약 본 연구의 목표는 낸드 플래시 메모리용 파일 시스템의 고질적인 문제점인 파일 시스템 마운트 지연시간을 개선하는 것이다. 이를 위하여 바이트 접근성을 가지는 비휘발성 랜덤 억세스 메모리 소자와 낸드 플래시로 계층적 저장시스템을 구성하고, 바이트 접근성을 가지는 비휘발성 랜덤 억세스 메모리 소자 상에 파일 시스템 메타 데이타를 위치시키는 기법을 개발하였다. 데이타 무결성을 지원하고, 고속의 랜덤 읽기/쓰기가 가능한 비휘발성 랜덤 억세스 메모리 소자에 메타 데이타를 위치시킴으로써 낸드 플래시에 저장된 메타 데이타를 관리하기 위해 Spare 영역과 Object Header 영역에 구성된 항목가운데 불필요한 항목을 제거할 수 있었다. 또한, Object Header의 자료구조를 효율적으로 설계하여 이전 연구인 FRASH1.0에서 파일 생성/삭제 성능 저하를 초래한 Object Header를 스캔하는 단계를 제거하였다. 최적화된 비휘발성 랜덤 억세스 메모리 소자의 자료구조와 개발된 알고리즘을 실제 파일 시스템으로 구현한 FRASH1.5를 개발하였다. 이것을 채용한 FRASH1.5의 성능을 평가한 결과 마운트 동작 시간은 이전 연구인 FRASH1.0 대비 2배 이상 단축시킬 수 있었고, 파일 생성 성능은 FRASH1.0 대비 3~8% 향상되었다. 특히, 파일 크기가 크고, 개수가 많을 경우는 기존 YAFFS 대비 성능 저하 없이 마운트 시간을 8배 이상 감소시킬 수 있었다.

키워드 : 파일시스템, 낸드 플래시, 마운트, NVRAM, FRAM

Abstract This paper describes an improvement of mount-time delay in NAND Flash file systems. To improve file system mount performance, this work configures a hierarchical storage system with byte-addressable NVRAM and NAND Flash memory, and let the meta data of a file system allocated in the NVRAM. Since the meta data are stored in NVRAM supporting data integrity, some of the items, which are stored in Spare area and Object Header area of NAND Flash memory to control meta data of NAND Flash file system, could be eliminated. And also, this work eliminates the scanning operation of the Object Header area of previous work FRASH1.0. The scanning operation is definitely required to find out the empty Object Header address for storing the Object Header data and provokes a certain amount of performance loss in file generation and deletion. In this work, an implemented file system, so-called FRASH1.5, is demonstrated, featuring new data structures and new algorithms. The mount time of FRASH1.5 becomes twice as fast as that of the FRASH1.0. The performance in file generation gets improved by about 3~8%. In particular, for most large-size files, the FRASH1.5 has 8 times faster mount time than YAFFS, without any performance loss as seen in the file generation.

Key words : File System, NAND Flash, mount, NVRAM, FRAM

1. 서 론

오늘날 휴대용 멀티미디어 장치의 저장 매체로서 가장 각광 받고 있는 낸드 플래시 메모리는 기존의 하드 디스크에 비해 성능이 뛰어나고, 크기가 작으며, 충격에 강한 장점을 가지고 있다. 따라서 이들 낸드 플래시 메모리 저장 장치는 디지털 카메라, 휴대폰, 휴대용 오락기, 노트북, 등에 사용되고 있다. 그런데 낸드 플래시 메

· 본 연구는 과학재단의 국가지정연구실 사업(ROA-2007-000-20114-0)에 의해 지원을 받았습니다.
· 이 논문은 제34회 추계학술대회에서 '바이트 접근성을 가지는 비휘발성 메모리 소자를 이용한 낸드 플래시 파일 시스템의 부팅시간 개선 기법'의 제목으로 발표된 논문을 확장한 것임

† 학생회원 : 한양대학교 전자컴퓨터통신
ugfman@ece.hanyang.ac.kr
newbell@ece.hanyang.ac.kr
shhan@ece.hanyang.ac.kr
†† 비 회 원 : 한양대학교 전자컴퓨터통신
zerobit@ece.hanyang.ac.kr
††† 종신회원 : 한양대학교 전자컴퓨터통신 교수
yjwon@ece.hanyang.ac.kr

논문접수 : 2007년 12월 7일
심사완료 : 2008년 3월 5일

Copyright@2008 한국정보과학회 : 개인 목적이나 교육 목적인 경우, 이 저작물의 전체 또는 일부에 대한 복사본 혹은 디지털 사본의 제작을 허가합니다. 이 때, 사본은 상업적 수단으로 사용할 수 없으며 첫 페이지에 본 문구와 출처를 반드시 명시해야 합니다. 이 외의 목적으로 복제, 배포, 출판, 전송 등 모든 유형의 사용행위를 하는 경우에 대하여는 사전에 허가를 얻고 비용을 지불해야 합니다.

정보과학회논문지: 컴퓨팅의 실제 및 레터 제14권 제3호(2008.5)

모리 장치의 고유 특성인 지우기(erase) 동작 및 제한적인 지우기 횟수, 지우기 후 쓰기 특성 등으로 인하여 DRAM이나 SRAM과 같이 직접 메모리 버스에 연결하여 사용하지 못하는 등의 이유로 소프트 웨어의 도움이 필수적으로 요구 된다. FTL[1], YAFFS[2], JFFS[3]이 그 대표적인 것으로, FTL(Flash Translation Layer)은 매 쓰기 동작 시 마다 블록 장치 주소를 낸드 플래시 메모리의 새로운 위치로 동적으로 할당하고, 이를 주소 맵(map)을 통하여 구동하는 드라이버 레이어(driver layer)로써 낸드 플래시 메모리 저장장치를 하나의 블록 장치로 구성하여 읽기/쓰기 동작을 제공한다. YAFFS (Yet Another Flash Filing System)나 JFFS(Journaling Flash File System)는 하드 디스크에서 사용하고 있는 LFS(Log- Structured File System)를 도입하여 매 쓰기 동작시 새로운 위치에 기입을 하고 그것의 기록을 저장하는 방식으로 낸드 플래시 메모리를 구동한다. YAFFS나 JFFS와 같은 LFS 구조는 성능 면에서는 뛰어나지만, 낸드 플래시 메모리 저장장치를 사용 할 때는 반드시 DRAM에 메타 데이타(meta data)를 구성하기 위하여 낸드 플래시 메모리 전체 영역을 스캔(scan)하여야 하기 때문에 마운트 시간이 많이 소모된다는 단점이 있다. 이것은 낸드 플래시 메모리 저장장치의 용량이 증가하면 할수록 더 많은 마운트 시간을 필요로 하기 때문에 반드시 개선되어야 할 부분이다. 따라서 이전 연구 에서는 낸드 플래시 메모리 저장장치의 빠른 마운트 시간을 위하여 NVRAM(Non-volatile Random Access Memory)인 FRAM(Ferro-electric RAM)을 채용한 파일 시스템인 FRASH[4](본 논문에서는 FRASH1.0으로 명명함)를 개발하였다. 본 연구에서는 FRASH1.0을 개발 하면서 발생한 파일의 생성과 삭제의 성능 저하를 개선하기 위하여 FRAM에 저장되는 메타 데이타를 효율적으로 관리 할 수 있는 최적의 NVRAM 자료구조를 도입하여 성능 저하를 최소화하고, 또한, FRAM을 사용함에 따라 불필요한 기능들을 제거함으로써 마운트 시간도 FRASH 1.0 대비 개선을 한 FRASH1.5를 구현하였다.

본론으로 들어가기 전에 본 논문에서 채용한 FRAM에 대해서 간략히 기술 하겠다. 비휘발성, 저 소비전력, 고속 읽기/쓰기, 랜덤 억세스(Random Access), 높은 개서회수(endurance)등의 특성을 갖는 FRAM은 그림 1에 도시된 것과 같이 메모리 셀이 하나의 트랜지스터와 하나의 강유전체 커패시터(Ferro-electric Capacitor)로 구성이 되어있다. FRAM의 메모리 셀 구조는 DRAM과 거의 유사하기 때문에 빠른 읽기/쓰기가 가능하며, DRAM과 달리 리프레시(Refresh)는 필요없다. 본 연구에서는 삼성전자가 개발한 64Mb(8MByte) FRAM[6]을 채용하였다.

본 논문은 서론, 본론, 결론, 그리고 참조문헌으로 구성되었으며, 본론은 관련 연구로써 FRASH1.0을 소개하고, 이들의 성능을 개선하기 위한 새로운 자료구조의 개념과 동작 방법, 그리고 이 자료구조를 채용한 FRASH1.5의 평가 순으로 기술 하였다.

2. 본 론

2.1 관련 연구

이전 연구인 FRASH1.0은 그림 2에 도시된 것과 같이 낸드 플래시 메모리의 Spare 영역과 Object Header 영역을 FRAM에 지정된 Tag 영역과 Object Header 영역에 각각 저장을 하였다. FRAM에 저장된 Tag는 낸드 플래시 메모리의 데이타 영역과 일대일로 대응되지만, Object Header는 파일의 수에 따라 가변적이기 때문에 각각 Tag에 추가적인 Object Header Pointer 영역을 생성하여 Tag의 Chunk Id가 "0"일 때 Object Header Pointer가 Object Header 영역을 지정하도록 구성하였다. 이렇게 구성된 FRASH1.0의 마운트는 FRAM에 저장된 Tag 정보를 읽고, Tag내의 Chunk Id가 "0"이면 Object Header Pointer를 읽어서 그것이 지정하는 Object Header 영역으로 이동하여 Object Header 정보를 읽는 과정을 통하여 DRAM에 파일 정보를 구성 하였다. FRASH1.0에서 FRAM에 저장된 Tag의 구동은 낸드 플래시의 Tag 구조를 그대로 복사한 후 낸드 플래시의 Tag를 구동하는 방법과 동일하게 진행하였다. 따라서 이 부분을 FRAM에 적합하게 최적화할 필요가 있다. 파일 생성은 Tag에 정보를 저장하고, Object Header

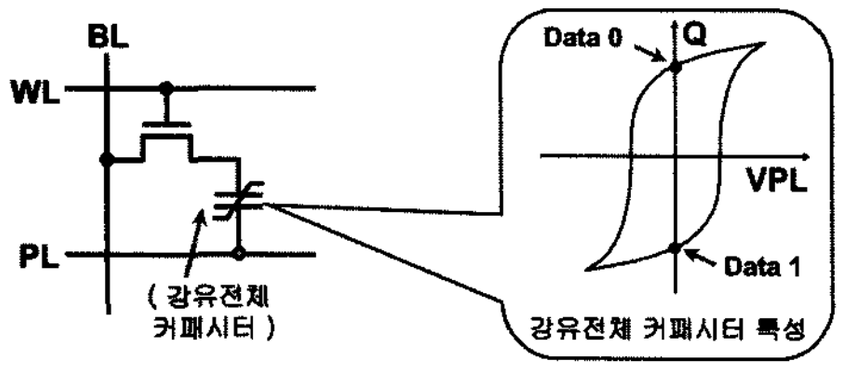


그림 1 FRAM 셀 구조

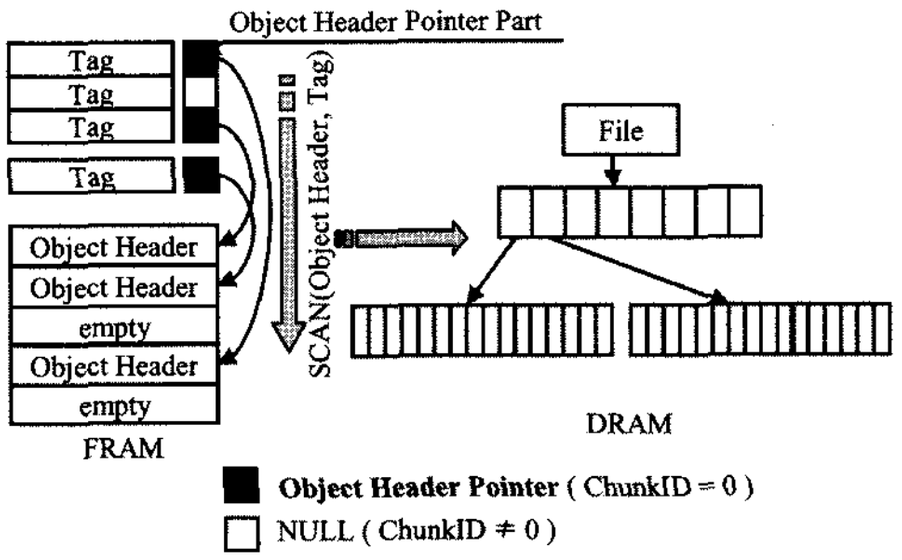


그림 2 FRASH1.0 구조

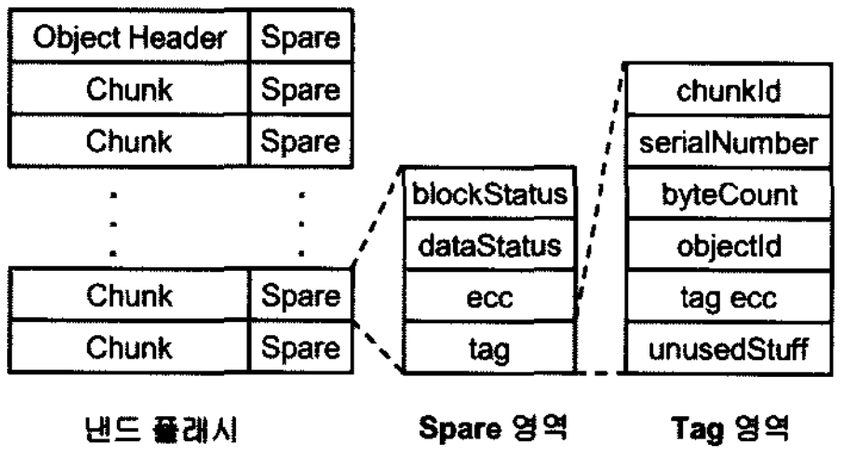
를 생성하기 위하여 Object Header 영역을 순차적으로 검색하여 빈 공간(empty)이 검색되면 여기에 Object Header 정보를 저장하고, Object Header Pointer가 이 Object Header 영역을 가리키도록 하는 과정으로 진행된다. 파일 제거는 Tag, Object Header Pointer를 제거하고, 해당하는 Object Header 영역을 삭제한 후 빈 공간으로 지정한다. FRASH1.0의 마운트 시간은 기존 YAFFS 대비 5배 정도 감소하였고, 파일 크기가 클수록 더 많은 마운트 시간을 줄일 수 있었다. 그러나 성능을 측정한 결과 파일 생성 성능은 기존 YAFFS 대비 5~10%, 그리고 파일 삭제 성능은 3~6%가 떨어졌다. 이것은 메타 데이타가 저장된 FRAM을 관리하는 구조가 유발시키고 있는 성능 저하 이다.

2.2 메타 데이타 관리를 위한 NVRAM 자료 구조 개선

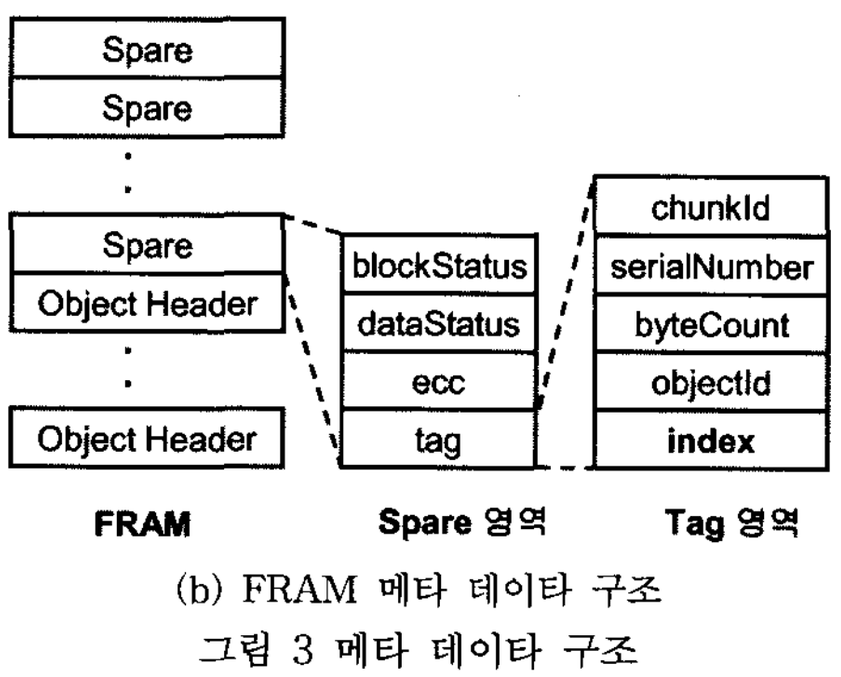
FRASH1.0에서의 성능 저하를 개선하기 위해 본 연구에서는 다음 두 가지 측면에서 접근 하였다.

첫 번째로 FRAM을 사용함으로써 불필요해지는 내용을 제거하여 메모리 사용 효율을 증가 시키고, Workload를 감소시키는 측면에서 접근하였고, 두 번째로 FRASH1.0에서 Object Header를 저장하기 위한 빈 공간을 찾는 Workload를 감소시키는 측면에 중점을 두어 본 연구를 진행 하였다.

이 두 가지 항목을 구현하기 위하여 FRAM에 메타 데이터를 관리하는 새로운 자료구조의 도입이 필요하다. 먼저 기존 YAFFS에서 구성된 메타 데이타의 구조는 그림 3의 (a)에 나타나 있다. 무결성을 지원하는 스페어 영역은 더 이상 낸드 플래시가 아닌 FRAM에 저장되므로 더 이상 tag ecc를 수행할 필요가 없다. 때문에 tag ecc로 할당된 부분은 다른 영역으로 활용할 수 있다. 본 연구에서는 tag ecc 영역을 Object Header의 위치를 지정하는 index로 활용하였다. 그림 3의 (b)는 본 연구에서 FRAM에 구성된 메타 데이타 구조를 나타낸다.



(a) 낸드 플래시 메타 데이타 구조



(b) FRAM 메타 데이타 구조

그림 3 메타 데이타 구조

다음으로는 FRAM에 메타 데이타를 저장/삭제 시 Object Header 영역을 효율적으로 관리할 수 있는 새로운 자료 구조를 도입하였다. 그림 4는 FRAM에 메타 데이타를 최적으로 관리할 수 있는 자료 구조를 도시한 것이다. FRASH1.0의 Object Header Pointer Part를 제거하고, 삭제된 tag ecc 공간을 Object Header를 지정하는 index로 설정하였다. 따라서 Object Header

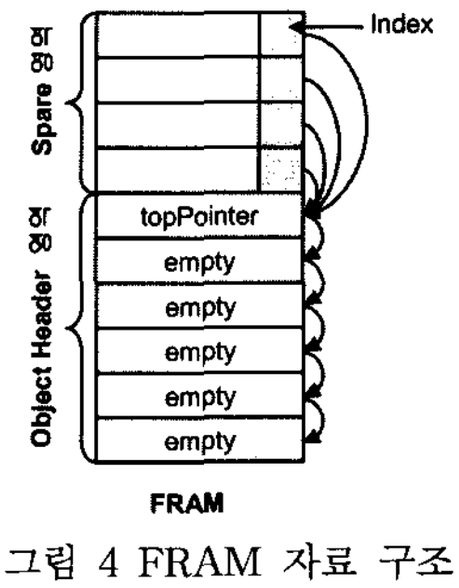


그림 4 FRAM 자료 구조

Pointer Part를 읽어야 하는 Workload를 한 단계 제거하였을 뿐만 아니라 FRAM 사용 공간도 절약할 수 있다. 이 index는 Object Header를 지정하고 있지 않을 경우에는 항상 Object Header 영역의 topPointer를 지정하게 구성되어 있다. Object Header 영역은 시작 주소를 topPointer로 지정하였다. 이 주소 공간에는 항상 빈 Object Header 주소를 지정하고 있다. topPointer가 지정하고 있는 빈 Object Header(empty)는 다음 빈 Object Header(empty)를 지정하고 있다. 마지막 빈 영역은 다음이 없음을 표시하고 있다. 이렇게 빈 Object Header들은 topPointer를 기준으로 모두 연결되어 있으며, 빈 Object Header가 Object Header 영역의 어디에 위치하든 관계없이 topPointer에서 한번의 Workload로 지정될 수 있다.

2.3 NVRAM 자료 구조 동작 방법

FRAM에 적합한 새로운 자료 구조를 기존 FRASH1.0에 채용하여 FRASH1.5를 구현하였다. 이것의 마운트, 파일 생성, 파일 삭제 과정을 하나씩 설명하도록 하겠다.

마운트: 그림 5는 FRASH1.5의 마운트 과정을 도시한 것으로 FRASH1.0과 유사하지만 Object Header를 찾기 위하여 Object Header Pointer를 추가로 읽어야

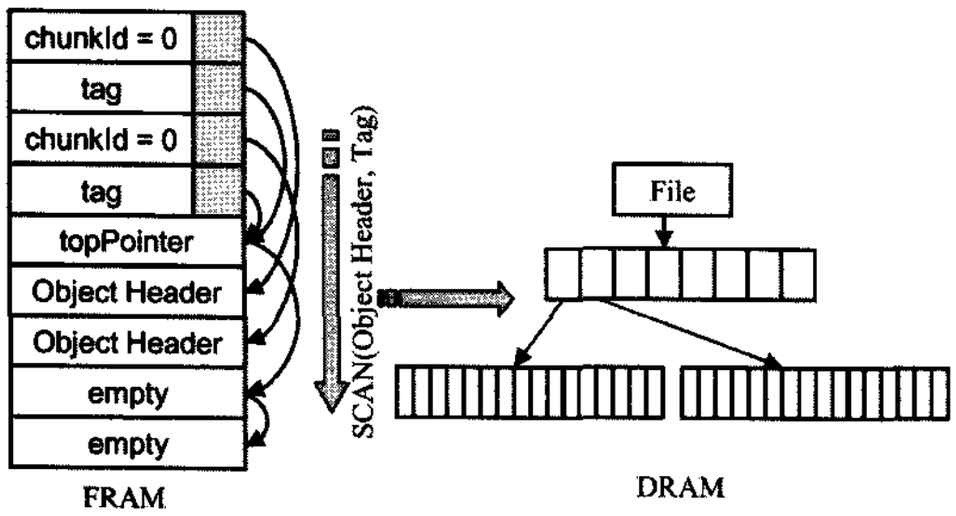


그림 5 FRASH1.5 마운트 동작

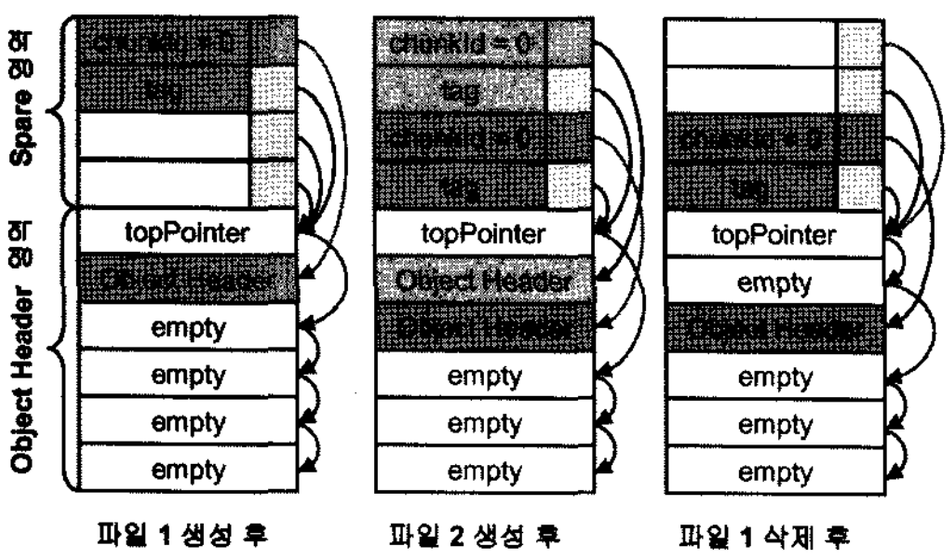


그림 6 FRASH1.5 파일 생성/삭제

하는 과정이(그림 2참조) 필요없어졌다. 또한, tag ecc를 제거함으로써 모든 tag에서 수행되는 tag ecc 확인 과정이 삭제되었다. 따라서 마운트에 필요한 최소의 동작만 수행하기 때문에 많은 시간을 감소시킬 수 있었다.

파일 생성/삭제: 본 연구에서 도입한 NVRAM에 적합한 자료 구조는 낸드 플래시 메모리 저장장치를 사용하는데 있어서 특히 파일 생성 부분의 성능 개선에 기여한다. 일련의 파일 생성과 삭제의 과정을 그림 6에 도시 하였다. 그림 4의 초기 구조로 설정된 상태에서 파일 1을 생성하면 Spare 영역에 Tag 정보가 기록되고, chunkId=0으로 설정된 Tag 영역에 있는 index는 topPointer가 가지고 있는 empty 영역을 Object Header로 지정하고, topPointer는 이 empty 영역이 가지고 있던 다음 empty 영역의 위치를 지정하도록 수정되고, Object Header 정보 및 Tag 정보를 기록한다. 하나의 파일이 생성된 후에 또 다른 새로운 파일 2의 생성이 진행될 경우, topPointer는 이미 Object Header 영역의 empty 영역을 지정하고 있기 때문에 첫 번째 파일 생성과 동일한 시간으로 진행될 수 있다. 즉, 그림 6의 두 번째 생성 후의 구조를 보면, 새로운 Spare 영역에 Tag 정보가 기록되고, chunkId=0으로 설정된 Tag 영역에 있는 index는 topPointer가 가지고 있는 empty 영역을 Object Header로 지정하고, topPointer는 이 empty 영역이 가지고 있던 다음 empty 영역의 위치를 지정하도록 수정되고. Object Header 정보 및 Tag 정보를 기록한다. 파일 삭제의 경우는 생성과 반대의 과정으로 진행 된다. 파일 1을 삭제할 경우 Spare 영역의 정보를 수정하고, topPointer는 삭제된 Object Header 영역을 지정하고, topPointer가 가지고 있던 empty 영역의 정보를 삭제된 Object Header 영역에 넘겨준다. 따라서 삭제된 Object Header 영역은 empty 상태가 되고, 그 다음 empty 영역을 지정하게 된다. 이후에 Tag 영역의 index가 topPointer를 지정하도록 수정한다.

2.4 NVRAM 자료 구조 평가

그림 7은 본 연구의 평가를 진행한 SMDK2440 Emulation Board을[7] 보여주고 있다. 128MByte의 SMC 낸드 플래시 메모리와[8] 8MB FRAM을[6] 이용하여 기존 YAFFS, FRASH1.0, 그리고 FRASH1.5의 평가를 진행하였다.

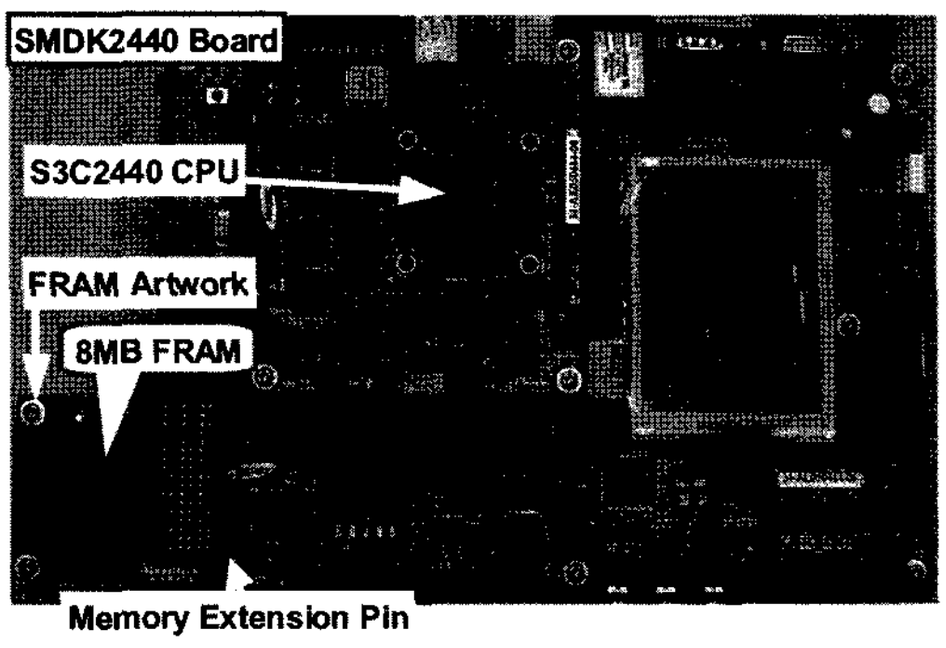
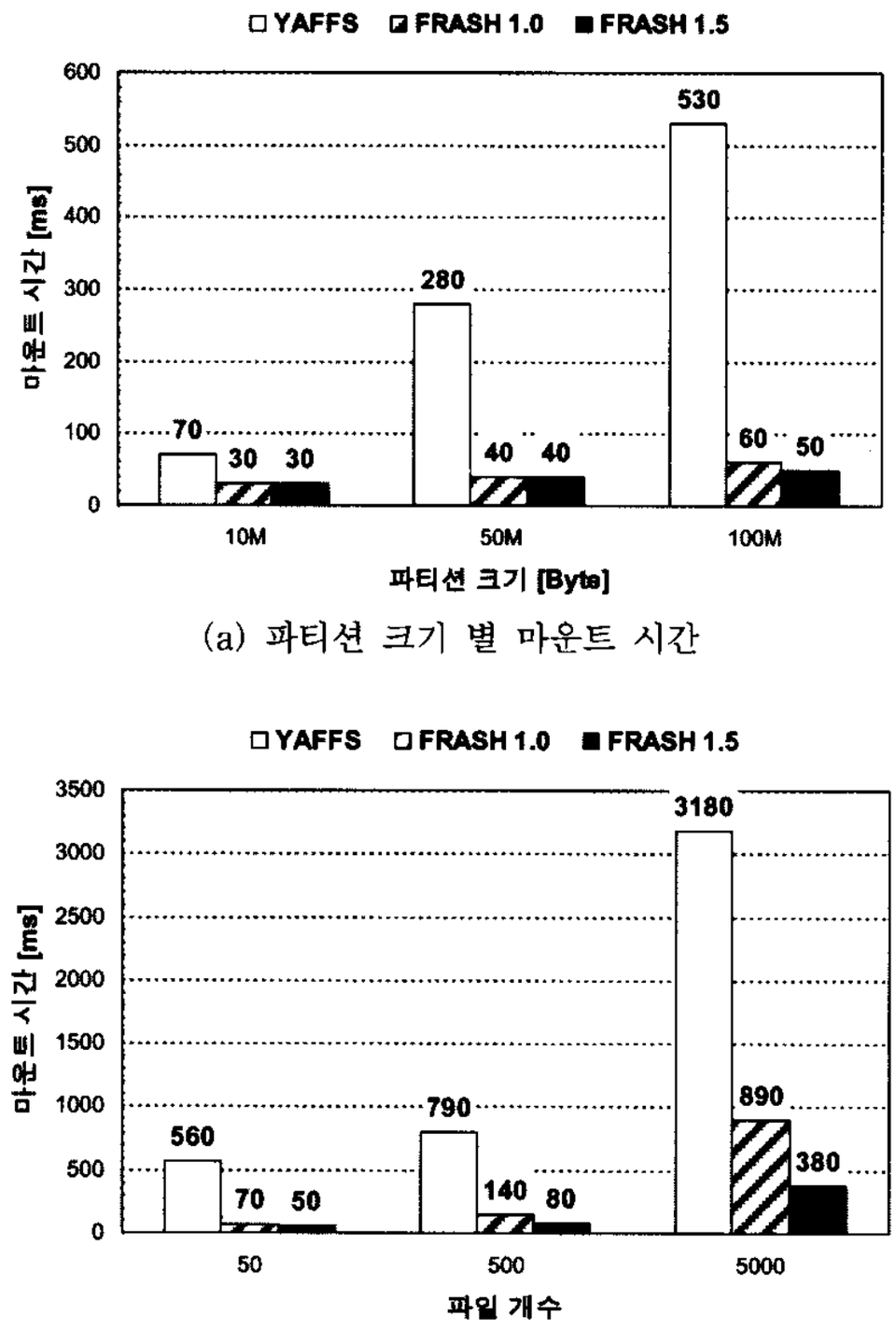


그림 7 SMDK2440 Emulation Board

그림 8의 (a)에서는 파티션 크기를 10M, 50M, 100M 등으로 구성하여 마운트 성능을 평가하였으며, 그림 8의 (b)에서는 파일 개수에 따른 평가로 100MB Partition에 1KB크기 파일을 50개, 500개, 5000개로 구성하여 평가하였다. 파티션 크기가 클수록 FRASH1.5의 성능은 더 좋아지는 것으로 나타났으며, 특히 파일 개수가 많아질수록 FRASH1.5의 마운트 성능은 이전 FRASH1.0 대비 탁월하게 향상되는 것을 알 수 있다. 즉, 파일 개수가 5000개의 경우 FRASH1.5는 FRASH1.0대비 2배 이상이 좋아졌으며, 기존 YAFFS 대비는 무려 8배 이상 향상되었다. 이것은 모든 tag영역에 있는 tag ecc의 제거와 Object Header 정보를 읽어오는 과정의 Workload의 감소로 인한 마운트 성능 개선이다. 그림 9는 새로운 NVRAM에 적합한 자료 구조를 사용하였을 경우 파일 생성/삭제의 성능 개선을 평가하기 위하여 Lmbench를[9] 수행한 결과를 나타내고 있다. 각 항목의 값은 해당파일 크기를 1000번 수행하여 초당 생성과 삭제할 수 있는 파일 개수를 평균한 수치이다. 결과에서 보는 바와 같이 FRASH1.5는 기존 FRASH1.0 대비 파일



(a) 파티션 크기 별 마운트 시간

(b) 파일 개수에 따른 마운트 시간

그림 8 마운트 성능 평가

그림 9 파일 생성/삭제 성능

생성은 3~8% 성능 향상이 되었으며, 파일 삭제의 경우는 1% 정도의 성능 향상을 가져왔다. 따라서 기존 YAFFS 대비 파일 생성 및 삭제는 1~4% 정도의 성능만 감소하였다.

3. 결 론

본 연구는 현재 많은 분야에서 응용되고 있는 낸드 플래시 메모리 저장장치의 성능을 향상시키기 위하여 고속, 저 소비 전력, 높은 개서 회수 특성을 갖는 비휘발성 랜덤 억세스 메모리인 FRAM을 낸드 플래시 저장장치에 채용한 이전 연구인 FRASH1.0의 성능 저하를 개선하고, 구조를 최적화하기 위하여 메타 데이타를 저장하는 FRAM을 효율적으로 관리하기 위한 새로운 자료 구조를 설계 하였다. FRAM에 메타 데이타를 저장함에 따라 낸드 플래시에 저장된 메타 데이타를 관리하기 위해 Spare 영역과 Object Header 영역에 구성된 항목가운데 불필요한 항목인 tag ecc를 제거 하고, 이 영역을 Object Header를 지정하는 index로 설정 하였으며, 이전 연구인 FRASH1.0에서 파일 생성시 성능 저하 요인으로 작용한 Object Header 영역의 스캔 동작을 제거하기 위하여 항상 사용하지 않는 Object Header 영역을 지정하고 있는 topPointer를 운용하는 자료구조를 설계하였다. 이것을 채용한 FRASH1.5의 성능을 평가 한 결과 마운트 동작 시간은 이전 연구인 FRASH1.0 대비 2배 이상 단축시킬 수 있었고, 파일 생성 성능은 FRASH1.0 대비 3~8% 향상되었다. 따라서 최적의 FRAM 자료 구조를 채용한 FRASH1.5는 기존 YAFFS 대비 8배 이상의 마운트 성능을 향상시켰으며, 파일 생성/삭제 성능은 기존 YAFFS 대비 평균 3% 정도만 저하되었다. 특히, 파일 크기가 크고, 개수가 많을 경우는 기존 YAFFS 대비 성능 저하 없이 마운트 시간을 8배 이상 감소시킬 수 있었다.

참 고 문 헌

[1] Intel Corporation, "Understanding the flash Translation layer(FTL) specification," http://www.intel.com, 1998.

[2] Aleph One Company, "Yet Another Flash Filing System," http://www.aleph1.co.uk/yaffs, 2002.

[3] Woodhouse, D., "JFFS: The Journaling Flash File System," Ottawa Linux Symposium, 2001.

[4] Eun-ki Kim, Hyungjong Shin, Byung-gil Jeon, Seokhee Han, Jaemin Jung, Youjip Won, 'FRASH: Hierarchical File System for FRAM and Flash," ICCSA 2007, Malaysia, pp. 238-251, 2007.

[5] RAMTRON International Corporation, FM22L16 http://www.ramtron.com.

[6] Kang, Y.M. 외 다수, "World Smallest 0.34um2 COB Cell 1T1C 64Mb FRAM with New Sensing Architecture and Highly Reliable MOCVD PZT Integration Technology," Symposium on VLSI Technology Digest of Technical Papers, pp. 124-125, 2006.

[7] Meritech Corporation: SMDK2440, http://www.meritech.co.kr/eng/

[8] 삼성 전자, "K9D1G08V0A:128MB Smart MediaTM Card," http://www.samsungsemi.com

[9] Larry McVoy, Carl Staelin, "lmbench: Portable Tools for Performance Analysis," USENIX Annual Technical Conference. 1996.