플래시 메모리 시뮬레이터의 설계 및 구현
(A Design and Implementation of Flash Memory Simulator)

정 재 용 † 노 삼 혁 †† 민 상 렬 ††† 조 유 근 †††
(JaeYong Jeong) (Sam Hyuk Noh) (Sang Lyul Min) (Yookun Cho)

요 약 본 논문에서는 실제 플래시 메모리와 동일한 특성을 갖는 플래시 메모리 시뮬레이터를 설계 및 구현한 내용을 설명한다. 본 시뮬레이터는 생산 방식, 전체 용량, 블록 크기, 페이지 크기 등 플래시 메모리의 특성을 변화시키면서 실험할 수 있을 뿐만 아니라 정확한 수행 시간과 인자 검증 기능을 제공함으로써, 운영체제 입장에서는 실제 플래시 메모리 장치를 사용하는 효과를 얻을 수 있다. 또한, 내부 루틴의 수행 시간 로깅 기능을 제공함으로써 플래시 메모리 관리 소프트웨어의 병목 지점을 판단할 수 있도록 하였다. 마지막으로, Linux 운영체제 환경에 구현된 시뮬레이터와 실제 플래시 메모리를 장착한 테스트 보드에서 응용 프로그램의 성능 측정 결과를 비교함으로써 본 시뮬레이터가 실제 플래시 메모리 장치 대용으로 사용할 수 있음을 보였다.
키워드 : 플래시 메모리, 시뮬레이터, 리눅스

Abstract This paper introduces the design and implementation of a flash memory simulator to emulate a real flash memory. Since this simulator provides exact execution time information and parameter testing functions as well as the type, total capacity, block size, and page size of flash memory, it can be used as a real flash memory as viewed by the operating system. Furthermore, the simulator provides time logging functions of the internal routines of the flash memory management software allowing the monitoring of bottlenecks within the software. Finally, we show the performance measurements of applications under the Linux operating systems on both the simulator and a test board verifying the simulator's use as a replacement for real flash memory.
Key words : Flash memory, simulator, Linux, FMSIM

1. 서 론

최근, 플래시 메모리(flash memory)는 하드디스크의 대체 저장 장치로 많은 관심을 모으고 있다. 플래시 메모리는 비휘발성 메모리로서 영구 데이타 저장소(permanent data storage)로 사용될 수 있으며, 데이타의 읽기 시간이 RAM과 비슷하고 쓰기 시간은 디스크보다 100배 이상의 좋은 성능을 보인다는 장점이 있다[1, 2]. 또한, 디스크에 비해 전력을 적게 소비하고, 충격에 강하다[3, 4]. 이러한 플래시 메모리의 특징은 특히 이동 컴퓨팅 환경에 적합하며, 현재 SmartPhone, PDA, MP3 재생기, 디지털 카메라, Pocket PC 등의 휴대정보단말기에서 데이타 저장소로 많이 사용하고 있다.

한편, 플래시 메모리는 동작 특성(operational characteristics)에서 디스크와는 다른 특징을 보이며, 이는 기존의 디스크에 최적화된 저장 장치 관리 소프트웨어를 그대로 플래시 메모리에 적용하는 것을 어렵게 하고 있다. 예를 들어, 플래시 메모리는 디스크와는 달리 데이타의 저장 위치에 상관없이 접근 시간이 동일하다. 따라서 디스크에서 사용되는 기존의 SCAN 알고리즘은 플래시 메모리에서 의미가 없다. 다른 예로, 플래시 메모리에서는 각 비트를 1에서 0으로 변환하는 쓰기 연산은 가능하지만, 0에서 1로 변환하는 쓰기 연산은 불가능하다. 이것은 플래시 메모리에서 데이타 갱신(update)을 제약하며, 메타 데이타에 대한 빈번한 갱신이 발생하는 기존의 파일 시스템을 플래시 메모리에 적용할 때 상당

† 비 회 원 : 서울대학교 전기·컴퓨터공학부
jjy@ssrnet.snu.ac.kr
†† 종신회원 : 홍익대학교 정보컴퓨터공학부 교수
samhnoh@hongik.ac.kr
††† 종신회원 : 서울대학교 컴퓨터공학부 교수
symin@dandelion.snu.ac.kr
cho@ssrnet.snu.ac.kr
논문접수 : 2001년 7월 6일
심사완료 : 2001년 11월 12일

한 성능 저하를 야기한다. 최근, 이러한 문제들을 해결하고 플래시 메모리에 최적화된 새로운 알고리즘을 개발하기 위한 연구가 많은 분야에서 진행되고 있다[3, 4, 5, 6, 7, 8, 9].

그러나, 플래시 메모리에 대한 연구들은 플래시 메모리의 다양성 때문에 그리 활발히 진행되고 있지는 못한 것이 현실이다. 거의 표준화되어 있는 디스크와는 달리 플래시 메모리는 다양한 방식과 종류가 존재한다. 플래시 메모리는 생산하는 방식에 따라 NAND 방식, NOR 방식 등의 서로 다른 유형으로 구분되며, 같은 NAND 방식이라 할지라도 삼성, Toshiba 등 제조하는 회사에 따라 종류와 특징이 조금씩 다르다. 동일한 작업 부하(workload)로 각 플래시 메모리의 유형과 종류의 차이에 대한 성능을 비교한 결과도 아직 없으며, 이러한 다양성은 제품 개발 시 '어떤 플래시 메모리를 기반으로 할 것인가' 라는 어려운 문제를 야기한다. 또한 특정 플래시 메모리를 결정하였다 하더라도, 메모리 용량 변화나 페이지 크기의 변화 같은 여러 가지 실험을 수행할 수 없다. 이러한 경우, 실제 플래시 메모리를 대신 할 수 있는 소프트웨어 시뮬레이터를 사용하는 것이 효과적이다.

소프트웨어 시뮬레이터를 통한 연구는 하드디스크부터 병렬처리 시스템에 이르기까지 다양한 분야에서 이루어지고 있다[10, 11, 12]. 소프트웨어 시뮬레이션은 크게 트레이스 기반 분석적 모델링(analytical modeling) 기법과 실제 응용 프로그램을 수행할 수 있는 직접 실행(direct execution) 또는 실행 구동(execution-driven) 기법이 있다. 플래시 메모리의 연구에는 다음과 같은 이유로 직접 실행 기법이 적합하다. 첫째, 다양한 형태의 플래시 메모리에 대한 분석적 모델링은 너무 복잡하다. 둘째, 플래시 메모리 장치 드라이버를 개발할 때 드라이버의 동작에 대한 모니터링, 디버깅을 쉽게 할 수 있다. 셋째, 여러 가지 응용 프로그램을 수행함으로써 서로 다른 환경에서의 플래시 메모리의 동작, 성능에 대한 관찰을 할 수 있다.

그러나, 프로세서, 메모리의 수행 시간에 비해서 상대적으로 느린 하드디스크와는 달리 플래시 메모리는 빠른 수행시간을 갖기 때문에 단일 처리기 시스템에서 소프트웨어 시뮬레이션을 수행할 수 없다. 처리기가 플래시 메모리를 시뮬레이션 하는 중에 커널의 인터럽트를 처리하거나 문맥교환을 하면 시간적으로 정확한 시뮬레이션 결과를 기대할 수 없기 때문이다.

본 논문에서는 이러한 문제점을 해결하기 위해 정확한 플래시 메모리의 특성을 반영하는 직접 실행 기법의 플래시 메모리 시뮬레이터(FMSIM)를 설계 및 구현하였다. FMSIM은 Linux 시스템에 디바이스 드라이버 수준에서 기존 Linux 인터페이스를 그대로 이용할 수 있도록 구현되었다. FMSIM의 장점을 정리하면 다음과 같다.

첫째, FMSIM은 플래시 메모리의 특성을 임의로 변화시키면서 실험할 수 있는 환경을 제공한다. 구체적으로 생산 방식, 전체 용량, 블록 크기, 페이지 크기, 연산의 수행 시간 등을 제어 변수로 조절할 수 있으며, 따라서 다양한 유형과 종류의 플래시 메모리를 모의실험(simulation) 할 수 있다. 또한 아직 제품으로 출시되어 있지 않은 1 GB 이상 대용량 플래시 메모리나 새로운 제어 모듈을 갖는 플래시 메모리에 대한 실험과 성능 측정을 수행할 수 있으며, 이 결과는 새로운 플래시 메모리의 설계 방향 제시 등의 기초 자료로 이용될 수 있다.

둘째, 단지 플래시 메모리에 대한 연산 기능을 제공할 뿐만 아니라 연산의 정확한 수행 시간 검증 기능을 제공한다. 이를 위해 본 연구에서는 다중 처리기를 이용해 시뮬레이터 수행 중에 인터럽트 같은 다른 소프트웨어 모듈의 간섭을 배제하였다. 따라서 플래시 메모리에 접근할 때 읽기/쓰기 시간 등을 측정할 수 있으며, 운영체제 또는 장치 드라이버 입장에서는 실제 플래시 메모리 장치를 사용하는 효과를 얻을 수 있다.

셋째, 실제 플래시 메모리 상에서는 결함을 생성하고, 결함이 있을 경우 이를 발견하고 해결하는 기능을 검증하는 것이 어렵다. 하지만 FMSIM은 인자 검증 기능과 연산 적합성 검사 기능을 제공하며, 따라서 플래시 메모리에 접근할 때 발생할 수 있는 관리 소프트웨어의 결함을 발견할 수 있다. 또한 FMSIM은 결함을 동적으로 삽입하고 이 결함을 플래시 메모리 관리 소프트웨어가 제대로 처리하는지 실험할 수 있다.

넷째, FMSIM은 Linux 시스템의 기존 디바이스 드라이버 인터페이스를 그대로 사용할 수 있도록 구현되었다. 따라서 FMSIM을 위한 장치 파일을 만들어 주어 format, mkfs 같은 기존의 Linux 응용들을 그대로 사용할 수 있으며, 플래시 메모리의 유형과 종류에 관계없이 단일한 인터페이스로 접근할 수 있다.

본 연구에서 FMSIM를 설계할 때 다음의 사항을 목표로 하였다.

o **수행시간의 정확성**: 일반 시뮬레이터는 동작 검증 기능만을 제공하지만 FMSIM은 실제 플래시 메모리 장치와 동일한 수행 시간을 실시간으로 제공한다. 따라서, 기존 시뮬레이터의 성능 평가 방법이 주로 trace 기반인 반면에, FMSIM을 사용하는 경우 실제 응용 프로그램을 수행함으로써 간편하고 실제적인 성능 평가 결과를 얻을 수 있다.

o **다양성**: 플래시 메모리는 NOR 방식, NAND 방식 등 다양한 형태로 생산되고 각각의 인터페이스가 다르다. 하지만 FMSIM에서는 여러 형태의 플래시 메모리를 일관된 인터페이스로 시뮬레이션 할 수 있도록 하였다.

o **Configurability**: 플래시 메모리는 전체 용량, 블록 크기, 페이지 크기, 스페어 영역의 유무 등의 특성이 생산 방식 및 종류에 따라 다양하다. 또한, 아직 생산되지 않는 대용량의 플래시 메모리는 이러한 값들이 어떻게 정해질 지 아직 정확한 예측이 불가능하다. FMSIM은 이러한 값들을 구성 인자(configuration parameter)로 지정하여 다양한 값에 따른 성능 측정 및 예측이 가능하도록 하였다.

본 논문의 구성은 다음과 같다. 2장에서는 본 논문에서 사용하는 용어를 정의한다. 3장에서는 FMSIM의 설계를 기술하였으며, 4장에서는 구현 방법에 대해서 설명하였다. 5장에서는 FMSIM의 타당성 검증 및 성능 분석 결과를 제시하였으며, 6장에서 결론을 맺는다.

2. 용어 정의

본 논문에서 사용하는 주요 용어들은 다음과 같다.

o **읽기(read) 연산**: 플래시 메모리로부터 데이타를 읽는 연산이다.

o **쓰기(write) 연산**: 플래시 메모리로 데이타를 쓰는 연산이다.

o **삭제(erase) 연산**: 플래시 메모리의 일정 영역을 초기값(0xFF)으로 초기화하는 연산이다.

o **블록(block)**: 삭제 연산의 단위이다.

o **페이지(page)**: 읽기 및 쓰기 연산의 단위이다.

o **데이타(data) 영역**: 일반 데이타가 플래시 메모리에 저장되는 공간이다. 페이지의 집합으로 생각할 수 있다.

o **스페어(spare) 영역**: NAND 방식의 플래시 메모리에서 제공하는 별도의 저장 공간이다. ECC(error checking code) 등을 저장하는데 사용한다.

o **Nop (Number of Operations)**: NAND 방식의 플래시 메모리에만 있는 제한으로, 한 페이지에 삭제 연산 이전까지 쓰기 연산을 할 수 있는 최대 횟수이다.

o **마모도(Wear-level)**: 한 블록에 대해 삭제 연산을 반복한 횟수이다. 한 블록의 마모도가 한계값 이상이 되면 해당 블록에 대한 연산이 올바르게 수행되지 못한다.

3. FMSIM의 설계

FMSIM은 다음과 같은 5개의 모듈로 구성된다. 첫째,

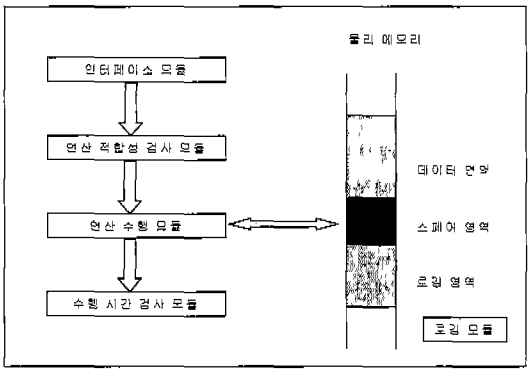

그림 1 FMSIM의 구조

인터페이스 모듈은 플래시 메모리 장치 드라이버와 같은 상위 계층과의 인터페이스를 제공한다. 둘째, 연산 적합성 검사 모듈은 Nop, 마모도 등 요청된 연산이 플래시 메모리의 제약을 위반하는지를 검사한다. 셋째, 연산 수행 모듈은 각 플래시 메모리 연산을 실제로 수행한다. 넷째, 수행 시간 검사 모듈은 정확한 연산 수행 시간을 준수하기 위해 각 모듈의 수행 시간을 모니터링한다. 다섯째, 로깅 모듈은 FMSIM 및 플래시 메모리 장치 드라이버의 성능 측정한다. 그림 1은 FMSIM의 구조를 도식화한 것이다.

3.1 인터페이스 모듈

FMSIM에서 상위 계층에 제공하는 인터페이스는 그림 2와 같다.

그림 2의 함수 중에서 read(), write(), erase() 함수는 NOR 방식과 NAND 방식에 공통적으로 제공되는 인터페이스이며, 나머지는 NAND 방식에만 제공된다. read() 함수는 주소 'from'부터 'len' byte 만큼 플래시 메모리에서 읽은 후 'buf'에 저장한다. write() 함수는 'buf'의 내용을 'len' byte 만큼 플래시 메모리 주소 'to'에 기록한다. read_oob()와 write_oob() 함수는 'ofs'가 가리키

```
read(unsigned long from, int len, unsigned char *buf)
write(unsigned long to, int len, unsigned char *buf)
read_oob(unsigned long ofs, int len, unsigned char
    *buf)
write_oob(unsigned long ofs, int len, unsigned char
    *buf)
read_ecc(unsigned long from, int len, unsigned char
    *buf, unsigned char *eccbuf)
write_ecc(unsigned long to, int len, unsigned char
    *buf, unsigned char *eccbuf)
erase(unsigned long ofs, int len)
```

그림 2 FMSIM의 인터페이스

는 스페어 영역의 데이타를 읽거나 쓰기 위해 제공하는 인터페이스이다. read_ecc()와 write_ecc() 함수는 데이타 영역과 스페어 영역을 동시에 접근하는 함수이다. 데이타 영역은 'buf'를 이용하고, 스페어 영역은 'eccbuf'를 이용해서 접근한다. erase() 함수는 주소 'ofs'부터 'len' byte 만큼 플래시 메모리를 삭제한다.

3.2 연산 적합성 검사 모듈

연산 적합성 검사 모듈은 다음과 같은 작업을 수행한다.

o **인자 적합성 검사**: 그림 2의 인터페이스에 주어지는 인자의 값이 플래시 메모리의 구성에 적합한지 검사한다. 예를 들면, read(from, len, buf) 함수가 호출되었을 때, from 값이 플래시 메모리 주소 영역 내에 포함되는지의 여부, from + len 값이 플래시 메모리 주소 영역 내에 포함되는지의 여부, buf 값이 올바른 주소값인지의 여부 등을 검사한다.

o **Nop 검사**: NAND 방식의 플래시 메모리는 각 페이지에 겹쳐 쓰기(overwrite)가 가능한 횟수가 제한되어 있다. 예를 들면, 삼성 전자의 NAND 방식 플래시 메모리 K9F5608U0M은 데이타 영역에 대해서 2회, 스페어 영역에 대해서 3회로 쓰기 연산 횟수를 제한하고 있다 [2]. 본 검사에서는 이와 같은 쓰기 연산 횟수를 위반하는 지 검사한다.

o **마모도 검사**: 각 블록에 대하여 삭제 연산이 가능한 횟수가 제한되어 있다. 플래시 메모리의 모든 블록에 대해서 마모도를 기록하고, 이 값이 제한값을 초과하는지 검사한다.

o **0->1 쓰기 검사**: 플래시 메모리는 초기 상태에 1의 값을 가지며, 0으로 쓰는 것이 가능하지만, 일단 0으로 쓴 후에 삭제 연산 없이 다시 1로 변환하는 것은 불가능하다. 따라서, 쓰기 연산을 통해 0의 값을 1로 변환하려는 시도를 검사한다.

3.3 연산 수행 모듈

연산 수행 모듈은 3.1절에서 설명한 각 인터페이스의 정의에 따라 플래시 메모리 연산을 수행한다. read(), write() 함수는 데이타 영역의 데이타를 읽거나 쓰고, read_oob(), write_oob() 함수는 스페어 영역의 데이타를 읽거나 쓴다. read_ecc(), write_ecc(), erase() 함수의 경우는 데이타 영역과 스페어 영역에 모두 데이타를 읽기, 쓰기 또는 삭제 연산을 수행한다.

3.4 수행 시간 검사 모듈

수행 시간 검사 모듈은 각 연산이 정확한 수행 시간을 준수할 수 있도록 모니터링 한다. 수행 시간 검사 모듈은 각 모듈 수행이 완료된 후에 수행되며, 현재까지 수행된 시간이 실제 플래시 메모리의 수행 시간과 일치될 때까지 기다린다.

3.5 로깅 모듈

로깅 모듈은 FMSIM의 각 연산 및 내부 루틴의 수행 시간을 측정하고, 호출 빈도에 대한 통계를 계산하는 등 FMSIM의 성능을 평가하기 위한 정보 추출 기능을 수행한다. 로깅을 위해서 그림 1과 같이 플래시 메모리의 데이타를 저장하는 데이타 영역, 스페어 영역과는 별도로 로깅 영역을 할당하여 호출된 인터페이스 함수의 id, 호출된 시간 및 완료된 시간의 time-stamp, 인자 등을 기록한다. 로깅 버퍼의 내용은 별도의 인터페이스를 통해 FMSIM 사용자가 접근할 수 있다.

4. FMSIM의 구현

4.1 플랫폼

FMSIM은 Linux 커널의 동적 적재 모듈(Dynamic Loadable Module)로 구현되었다. 개발 단계에 있는 시스템은 디버깅을 통해 잦은 소스 프로그램의 변경이 필요하다. 만일 커널 내부에 FMSIM을 구현한다면 소스 프로그램이 변경될 때마다 커널을 새로 컴파일하고 재부팅 하는 작업이 필요하다. 그러나 동적 적재 모듈로 구현하였기 때문에 커널의 컴파일이나 시스템의 재부팅이 필요 없으며, 플래시 메모리의 구성을 변경하면서 실험하기에 편리하다.

FMSIM은 다중 처리기 시스템을 기반으로 구현되었다. N 개의 처리기 중에서 1 개의 처리기는 오직 FMSIM만을 실행하고, 나머지 N-1 개의 처리기가 Linux 커널, 응용 프로그램 등 시스템의 나머지를 실행하도록 하였다. 그 이유는 단일 처리기 시스템 또는 다중 처리기 시스템에서 전용처리기를 두지 않고 FMSIM을 실행하도록 하였을 경우, 플래시 메모리 연산을 시뮬레이션 하는 중에 연산을 수행하는 중에 스케쥴링이나 인터럽트에 의해 수행이 중단되어 정확한 수행 시간을 얻을 수 없기 때문이다. 따라서, 1개의 처리기가 FMSIM 수행을 전담하며, 스케쥴링이나 인터럽트 처리에는 전혀 참여하지 않도록 구현하였다. Linux 시스템의 버전은 2.2.16이며, Intel 다중 처리기 명세 1.4[13]를 따르는 다중 처리기 시스템 상에 구현하였다. 그림 3은 다중 처리기용 Linux 시스템에서의 FMSIM의 구현을 도시한 것이다.

FMSIM이 Linux 커널의 입장에서 실제 플래시 메모리와 동일하게 사용되기 위해서는 Linux 커널의 I/O 명령을 플래시 메모리에 전달하는 부분이 필요하다. 이 부분은 Linux 시스템의 표준 인터페이스로 MTD에서 개발한 플래시 메모리 framework을 그대로 사용하였다 [14]. MTD framework에서는 플래시 메모리를 하드디

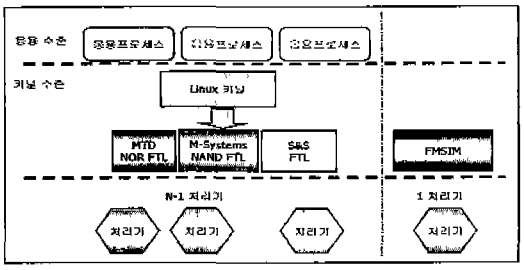

그림 3 다중 처리기용 Linux 시스템에서 FMSIM의 구현

스크처럼 관리하는 FTL (Flash Translation Layer)과 여러 가지 플래시 메모리를 동일한 장치 파일(device file) 인터페이스로 접근 가능하게 하는 mtdcore 커널 모듈을 제공한다. 즉, mtdcore 커널 모듈을 통해 FMSIM이 실제 플래시 메모리와 마찬가지로 Linux 커널에 장치로 등록이 되고, Linux 파일 시스템의 I/O 명령이 FTL을 통해 FMSIM 인터페이스 모듈을 호출한다.

FMSIM은 MTD framework에서 하나의 장치 파일로 구현되었으며, 그 결과 기존 MTD 사용자나 응용 프로그램은 기존 인터페이스의 변경 없이 FMSIM을 사용할 수 있다.

그림 3에서 "MTD NOR FTL", "M-Systems NAND FTL" 및 "S&S FTL"은 본 연구진에서 실험에 사용한 FTL의 구현의 예이며, 각각 MTD에서 개발한 NOR 방식 플래시 메모리 장치 드라이버[14], M-Systems에서 개발한 NAND 방식 플래시 메모리 장치 드라이버[14], 본 연구진이 개발한 NAND 방식 플래시 메모리 장치 드라이버[15]이다.

4.2 물리 메모리(physical memory) 관리

FMSIM은 그림 1과 같이 물리 메모리를 이용하여 플래시 메모리 연산을 시뮬레이션하기 때문에, 시뮬레이션 중에 물리 메모리에 대한 페이지 오류(page fault)가 발생하면 정확한 수행 시간을 보장하는 것이 불가능하다. 이러한 문제점을 해결하기 위해서 다음과 같은 과정을 통해 FMSIM이 사용하는 물리 메모리에 대해 페이지 오류가 발생하지 않도록 하였다.

커널 부팅 시에는 Linux 커널의 내부 함수인 setup_arch()에서 부팅 시에 필요한 메모리 관리 작업을 수행하며, 이 곳에서 FMSIM이 데이타 영역, 스페어 영역 및 로깅 영역으로 사용할 물리 메모리를 커널 주소 공간에서 제외시킨다. FMSIM 커널 모듈을 로딩할 때에는 4.4절에서 설명할 FMSIM configuration 인자 값에 따라서 플래시 메모리의 구성에 맞도록 데이타 영역, 스페어 영역 및 로깅 영역을 커널 주소 공간으로 사상(mapping) 시킨다. 이는 커널 내부 함수인 ioremap()을 사용하여 구현하였다. Linux 커널은 ioremap() 함수로 사상된 주소 공간에 대해 페이지 오류가 발생하지 않도록 스와핑(swapping)에서 제외한다.

4.3 FMSIM의 내부 모듈의 구현

4.3.1 인터페이스 모듈

인터페이스 모듈은 플래시 메모리 장치 드라이버와 같은 상위 계층에서 호출되는 부분과 FMSIM의 다른 모듈과 연동하는 부분으로 분리하여 구현하였다. 상위 계층에서 호출되는 부분은 그림 4에서 FMSIM 인터페이스 STUB로 표현하였다. FMSIM 인터페이스 STUB는 상위 계층에서 제공된 인자 및 데이타 버퍼를 inter-processor 인터럽트를 통해서 FMSIM을 수행하는 처리기로 전달한다. FMSIM 수행 처리기는 FMSIM 연동 STUB를 통해서 호출된 연산의 종류와 인자를 지정된 내부 함수로 전달한다. 연산의 수행이 종료되면 FMSIM 연동 STUB는 inter-processor 인터럽트를 이용하여 수행 결과를 FMSIM 인터페이스 STUB로 전달한다.

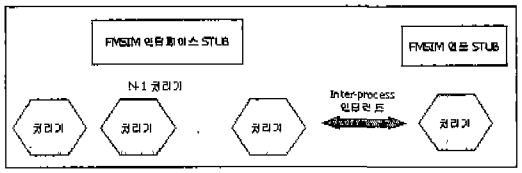

그림 4 인터페이스 모듈의 구현

인터페이스 모듈의 알고리즘의 예는 그림 5와 같다. 그림 5는 읽기 연산에 대한 예이며, FMSIM_read()는 FMSIM 인터페이스 STUB에 해당되고, __FMSIM_read()는 FMSIM 연동 STUB에 해당된다.

```
FMSIM_read()
{
  fitness_test(); /* 인자 적합성 검사 */
  /* 인자 정렬(argument marshaling) */
  xcall(__FMSIM_read(), ...); /* inter-processor 인터럽트 */
  /* 에러 검사 */
}

__FMSIM_read()
{
  /* 연산 적합성 검사 모듈 호출 */
  /* 연산 수행 */
  /* 수행 시간 검사 모듈 호출 */
  /* 로깅 모듈 호출 */
}
```

그림 5 인터페이스 모듈의 알고리즘

4.3.2 연산 적합성 검사 모듈

3.2절에서 설명하였듯이 연산 적합성 검사는 인자 적합성 검사, Nop 검사, 마모도 검사, 0->1 쓰기 검사로 나뉜다. 이를 구현하기 위해서 플래시 메모리의 구성 테이블(configuration table)과 각 블록에 대한 상태를 기록하는 상태표(statistics table)를 유지 관리한다. 플래시 메모리의 구성 테이블과 상태표는 각각 그림 6, 그림 7과 같다.

```
struct flash_config_table {
    long erase_time;
            /* usec per erase block */
    long default_read_time;    /* nsec */
    long read_time_per_byte;  /* nsec per 1 byte */
    long default_write_time;   /* nsec */
    long write_time_per_byte;
            /* nsec per 1 byte */
    int nop;
            /* maximum number of write operations */
    long wear_level;
            /* maximum wear-level */
    unsigned long totaldatasize;
            /* total size of flash memory in MB */
    unsigned long pagesize;
            /* size of page in byte */
    unsigned long blocksize;
            /* size of erase block in KB */
    unsigned long oobsize;
            /* size of spare area in byte */
};
```

그림 6 플래시 메모리 구성표

```
struct block_stat {
    unsigned int nop : 3;
       /* number of write operations */
    unsigned int wear_level : 20;
       /* wear-level of the block */
};
```

그림 7 블록의 상태표

인자 적합성 검사는 요청된 연산의 인자가 플래시 메모리의 주소 공간 내에 포함되는 지를 플래시 메모리 구성 테이블의 totaldatasize, pagesize, blocksize, oobsize 값을 이용하여 검사한다. Nop 검사와 마모도 검사는 요청된 블록에 대한 상태표의 값이 플래시 메모리 구성 테이블의 nop 또는 wear_level 값을 초과하는 지를 검사하여 수행한다. 0->1 쓰기 검사 방법은 해당 페이지의 모든 bit를 검사하는 것으로 가능하지만, 실제 플래시 메모리에서는 이미 0 값을 가진 bit에 1을 쓰면 오류가 발생하지 않고 0을 그대로 유지하기 때문에 본 구현에서도 다음과 같은 연산을 통하여 쓰기 작업을 수행한다.

olddata = olddata & newdata;

그림 8은 연산 적합성 검사의 알고리즘이다. 그림 8에서 start는 연산의 시작 주소, len은 연산의 크기, end는 플래시 메모리의 용량을 의미한다. 또한, page.nop는 해당 페이지에 대한 현재까지의 쓰기 연산 횟수이며, FM.nop는 플래시 메모리의 초기 구성으로 정해진 한 페이지에 대한 최대 쓰기 연산 횟수이다. block.wearlevel은 해당 블록에 대한 삭제 연산 횟수이며, FM.wearlevel은 플래시 메모리에서 각 블록당 허용되는 최대 삭제 연산 횟수이다.

```
fitness_test()
{
    if (start + len >= end)  /* 인자 적합성 검사 */
        return -EINVAL;
    if (page.nop >= FM.nop) /* Nop 검사 */
        return -EINVAL;
    page.nop++;
    if (block.wearlevel >= FM.wearlevel) /* 마모도 검사 */
        return -EINVAL;
    block.wearlevel++;
    for (i = 0; i < len; i++) /* 0->1 쓰기 검사 */
        olddata &= newdata;
}
```

그림 8 연산 적합성 검사의 알고리즘

4.3.3 수행 시간 검사 모듈

수행 시간 검사는 각 연산의 수행이 완료된 시점에 하고, 수행 시간이 실제 플래시 메모리의 수행 시간과 동일하게 될 때까지 대기한다. FMSIM의 각 모듈의 수행이 실제 플래시 메모리의 수행과 동일하게 완료되려면 데이타의 접근 시간이 플래시 메모리보다 빠른 기억장치를 사용하여야 한다. 현재 플래시 메모리의 읽기, 쓰기 연산의 수행 시간이 주기억장치보다 느리기 때문에 그림 1과 같이 데이타 영역과 스페어 영역을 주기억장치에 할당하였다. 따라서 각 모듈을 모두 수행한 후에도 플래시 메모리의 수행 시간을 보장할 수 있다. 플래시 메모리의 수행 시간에 대한 정보는 그림 6의 플래시 메모리 구성표에 저장되며, 각 필드(field)의 의미는 다음과 같다.

erase_time은 플래시 메모리의 한 블록을 삭제하는 데 소요되는 시간이며, 단위는 μsec이다. default_read_time은 플래시 메모리의 한 페이지에 대한 읽기 시간 중에서 데이타의 크기에 상관없이 소요되는 시간이다. 예를 들면, NAND 방식의 플래시 메모리는 데이타의 크기에 상관없이 약 10 μsec이 소요된다. 단위 시간은 nsec이다. read_time_per_byte는 default_read_time 외에 데이타의 크기에 따라 증가되는 시간을 1 byte 단위로 환산한

시간이다. 따라서, 한 페이지를 읽는데 걸리는 총 시간은 다음과 같은 수식으로 표현할 수 있다.

T(R)=default_read_time+read_time_per_byte*n

n은 읽기 연산 시 요청된 데이타의 크기이다. 단위 시간은 nsec이다. default_write_time은 플래시 메모리의 한 페이지에 대한 쓰기 시간 중에서 데이타의 크기에 상관없이 소요되는 시간이다. 단위 시간은 nsec이다. write_time_per_byte는 read_time_per_byte와 마찬가지로 쓰기 연산에서 1 byte 당 소요되는 시간이다. 따라서, 한 페이지를 쓰는데 걸리는 총 시간은 다음과 같다.

T(W)=default_write_time+write_time_per_byte*n

n은 쓰기 연산 시 요청된 데이타의 크기이다. 단위 시간은 nsec이다.

수행 시간은 펜티엄 처리기의 내부 클록인 time-stamp counter를 사용하여 측정하였다[16]. Time-stamp counter는 처리기의 성능에 따라 resolution이 달라지지만, 현재 500MHz 이상의 성능을 갖는 처리기에서는 resolution이 10ns 이하이기 때문에 정확히 플래시 메모리연산의 타이밍을 시뮬레이션 할 수 있다.

그림 9는 수행 시간 검사 모듈의 알고리즘이다. 그림 9는 읽기 연산에 대한 예이며, 연산 시작 전에 현재 시간을 기록한 후 연산이 종료된 후 정확한 수행 시간을 계산하여 현재 시간과 연산 시작 시간의 차가 수행 시간과 일치할 때까지 기다린다.

```
__FMSIM_read()
{
    read_time(start_time); /* 시작 시간 기록 */
    /* 읽기 연산 수행 */
    loop = exact_time(); /* 정확한 수행 시간 계산 */
    do {
        read_time(finish_time);
    } while (finish_time - start_time < loop);
}
```

그림 9 수행 시간 검사 모듈의 알고리즘

4.3.4 로깅 모듈

FMSIM의 내부 함수 및 플래시 메모리 장치 드라이버의 수행 시간을 기록하기 위해서 로깅 버퍼를 그림 1에서 설명한 바와 같이 물리 메모리의 일정 영역을 할당하여 사용하였다. 로깅 버퍼는 환영 버퍼(circular buffer) 구조로 구성하였으며, 기록하는 내용은 함수 id, 일련 번호(sequence number), 함수 인자, time stamp이다. 함수 id는 로깅 entry가 어떤 함수 내에서 기록되었는지를 나타내며, 일련 번호는 시간 측정 시작과 끝을 나타내기 위해 사용되었다. 즉, 일련 번호가 같은 두 로깅 entry의 time stamp의 차이를 계산하면 함수의 수행 시간을 알 수 있다. 함수 인자는 시작 주소, 길이, return 값 등을 기록한다. Time stamp는 수행 시간 검사 모듈과 마찬가지로 펜티엄 처리기의 time-stamp counter 값을 사용하였다.

로깅 버퍼는 커널 모듈에 의하여 채워지기 때문에 응용 프로그램에서 접근할 수 없다. 응용 프로그램이 로깅 버퍼에 저장된 entry들을 읽을 수 있도록 하기 위해서, Linux의 proc 파일 시스템을 사용하였다. FMSIM을 로딩하면, /proc 디렉토리에 'fmsim'이라는 이름의 파일이 생성되며, cat 등의 명령으로 로깅 버퍼의 내용을 읽을 수 있다.

4.4 Configuration

FMSIM은 커널 모듈로 구현되었기 때문에 configuration은 /etc/conf.modules 파일을 통해서 다음과 같은 항목의 값을 변경함으로써 가능하다.

- 생산 방식
- 읽기, 쓰기 및 삭제 연산의 수행 시간
- Nop
- Wear-level
- 전체 용량
- 블록 크기
- 페이지 크기
- 스페어 영역 크기
- 데이타 영역, 스페어 영역, 로깅 영역의 물리 주소

현재 생산되고 있는 몇 종류의 플래시 메모리에 대해서는 FMSIM 내부에 테이블로 위의 항목에 대한 값을 저장하고 있으며, configuration을 통해 이를 선택할 수 있다.

그림 10은 /etc/conf.modules 파일의 FMSIM과 관련된 항목의 예이다. 첫 번째 줄은 FMSIM의 내부 테이블 항목 중에서 2번째 항목을 선택하는 예이고, 두 번째 줄은 플래시 메모리의 생산 방식을 NAND 방식으로 선택한 예이다. 세 번째 줄은 기본 읽기 시간을 10 μs로 설정한 예이고, 네 번째 줄은 삭제 시간과 기본 쓰기 시간을 각각 2 ms와 100 μs로 선택한 예이다. 다섯 번째 줄은 페이지의 크기를 512 byte로 설정하는 예이다.

```
options fmsim cur_flash_config=1
options fmsim flash_type=0
options fmsim default_read_time=10
options fmsim erase_time=2000 default_write_time=100
options fmsim pagesize=512
```

그림 10 FMSIM 구성 파일의 예

5. FMSIM의 타당성 검증(validation) 및 성능 분석

FMSIM의 타당성 검증 및 성능 분석을 위하여 다음과 같은 절차에 의해 실험을 수행하였다. 첫째, 플래시 메모리의 각 연산이 정확한 시간에 수행될 수 있는지를 검증하기 위해서 각 연산의 수행 시간의 평균 및 분산을 측정하였다. 둘째, FMSIM이 실제 플래시 메모리와 동일한 수행 시간에 동작할 수 있도록 교정(calibration)하기 위해서 FMSIM의 각 연산의 수행 시간을 플래시 메모리 테스트 보드에서의 수행 시간과 측정 및 비교하였다. 셋째, 여러 가지 응용 프로그램을 FMSIM과 플래시 메모리 테스트 보드에서 실행하여 수행 시간을 측정 및 비교하였다.

표 1과 2는 첫 번째 실험의 결과이다. 표 1은 NAND 방식 플래시 메모리에 대하여 각 연산의 수행 시간 및 분산을 구한 결과이며, 표 2는 NOR 방식 플래시 메모리에 대한 결과이다. 수행 시간은 로깅 기능을 이용하여 측정하였으며, 각 연산을 10,000번 실행하여 평균 및 분산을 구하였다.

표 1 NAND 방식 FMSIM의 성능

연산	접근 크기	수행 시간	
		평균 (μs)	분산
read	512 B	36.1	0.0004
write	512 B	226.2	0.0004
erase	16 KB	2002.0	0.0255

표 2 NOR 방식 FMSIM의 성능

연산	접근 크기	수행 시간	
		평균	분산
read	512 B	29.3 μs	0.0003
write	512 B	3488.3 μs	0.0004
erase	128 KB	1.2 s	0.0246

표에서 알 수 있듯이, FMSIM의 수행 시간에 대한 분산은 대부분 0에 가까운 값이기 때문에, 연산의 수행 시간이 일정하게 유지됨을 알 수 있다. 분산 값이 0에 가까운 이유는 FMSIM을 하나의 처리기가 전담해서 수행하므로, 다른 작업에 의한 방해가 없기 때문으로 분석된다.

표 3은 두 번째 실험의 결과이다. 그림 11은 본 논문에서 사용한 ISA 버스 방식 플래시 메모리 테스트 보드의 사진이다. 테스트 보드에 사용된 플래시 메모리는 NAND 방식의 삼성 K9F6408U0A이다. 성능 측정 환경은 2개의 650 MHz Intel Pentium III CPU와 1 GB의 주기억장치를 가진 시스템이다. 운영체제는 Linux 2.2.16이다.

표 3 NAND 방식 FMSIM의 성능 교정(calibration)

연산	접근 크기	FMSIM		테스트 보드		오차 (%)
		평균(μs)	분산	평균(μs)	분산	
read	512 B	862.1	0.0014	867.8	39.6	0.7
read_oob	16 B	40.7	0.0008	41.0	2.5	0.7
write	512 B	1049.8	0.0006	1053.1	600.3	0.3
write_oob	16 B	246.2	0.0006	247.7	80.6	0.6
erase	16 KB	1917.4	0.0410	1921.1	3.0	0.2

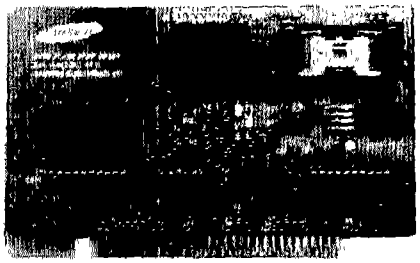

그림 11 ISA 방식 테스트 보드

FMSIM의 각 연산의 수행 시간을 교정하기 위해서 시스템 버스 지연 시간(system bus delay)을 고려하였다. 시스템 버스의 전송률은 5.33 Mbps이다[17]. 수행 시간은 각 연산을 10,000번 실행하여 평균 및 분산을 구하였다.

테스트 보드의 실행 결과는 분산 값이 매우 큼을 알 수 있다. 이는 테스트 보드의 경우에 시스템 버스를 다른 장치와 공유하기 때문에 일정한 값을 유지하기 힘들기 때문인 것으로 분석된다. FMSIM의 수행 시간은 시스템 버스 지연 시간을 조정하여 실제 플래시 메모리 테스트 보드의 수행 결과와 1 % 내의 오차를 유지하도록 하였다. 특히, 삭제 연산은 테스트 보드의 경우 플래시 메모리의 data sheet의 값 보다 적게 나왔기 때문에 FMSIM에서 삭제 연산의 수행 시간을 조정하여 오차 한계를 유지하도록 하였다.

표 4는 세 번째 실험에 사용한 응용프로그램들이다. MAB는 Modified Andrew Benchmark이며[18], 디렉토리 생성, 파일 복사, 각 디렉토리의 상태 정보 읽기, 파일 읽기, 컴파일의 다섯 단계로 구성되어 있다. MPG123는 음악 파일의 한 종류인 MP3 파일을 재생하는 프로그램이다. 본 실험에는 실제로 음악 파일을 재생하지 않고, MP3 파일을 decoding하는 작업만을 100회 반복하도록 하였다. CSCOPE는 프로그램 소스 분석기

표 4 응용 프로그램의 종류 및 설명

응용 프로그램	설명
MAB	Modified Andrew Benchmark
MPG123	MP3 재생기
CSCOPE	소스 분석기
BONNIE	파일 시스템 benchmark

이다[19]. CSCOPE는 교차 참조(cross-reference) 파일을 생성한 후 사용자가 소스를 편리하게 분석할 수 있도록 여러 가지 기능을 제공한다. 본 실험에서는 교차 참조 파일만을 생성한 후에 교차 참조 파일을 삭제하는 작업을 100회 반복하도록 하였다. BONNIE는 UNIX 파일 시스템 benchmark이다[20]. BONNIE는 크게 세 가지 작업을 수행한다. 첫 번째는 저장장치로의 출력이다. 문자 단위로 쓰기, 블록 - 여기서의 블록은 파일 시스템의 입출력 단위를 의미한다 - 단위로 쓰기, 다시 쓰기(이전에 썼던 내용을 변경하여 덮어쓰기) 작업을 수행한다. 두 번째는 저장장치로부터의 입력이다. 문자 단위로 읽기, 블록 단위로 읽기 작업을 수행한다. 세 번째는 random() 함수와 lseek() 함수를 이용하여 임의의 위치를 구한 뒤, 그 곳에 읽기와 쓰기 작업을 수행한다.

표 5는 세 번째 실험의 수행 결과이다. 표 5의 두 번째, 세 번째 열의 값은 각 응용 프로그램을 10회 반복 수행한 시간의 평균값이다. 네 번째 열의 오차는 FMSIM의 값과 테스트 보드의 값의 차이를 백분율로 표시한 값이다. FMSIM은 실제 플래시 메모리와 비교할 때 1 % 오차 범위 내에서 동작함을 알 수 있다.

표 5 FMSIM과 테스트 보드에서 응용 프로그램의 성능 비교

응용 프로그램	FMSIM (초)	테스트 보드 (초)	오차 (%)
MAB	3.403	3.425	0.6
MPG123	528.798	533.695	0.9
CSCOPE	118.546	119.218	0.6
BONNIE	6.758	6.811	0.8

이상의 성능 측정 결과로 FMSIM이 실제 플래시 메모리를 올바르게 시뮬레이션하고, 실제 플래시 메모리의 대용 장치로 사용할 수 있음을 보였다.

6. 결 론

FMSIM은 플래시 메모리의 특성(생산 방식, 전체 용량, 블록 크기, 페이지 크기, 연산의 수행 시간)을 임의로 변화시키면서 실험할 수 있는 환경을 제공한다. 또한 정확한 연산의 수행 시간과 인자 검증 기능을 제공함으로써, 실제 응용 프로그램을 수행하여 간편하고 실제적인 성능 평가 결과를 얻을 수 있도록 하였다. 로깅 기능을 제공함으로써 FMSIM 내부의 수행 시간과 호출 빈도 등의 통계 자료를 얻을 수 있도록 하였다.

FMSIM은 정확한 연산 수행 시간을 제공하기 위해서 다중 처리기 시스템용 Linux 시스템을 기반으로 구현되었으며, 성능 분석 결과 FMSIM이 실제 플래시 메모리를 올바르게 시뮬레이션 함을 보였다.

FMSIM은 운영체제 또는 응용 프로그램의 입장에서 실제 플래시 메모리와 동일한 인터페이스를 제공하기 때문에 플래시 메모리를 사용하는 응용 프로그램의 개발, 플래시 메모리의 장치 드라이버의 개발에 사용될 수 있다. 또한, 각 연산의 수행 시간에 대한 로깅 기능을 제공하기 때문에 기존 응용 프로그램 또는 장치 드라이버의 성능 개선 작업에 사용될 수 있다. 실제로 본 연구진은 FMSIM을 사용하여 Linux 시스템의 플래시 메모리 장치 드라이버에 대한 새로운 알고리즘을 개발하였다[7].

현재의 FMSIM의 제약은 다음과 같다. 첫째, FMSIM은 Linux 시스템 환경에서 플래시 메모리를 시뮬레이션 할 수 있는 환경을 제공하고 있으며, Windows와 같은 다른 운영체제 환경에서는 시뮬레이션 할 수 없다. 둘째, NOR 방식 플래시 메모리에서 제공하는 XIP(eXecution In Place) 기능을 제공하지 못한다. XIP 기능은 주로 플래시 메모리를 ROM 대용으로 사용하기 위함이며, 부팅 및 실행 기능을 시뮬레이터에 제공해야한다. 그러나, FMSIM은 소프트웨어 시뮬레이션 방식의 제약으로 인해 부팅 기능을 제공하지 못하며, Linux의 커널 모듈로 구현되었기 때문에 실행 기능을 제공하는데 많은 문제점이 있다. 마지막으로, CompactFlash, Multimedia card, Memory Stick과 같은 플래시 메모리와 제어기(controller)가 결합된 형태의 모델은 시뮬레이션 할 수 없다. 이를 정확히 시뮬레이션하기 위해서는 제어기, 제어 소프트웨어, 버스, 플래시 메모리 등을 함께 시뮬레이션 할 수 있어야 한다.

참 고 문 헌

[1] Intel Corporation, "Intel Flash Memory," http://developer.intel.com/design/flash

[2] Samsung Semiconductor, "Flash Memory Data Sheets," http://www.intl.samsungsemi.com/Memory/Flash/datasheets.html

[3] M. Wu and W. Zwaenepoel, "eNVy: A Non-Volatile, Main Memory Storage System," Proceedings of the 6th Symposium on

Architectural Support for Programming Languages and Operating Systems, pp. 86-97, October 1994.
[4] F. Douglis, F. Kaashoek, B. Marsh, R. Caceres, K. Li, and J. Tauber, "Storage Alternatives for Mobile Computers," Proceedings of the 1st Symposium on Operating Systems Design and Implementation, pp. 25-37, January 1994.
[5] M. L. Chiang and R.-C. Chang, "Cleaning policies in mobile computers using flash memory," Journal of Systems and Software, vol. 48, no. 3, pp. 213-231, 1999.
[6] A. Kawaguchi, S. Nishioka, and H. Motoda, "A flash-memory based file system," Proceedings of the USENIX 1995 Winter Conference, pp. 155-164, 1995.
[7] H.-J. Kim and S.-G. Lee, "A new flash memory management for flash storage system," Proceedings of the 23rd International Computer Software and Application Conference, pp. 294-295, 1999.
[8] A. Ban, "Flash file system," United States Patent, no. 5,404,485, April 1995.
[9] M. Assar, M. Hill, S. Nemazie, and P. Estakhri, "Flash memory mass storage architecture," United States Patent, no. 5,388,083, February 1995.
[10] S. S. Mukherjee, S. K. Reinhardt, B. Falsafi, M. Litzkow, S. Huss-Lederman, M. D. Hill, J. R. Larus, and D. A. Wood, "Wisconsin Wind Tunnel II: A Fast and Portable Parallel Architecture Simulator," Workshop on Performance Analysis and Its Impact on Design (PAID), June 1, 1997.
[11] E. A. Brewer, C. N. Dellarocas, A. Colbrook, and W. E. Weihl, "PROTEUS: A High-Performance Parallel-Architecture Simulator," Technical Report MIT/LCS/TR-516, 1991.
[12] G. R. Ganger, B. L. Worthington, and Y. N. Patt, "The DiskSim Simulation Environment Version 1.0 Reference Manual," Technical Report CSE·TR-358·98, 1998.
[13] Intel Corporation, "MultiProcessor Specification Version 1.4," http://www.intel.com
[14] MTD, "Memory Technology Device (MTD) Subsystem for Linux," http://www.linux-mtd.infradead.org
[15] J. Kim, J. M. Kim, J. Choi, J. Y. Jeong, S. H. Noh, S. L. Min, and Y. Cho, "Design and Implementation of a Flash Memory Based Storage System for Mobile Devices," http://headongdb.snu.ac.kr/technical/snu-cc-tr-2001-1.doc
[16] Intel Corporation, "Intel Architecure Software Developer's Manual Volume 3: System Programming," http://www.intel.com
[17] Intel Corporation, "L440GX+ Server Board Product Guide," http://download.intel.com/support/motherboards/server/l440gx/722077051.pdf
[18] J. Ousterhout, "Why Arent Operating Systems Getting Faster As Fast as Hardware?," Proceedings of the Summer 1990 USENIX Conference, pp. 247-256, 1990.
[19] P. Sorfa, "CSCOPE," http://cscope.sourceforge.net
[20] T. Bray, "BONNIE," http://www.textuality.com/bonnie/index.html

정 재 용

1992년 서울대학교 컴퓨터공학과, 이학사. 1994년 서울대학교 컴퓨터공학과, 공학석사. 1994년 ~ 현재 서울대학교 전기·컴퓨터공학부 박사 과정. 관심 분야는 운영체제, 입출력 시스템, 실시간 시스템

노 삼 혁

1986년 서울대학교 컴퓨터공학과, 공학사. 1993년 메릴랜드대학교 컴퓨터과학과 박사. 1994년 ~ 현재 홍익대학교 정보컴퓨터공학부 부교수. 관심 분야는 시스템 소프트웨어, 병렬처리 시스템

민 상 렬

1983년 서울대학교 전자계산기공학과, 공학사. 1985년 서울대학교 전자계산기공학과, 공학석사. 1989년 워싱턴대학 전산학과 박사. 1989년 ~ 1990년 IBM Watson 연구소 객원 연구원. 1990년 ~ 1992년 부산대학교 컴퓨터공학과 조교수. 1992년 ~ 현재 서울대학교 컴퓨터공학부 교수. 관심 분야는 컴퓨터 구조, 병렬처리 시스템, 캐쉬 메모리 시스템 등

조 유 근

1971년 서울대학교 건축공학과 학사. 1978년 미네소타대학교 전산학 박사. 1979년 ~ 현재 서울대학교 컴퓨터공학부 교수. 1984년 ~ 1985년 미네소타대학교 교환 교수. 1993년 ~ 1995년 서울대학교 중앙교육연구전산원장. 1995년 한국정보과학회 부회장. 1999년 ~ 2001년 서울대학교 공과대학 부학장. 2001년 ~ 현재 한국정보과학회 회장. 관심분야는 운영체제, 알고리즘 설계 및 분석, 암호학