논문 2008-45CI-3-7

데이터 중심 응용을 지원하기 위한 센서노드용 NAND 플래쉬 파일 시스템

(A NAND Flash File System for Sensor Nodes to support Data-centric Applications)

손 기 락*, 한 경 훈**, 최 원 철**, 한 형 진**, 한 지 연**, 이 기 혁**

(Kirack Sohn, Kyung-Hun Han, Won-Chul Choi, Hyung-Jin Han, Ji-Yean Han, and Ki-Hyeok Lee)

요 약

최근에 에너지의 효율성이 좋고 대용량화가 쉬운 NAND 플래시가 센서 노드를 위한 차세대 저장소로 각광을 받고 있다. 현재 대부분의 센서 노드용 파일 시스템은 NOR 플래시 기반으로 개발되어 있으며 NAND 플래시에 적용할 수 있는 파일 시스템은 거의 존재하지 않는다. 대용량 NAND 플래시 메모리의 특성을 고려한 새로운 파일 시스템의 구축이 요구되지만, 센서 노드는 오직 4~10 KBytes의 매우 작은 크기의 메모리를 지원하므로 효율성이 뛰어난 파일 시스템을 구축하는 것은 매우 어렵다. 본 논문은 1 KBytes의 매우 작은 크기의 EEPROM을 부착하여 이러한 메모리 한계를 극복하였으며 자원의 효율성, 대용량의 지원 및 신뢰성을 고려한 새로운 파일 시스템의 설계 및 구현에 대하여 논한다. 파일 디스크립션을 위하여 EEPROM을 사용하며 효과적으로 플래시 메모리를 쓸 수 있는 방법과 wear-leveling을 할 수 있는 방법에 대하여 제안한다. 이는 획기적으로 페이지 갱신 횟수를 줄임으로써 에너지를 절약하고 보다 긴 시간동안 데이터의 수집을 용이하게 만들며 센서 노드의 수명을 증가시킨다.

Abstract

Recently, energy-efficient NAND Flash memory of large volume is favored as next-generation storage for sensor nodes. So far, most sensor node file systems are based on NOR flash and few file systems are applicable to large NAND flash memory. Although it is required to develop new file systems taking account of the features of NAND flash memory, it is difficult to develop them mainly due to the limit of SRAM memory on sensor nodes. Sensor nodes support SRAM of 4~10 KBytes only. In this paper, we designed and implemented a novel file system to support data-centric applications. To do this, we added EEPROM of 1 KBytes to store persistent file description data efficiently and devised a simple wear-leveling method. This reduces the number of page updates, resulting in reduction in energy use and increase in lifetime of sensor nodes.

Keywords : 파일 시스템, NAND 플래시 메모리, 센서 노드, 데이터 중심적 응용

I. 서 론

최근 각광받고 있는 센서 네트워크 기술은 주변의 빛, 소리, 온도, 습도 등의 물리적 데이터의 수집을 바탕으로 환경, 의료, IT 등의 다양한 분야에 적용되고 있다. 점차 데이터의 수집 분야가 다양해지고 장기간 수집을 목적으로 사용되는 경우가 늘어나면서 대량의 데

* 정회원, ** 학생회원, 한국외국어대학교 컴퓨터및 정보통신공학부
(Hankuk University of Foreign Studies, School of Computer Science and Information Communications Engineering)

※ 이 연구는 2008학년도 한국외국어대학교 교내학술연구비의 지원에 의하여 이루어진 것임
접수일자: 2008년4월26일, 수정완료일: 2008년5월6일

이터를 저장하고 관리하기 위한 효율적인 저장장치가 요구되고 있다.

현재 센서 노드의 저장소로는 플래시 메모리가 많이 사용되고 있다. 플래시 메모리는 최근에 주목받는 반도체 기반의 데이터 저장매체로서 비휘발성(non-volatile)이고 하드디스크와 같은 보조기억 장치보다 실행속도가 훨씬 빠르다는 장점이 있다. 또한 전력소모가 매우 적으며, 물리적 충격에 강하기 때문에 센서 노드용 저장소로 가장 적합하다. 하지만 기존 데이터의 덮어쓰기(in-place update)가 불가능 하며 쓰기 횟수의 제한이 있다. 이는 파일 시스템의 성능의 한계와 알고리즘 개발에 제약을 가지게 된다[1].

플래시 메모리는 내부 소자들의 구성 형태에 따라 다양한 종류가 있는데, 그 중 가장 많이 사용되고 있는 것은 NOR형 플래시 메모리와 NAND형 플래시 메모리이다. NOR 플래시는 Read 속도가 빠르고 데이터의 안정성이 뛰어나다는 장점이 있지만 가격이 비싸고 회로가 복잡하여 대용량화하기가 힘들다. NAND 플래시는 NOR 플래시와 비교하여 Write 및 Erase속도가 월등히 빠르며 최근 단가가 낮아지고 대용량화되고 있는 추세이므로 앞으로의 센서 노드용 저장소로 적합할 것으로 예상된다. 하지만 현재 대부분의 센서 노드용 파일 시스템은 NOR 플래시 기반으로 개발되었기 때문에 NAND 플래시에 적용할 수 있는 파일 시스템은 거의 없다.

센서 네트워크를 구축할 때, 가장 많은 전력을 소모하는 부분은 통신 부분이다. RF 통신이 소모하는 에너지양은 저장에 필요한 그것의 100배이다[2]. 그래서 통신을 줄이고 데이터를 센서 노드의 저장소에 저장하는 방식이 에너지 효율적인 센서 네트워크 구축에 주효하다. 게다가 통신이 단절되었거나 쓸 수 없는 상황에서 센서 노드의 저장소는 필수적이다.

이러한 이유들로 대용량 NAND 플래시의 특성을 고려한 새로운 파일 시스템의 구축이 요구되지만 센서 노드는 오직 4~10 KByte의 매우 작은 크기의 메모리를 지원한다. 이런 이유로 기존의 Matchbox, ELF, Capsule과 같은 많은 NAND 플래시 파일시스템들은 메가바이트 단위의 많은 메모리 사용량을 요구함으로 사용이 불가능하다[2, 3, 5]. 본 논문은 1 Kbit의 매우 작은 크기의 EEPROM을 부착하여 이러한 메모리 한계를 극복하였으며 자원의 효율성, 대용량 지원 및 신뢰성을 고려한 새로운 파일 시스템의 설계에 대하여 논한다. 그리고 본 파일 시스템은 시뮬레이터를 통해 구현하였다.

Ⅱ. 관련 연구

1. 플래시 메모리의 특징

플래시 메모리용 저장 시스템의 기본이 되는 플래시 메모리는 ROM처럼 전원이 제거되어도 정보를 그대로 유지하는 비휘발성 기억 장치이다. 하드디스크와 같은 보조기억장치보다 실행속도가 훨씬 빠르고 전력소모가 매우 적고, 물리적 충격에 강해 노트북과 PC등 각종 휴대용 단말기에 적합한 기록매체로 각광 받고 있다.

하지만 RAM 이나 하드디스크와 같은 기억장치가 데이터를 읽고 쓰는 횟수가 거의 무한대인 반면에 플래시 메모리는 그 횟수에 제한이 있다는 단점이 있다. 플래시 메모리는 처리속도가 빠른 NOR 플래시와 저장용량이 큰 NAND 플래시로 구분되는데 메모리를 만드는 기본 원리는 동일하지만 셀을 어떻게 구성했느냐에 따라 구분이 되어 진다.

가. NOR 플래시의 특징

코드 저장 형이라고 불리는 NOR 플래시는 셀이 병렬구조로 어드레스라인과 데이터라인으로 연결되어 있기 때문에 셀에 바로 접근할 수 있어서 데이터의 읽기 속도가 빠르다. 그리고 데이터안정성이 뛰어나다는 장점이 있지만 가격이 비싸다는 단점이 있다. 또한 데이터 및 어드레스 라인이 있어 회로가 복잡하고 차지하는 공간이 넓기 때문에 이로 인해 반도체를 집적하기 힘들다는 단점이 있다.

나. NAND 플래시의 특징

데이터 저장 형이라 불리는 NAND 플래시는 셀이 직렬로 연결되어 있어 먼저 해당 블록으로 이동 후 직렬로 연결된 각 셀 가운데 자신이 필요로 하는 데이터에 접근하기 위해서는 다른 셀을 거쳐야만 하기 때문에 읽는 속도가 느리다는 단점이 있지만 가격이 싸고 대용량이 가능하다는 장점 때문에 많은 인기를 누리고 있다.

NAND 플래시에서 할 수 있는 4가지 주요 기능으로는 Read, Write, Erase 이 있다. Read는 플래시 메모리로부터 내용을 읽고, Write 는 플래시 메모리에 내용을 쓰는 것 이다. 이 작업들은 모두 페이지 단위로 진행된

다. Erase 기능은 Read, Write 와 다르게 블록단위로 작업되며 플래시 메모리에 새롭게 써야 할 작업이 발생하면 이를 위해 메모리 내용을 제공한다.

2. 센서노드용 파일 시스템

기존 플래시 메모리 기반의 센서 노드용 파일 시스템은 센서 노드라는 장치적 제약 때문에 그 수가 많지 않다. 대표적인 파일 시스템으로는 Matchbox, ELF, Capsule이 있으며, 효율적인 파일 시스템의 설계를 위하여 기존 파일 시스템의 특징을 비교해 보았다.

표 1. 센서노드용 플래시 메모리 파일 시스템 특징 비교

Table 1. The keynote of flash memory file system for sensor device.

구 분	Matchbox	ELF	Capsule
저장 장치	NOR	NOR	Nor, NAND
에너지 최적화	No	No	Yes
메모리 최적화	Yes	Yes	Yes
자원 평준화	No	Yes	Yes
Crash Recovery	No	Snapshot	Check pointing
추상화	File System	File System	Object
Operation	Open Create Append Read Delete	Open Create Append Read Delete Modify Seek Rename Mkdir RmDir	Open Create Append Read Delete Move
EEPROM		Metadata Index	

가. Matchbox

Matchbox[3]는 TinyOS[4]에서 기본적으로 제공하는 파일 시스템이다. NOR 플래시 기반에서 설계되었으며 오직 메모리 최적화에 대한 측면만 고려되었다. 에너지 효율성과 자원 평준화는 고려되지 않았고 Crash Recovery에 대한 정책은 없다. NOR 플래시 기반이지만 Write-append, Seek명령을 지원하지 않으며 Free 페이지 리스트를 메인 메모리에서 관리하므로 전원이 소실되면 부팅 시간에 재구성한다.

나. ELF

ELF[5]는 Matchbox와 같이 NOR 플래시 기반에서 설계되었으며 NOR 플래시의 특징을 살려 Write-modify 연산과 Seek 기능을 지원한다. 메모리 최적화와 자원 평준화를 고려했으며 Crash Recovery에 대한 정책으로 Snapshot을 사용한다. 유일하게 EEPROM을 사용하며 디렉토리와 파일의 메타데이터와 인덱스를 저장한다. 하지만 Atmel 데이터 플래시 장치에 특성화하여 구현되었기 때문에 다른 장치를 대상으로 사용할 수 없으며 버퍼를 순환 형태로 사용하여 센서 네트워크에서 사용하는 다양한 응용 프로그램들을 만족시키지 못한다.

다. Capsule

Capsule[2]은 NOR 플래시에서 NAND 플래시의 특징을 살려 구현되었다. 에너지 최적화, 메모리 최적화, 자원 평준화를 모두 고려하였으며 Crash Recovery 정책으로 Checkpointing을 사용한다. 파일 기반이 아닌 객체 기반으로 스택, 큐, 인덱스 등의 다양한 자료 구조를 저장 시스템 단위에서 지원한다. 하지만 인덱스는 컴파일시간에 크기가 고정되므로 파일의 크기나 Fragment 수가 제한된다. 또한 Backward Chaining을 사용하므로 Write-modify와 Read연산이 매우 비효율 적이다. 이러한 이유 때문에 유일하게 NAND 플래시에서 동작되도록 설계되었지만 대용량 NAND 플래시 파일 시스템으로는 적합하지 않다.

라. 각 파일 시스템의 취약점 분석

위의 언급한 각 파일 시스템에 대한 분석을 기반으로 보완할 수 있는 사항을 고려해 본다.

(1) Matchbox

Matchbox는 NOR 플래시 기반이지만, write-append 나 seek 명령을 지원하지 않는다. Free-list를 사용하여 공간 관리를 하지만 웨어 레벨링을 고려하지 않았으므로 앞부분의 삭제가 자주 발생할 경우 앞부분만 과도하게 사용하게 된다.

Free-list가 메인 메모리에서 관리되므로 전원을 소실하였을 때 free-list 정보도 소실하게 되어 재구성을 해야 하는 문제가 있다.

(2) ELF

ELF는 Atmel 데이터 플래시 장치에 특성화하여 구현 되어 다른 장치를 대상으로 코드를 재사용할 수 없다. ELF에서 사용하는 circular 버퍼 형태의 기술은 센서 네트워크에서 사용하는 다양한 응용 프로그램을 만족시키지 못한다.

센서 네트워크의 특징상 자주 발생하는 연산(reboot, file open, file create)을 잘 처리하도록 디자인되지 않는다. 버퍼링은 ELF의 성능과 효율성에 중요한 역할을 하지만 신뢰성 측면에서 비용을 부담해야 한다. 센서 노드에서 파일의 신뢰성을 보장하기 위한 latency 비용 혹은 에너지 비용을 측정하지 않는다.

(3) Capsule

Capsule에서 제공하는 인덱스 기능은 compile시에 크기가 고정된다. Forward chaining이 되지 않아 앞부분을 읽으려면 가장 뒤부터 전체를 읽어야 한다.

수정 자체는 가능하지만 인덱스가 고정되어 있어 중간 부분이 페이지크기를 초과할 경우 처리를 할 수 없다. 인덱스를 메모리에도 유지시키기 때문에 고정된 크기를 벗어날 수 없으며, 인덱스의 사용이 파일시스템 용도로 사용할 경우에도 사용되면서 파일 크기나 fragment 수의 제한이라는 제약 조건을 만든다.

Compaction에 많은 시간이 요구되어 사실상 이용이 쉽지 않다. 그리고 Capsule에서 제안하는 객체 이외의 자료구조에는 적용이 되지 않아 일반적으로 사용하기 어렵다는 단점이 있다.

III. 설계 및 구현

1. 하드웨어 환경

고성능의 파일 시스템을 설계하기 위해서는 효율적인 알고리즘과 적절한 크기의 저장매체의 조합이 매우 중요하다. 파일 시스템의 요구사항에 대해 하드웨어를 적절히 배분하여 전체 시스템을 설계함으로써 보다 우수한 성능의 파일 시스템을 설계할 수 있다. 본 파일 시스템은 저 전력을 목적으로 제작된 TIP시리즈의 TIP810CM Mote[6]를 기반으로 설계하였다. 저장 장치의 구체적인 사양은 다음과 같다.

■ Configuration
- NAND 플래시 : 1Gbit = 128 MBytes
- EEPROM : 1 KBytes
- SRAM : 10 KBytes

■ NAND 플래시 Page Count
- Total Page count : 262144
- Block count : 8192
- Page count per Block : 32

■ Page Size (small page 사용)
- Data size : 512 Bytes
- Spare size : 16 Bytes

■ Addressing
- Offset uses 1 Byte (A0 ~ A7)
- Address uses 3 Bytes (A9 ~ A25)
- A8 is Low(00H Command) or High(01H Command)

가. NAND 플래시 1 Gbits

Matchbox[3]는 TinyOS[4]에서 기본적으로 제공하는 파일 시스템이다. NOR 플래시 기반에서 설계되었으며 오직 메모리 최적화에 대한 측면만 고려되었다. 에너지 효율성과 자원 평준화는 고려되지 않았고 Crash Recovery에 대한 정책은 없다. NOR 플래시 기반이지만 Write-append, Seek명령을 지원하지 않으며 Free 페이지 리스트를 메인 메모리에서 관리하므로 전원이 소실되면 부팅 시간에 재구성한다.

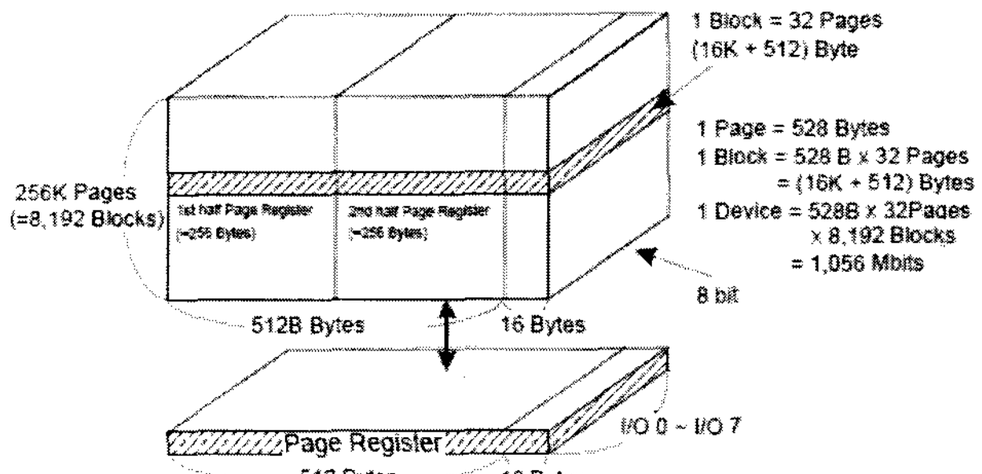


그림 1. SAMSUNG NAND 1Gbit 메모리 배열 구성
Fig. 1. Array configuration of SAMSUNG NAND 1Gbit Memory.

나. EEPROM 1 KBytes

NAND 플래시는 동일한 페이지에 덮어쓰기가 불가능 하므로 특정 위치에 데이터를 보관할 수 없다. 이로

인해 데이터가 갱신될 때 마다 매번 그 위치가 변한다. 그래서 기존에 존재하는 대부분의 파일 시스템에서는 부팅타임에 전체 플래시 메모리를 스캔해서 필요한 정보들을 SRAM에 구성하여 사용한다. 하지만 플래시 메모리는 점점 대용량화되고 있기 때문에 매번 전체 메모리를 모두 스캔하기에는 시간과 전력의 한계가 있다.

본 파일 시스템의 설계에서는 EEPROM을 사용하고, 1 KBytes 를 외부 메모리로 부착했다. EEPROM은 동일한 위치에 덮어쓰기가 가능하고 비휘발성이므로 필요한 데이터의 위치를 기록하는데 적합하다. 즉, 부팅 시간에 플래시 메모리 전체를 스캔하지 않고 EEPROM의 특정 위치만 확인하면 필요한 정보를 얻을 수 있다.

표 2는 EEPROM에 기록하는 세부 목록이다. 총 528 Byte를 사용하는데 그중 520 Byte는 MAX_FILES로 정의된 20개 파일에 대한 메타 정보를 저장하고, 1 Byte는 현재 파일 시스템에 존재하는 파일의 수를 기록한다. 2 Byte는 현재 OPEN 상태인 파일의 ID를 기록하는데 동시 OPEN 가능한 파일 수를 OPEN_FILE_ NUM으로 정의한 2개로 제한하고 있다. 1 Byte는 파일 생성시 마지막으로 할당한 ID를 저장하여 새로운 파일이 생성될 때 이 ID의 다음 값을 할당한다. 마지막 4 Byte는 전체 블록의 wear leveling을 위한 한계 값으로 블록을 할당할 때 할당을 허용하는 제한 값을 의미한다.

표 2. EEPROM에 기록할 데이터 구조
Table 2. Data Structure of EEPROM.

목 적	할당 Byte
File Description[MAX_FILES]	520 Byte
File Count	1 Byte
Open File Info [OPEN_FILE_NUM]	2 Byte
Last Allocated File ID	1 Byte
Wear leveling을 위한 한계 값	4 Byte
총 계	528 Byte

다. SRAM 10KBytes

TIP810CM은 10 KBytes의 SRAM을 제공한다. SRAM은 파일 시스템 뿐만 아니라 응용프로그램에서도 사용하므로 매우 작은 크기이다. SRAM의 크기가 매우 작기 때문에 동시에 Open가능한 파일의 수를 2개로 제한시켰다. 파일이 Open될 경우 빈 버퍼를 점유하여 Close될 때 까지 사용한다.

본 파일 시스템에서는 10 KByte중에 1088 Byte를 4

표 3. SRAM의 데이터 구조
Table 3. Data Structure of SRAM.

목적	크기
File Description Map	520 Byte
File count	1 Byte
Block info bitmap	2048 Byte
Priority queue	30 Byte
Priority queue length	1 Byte
Buffer1 - for open file 1	528 Byte
Buffer2 - for open file 2	528 Byte
S_Buffer1	16 Byte
S_Buffer2	16 Byte
총계	3688 Byte (3.6K)

개의 버퍼로 사용한다. 두 개의 버퍼는 528 Byte이며, 다른 두 개의 버퍼는 16 Byte이다. 528 Byte의 버퍼는 READ나 WRITE와 같이 데이터 영역과 스페어 영역 모두를 사용하는 경우에 사용되며, 16 Byte의 버퍼는 스페어 영역만을 접근할 때 사용된다. 520 Byte는 EEPROM에 저장된 File Desctiption을 읽어 저장하는 용도로 사용되는데, 이는 파일의 생성할 때 중복된 파일의 여부를 판단하는 경우나 DIR을 수행하는 경우 파일 디스크립션 모두에 접근하게 되는 과정에서 EEPROM에 접근하는 횟수와 시간을 줄이기 위함이다.

Block info bitmap은 NAND 플래시 메모리의 모든 블록에 대한 상태 정보를 저장하고 있다. 하나의 블록의 상태는 2 bit를 사용하여 FREE_HIGH, FREE_LOW, IN_USE, GARBAGE의 네 가지의 상태로 표현한다.

두 개의 작은 버퍼 (S_Buffer1과 S_Buffer2)는 16 Byte로 이루어진 NAND 플래시의 스페어 영역의 정보를 관리하는 용도로 사용된다. 스페어 영역의 정보는 free block map을 생성할 때, free page를 할당할 때, garbage collection을 수행할 때, 파일을 생성할 때, 파일을 삭제할 때 사용된다.

2. NAND 플래시 파일 시스템의 구조

파일 시스템은 페이지, 블록, 파일, 파일 시스템의 순으로 아래와 같이 구성된다.

가. 페이지의 구성

각각의 페이지는 데이터 영역 512 Byte와 스페어 영역 16 Byte로 이루어져 있다. 512 Byte의 데이터 영역 중 0 ~ 1번에 해당하는 2 Byte는 해당 페이지에 기록된 데이터의 크기를 나타내며, 2 ~ 511번 Byte에 해당

하는 510 Byte는 실제 데이터를 기록하는 데 사용한다. 16 Byte의 스페어 영역은 해당 페이지의 정보 및 자신이 속하는 블록에 대한 메타 정보를 기록한다.

나. 블록의 상태

NAND 플래시의 특성인 블록 단위로 삭제가 이루어진다는 특징에 맞추어 WRITE를 위한 공간의 할당은 블록 단위로 이루어진다. 블록은 크게 본다면 그림 2와 같이 FREE, IN_USE, GARBAGE의 세 가지 상태로 구분이 되며, 파일 시스템 내에서는 FREE 상태를 FREE_HIGH와 FREE_LOW로 구분한 네 가지 상태가 존재한다. 구분된 상태는 SRAM의 free Block info bitmap에 기록된다. 각 상태가 갖는 의미는 다음과 같다.

- FREE_HIGH - 블록이 할당 가능한 free 상태이지만, 사용 횟수가 wear level 한계 값과 일치 또는 초과함을 나타낸다. 이 블록의 할당은 이루어 지지 않으며 FREE_LOW인 다른 블록을 할당하게 된다.
- FREE_LOW - 블록이 할당 가능한 free 상태이며, 사용 횟수 또한 wear level 한계 값보다 적어 할당이 가능함을 나타낸다.
- IN_USE - 블록의 데이터가 유효하다.
- GARBAGE - 블록의 데이터가 더 이상 유효하지 않으며 garbage collection의 수행 대상임을 나타낸다.

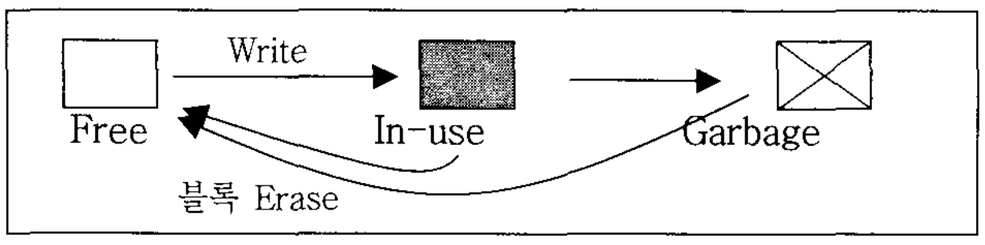


그림 2. 블록의 상태 변화
Fig. 2. Status changes of block.

다. 파일의 구조

파일은 WRITE_APPEND 타입과 WRITE_FIXED 타입의 두 가지 타입을 지원한다. WRITE_APPEND 타입은 용량에 제한을 두지 않고 계속해서 기록해 나아가는 타입이고, WRITE_FIXED 타입은 파일을 생성할 때 파일의 크기를 고정하는 타입이다. 일반적인 방식인 WRITE_APPEND 외에 WRITE_FIXED를 제공하여 얻을 수 있는 이점은, WRITE_FIXED 타입의 경우 파일을 생성할 때 그 크기에 필요한 만큼의 블록을 미리 할당하여 두기 때문에 WRITE를 수행하는 과정에서 저장장치의 용량이 부족하여 WRITE를 할 수 없는 상황을 미연에 방지할 수 있다는 것이다.

표 4. 파일 디스크립션의 구조
Table 4. Configuration of file description.

목적	할당 Byte
File Name	14 Byte
File Mode	1 Byte
File ID	1 Byte
File Size	4 Byte
초기 블록 번호	2 Byte
마지막에 기록한 페이지 번호	3 Byte
현재 기록 중인 블록에서 유효하지 않은 페이지 수	1 Byte
총 계	26 Byte

본 파일 시스템에서는 파일의 관리를 EEPROM에 기록한 메타데이터인 파일 디스크립션을 통하여 수행하며 그 구조는 표 4와 같다.

File ID는 파일이 생성될 때 파일시스템에서 부여하는 것으로 여러 작업을 수행할 때 해당하는 파일을 대표하는 적은 용량의 키 값이라고 볼 수 있다.

초기 블록 번호는 해당 파일의 데이터가 기록된 첫 번째 블록의 번호를 나타내며, 이후의 블록은 해당 블록에 할당 된 next block 번호를 기록하는 페이지의 스페어 영역에 기록된다.

마지막에 기록한 페이지 번호는 파일의 데이터가 마지막으로 기록된 페이지의 번호로 블록 번호는 별도로 기록되지 않으나 초기 블록 번호로부터 차례로 이동하여 찾아갈 수 있다.

현재 기록 중인 블록에서 유효하지 않은 페이지 수는 현재 기록 중인 블록에서 sync와 같은 full page flush (페이지가 가득 차도록 기록하는 것)이 이루어지지 않은 페이지의 수를 나타낸다. 만약 가득차지 않은 페이

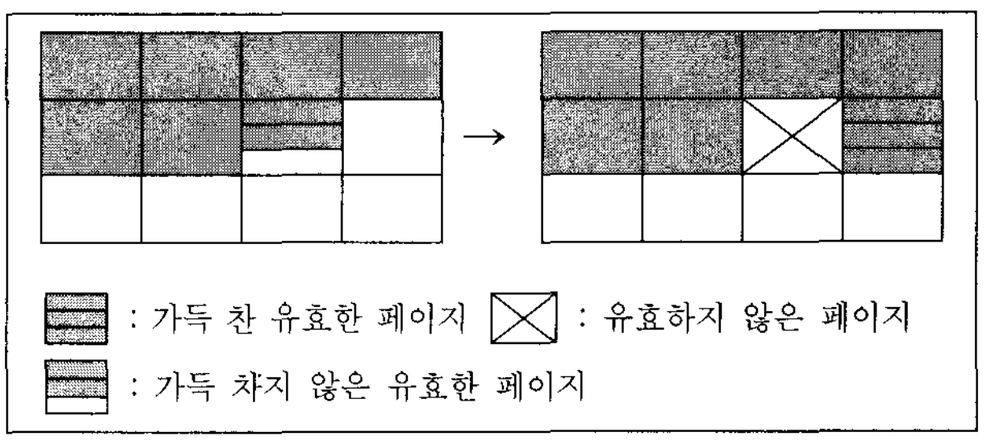


그림 3. Underfull page flush 와 Full page flush
Fig. 3. Underfull page flush and Full page flush.

지를 기록한 뒤에 WRITE 명령이 다시 수행될 경우, 그림 3과 같이 이전에 기록한 페이지의 내용을 그대로 복제하여 추가되는 내용을 연결한 뒤 full page로 기록을 수행하고 이전에 기록된 페이지는 유효하지 않은 page로 추가한다. 유효하지 않은 페이지 수는 기록하는 블록이 변경될 때 기록 중이던 블록의 해당하는 스페어 영역에 기록하고 새로운 블록을 위해 초기화 한다.

라. 파일 시스템의 구조

파일 시스템은 동시에 OPEN 할 수 있는 파일의 수를 두 개로 제한하였다. READ나 WRITE의 경우 명령이 수행 중인 경우 BUSY 상태로 설정하여 다른 작업이 중복 접근되는 것을 미연에 방지한다.

3. NAND 플래시 파일 시스템에 적용한 알고리즘

NAND 플래시와 센서라는 제한된 환경에서 파일 시스템을 구성하기 위하여 저장 공간을 효율적으로 사용하는 방법과 장치 수명을 고려한 wear leveling을 위한 몇 가지 알고리즘을 제안한다.

가. 저장 공간과 에너지 효율성을 고려한 WRITE

NAND 플래시의 특성인 한 번 쓰면 삭제하기 전까지 쓰기를 할 수 없다는 것과 한 페이지에 쓰기를 할 수 있는 횟수가 제한되어 있다는 점을 고려하여 본 파일 시스템은 한 페이지에는 데이터 영역 한 번, 스페어 영역은 두 번의 WRITE 만을 하도록 설계하였다.

일반적인 파일 시스템과 같이 상위 응용프로그램에서 write 명령을 내리면 바로 저장장소에 기록하는 방식을 사용할 경우, 적은 용량 단위로 다수의 write 명령을 받게 된다면 접근 횟수 제한에 의하여 한 페이지의 데이터 영역인 512 Byte를 모두 사용하지 못하는 상황이 발생하게 된다. 본 파일 시스템은 상위 응용프로그램에서 write 명령을 내리면 내부에 가지고 있는 510 Byte의 write 버퍼를 가득 채우기 전까진 플래시 메모리에 실제 write를 수행하지 않는다. 하지만 파일 시스템을 이용하는 과정에서 버퍼가 가득 차지 못했지만 write를 해야만 하는 경우(파일의 CLOSE 또는 지금까지 write 버퍼의 내용을 강제로 기록하는 명령인 SYNC)가 발생할 수 있는데, 이러한 경우엔 그림 3과 같은 방식으로 유효하지 않은 페이지를 발생 및 그 수를 기록하는 방법으로 수행한다.

나. wear leveling

NAND 플래시가 가진 수명을 고려할 때 한 블록만을 집중적으로 사용할 경우 사용 한계에 도달하여 전체 저장소 중 일부의 영역만을 사용하는 문제가 발생한다. 이러한 문제점을 해결하기 위하여 본 파일 시스템에선 장치 전체의 블록을 균등한 사용량을 유지하도록 할당하는 방법을 제안한다.

SRAM에 각 블록의 상태에 대한 bitmap을 구성하고, 사용횟수를 블록의 특정 페이지에 기록하여 EEPROM에 기록한 한계 값과 비교하여 할당 여부를 결정한다. IN_USE 상태인 블록을 FREE 상태로 변경하는 것은, 파일의 DELETE 명령이 수행될 때가 아닌 garbage_collection 때이다. DELETE 명령은 삭제 될 파일이 사용하고 있는 블록을 GARBAGE 상태로 변경하고 파일 description을 삭제하는 과정만을 수행한다. 블록의 상태가 GARBAGE인 블록은 파일 시스템이 할당받을 블록 여유가 충분하지 않거나, 작업의 여유가 있다고 판단될 때 수행하게 되는 garbage_collection에 의하여 실제 erase 명령이 수행하게 되며, 이미 기록된 사용 횟수를 1회 증가시키고 EEPROM의 사용 한계 값과 비교하여 FREE_HIGH, FREE_LOW로 설정하게 된다. EEPROM에 기록된 사용 한계 값은 free block bitmap에 FREE_LOW가 존재하지 않아서 더 이상 할당할 수 있는 블록이 없을 경우에 EEPROM의 한계 값을 1 증가시키고 free block bitmap의 FREE_HIGH 값을 FREE_LOW로 변경한다.

파일 시스템에 어떠한 파일이 오랫동안 존재하여 일부 블록의 사용 횟수가 다른 블록에 비하여 크게 적을 경우가 발생할 수 있는데, 그러한 블록은 언젠가 GRABAGE 상태가 되었을 때 사용 횟수가 적은 순으로 priority Queue(우선순위 큐)에 저장하여 우선적으로 할당한다.

(1) booting time

파일 시스템의 부팅이 이루어질 때 파일 시스템은 NAND 플래시 전체 블록의 스페어 영역을 확인하여 SRAM에 free block bitmap을 생성한다. 이렇게 생성된 free block bitmap은 새로운 블록이 필요할 때 free block allocate 모듈에 의하여 할당하게 된다. 그림 4는 부팅이 이루어 질 때 수행하게 되는 작업의 흐름을 보여주고 있다.

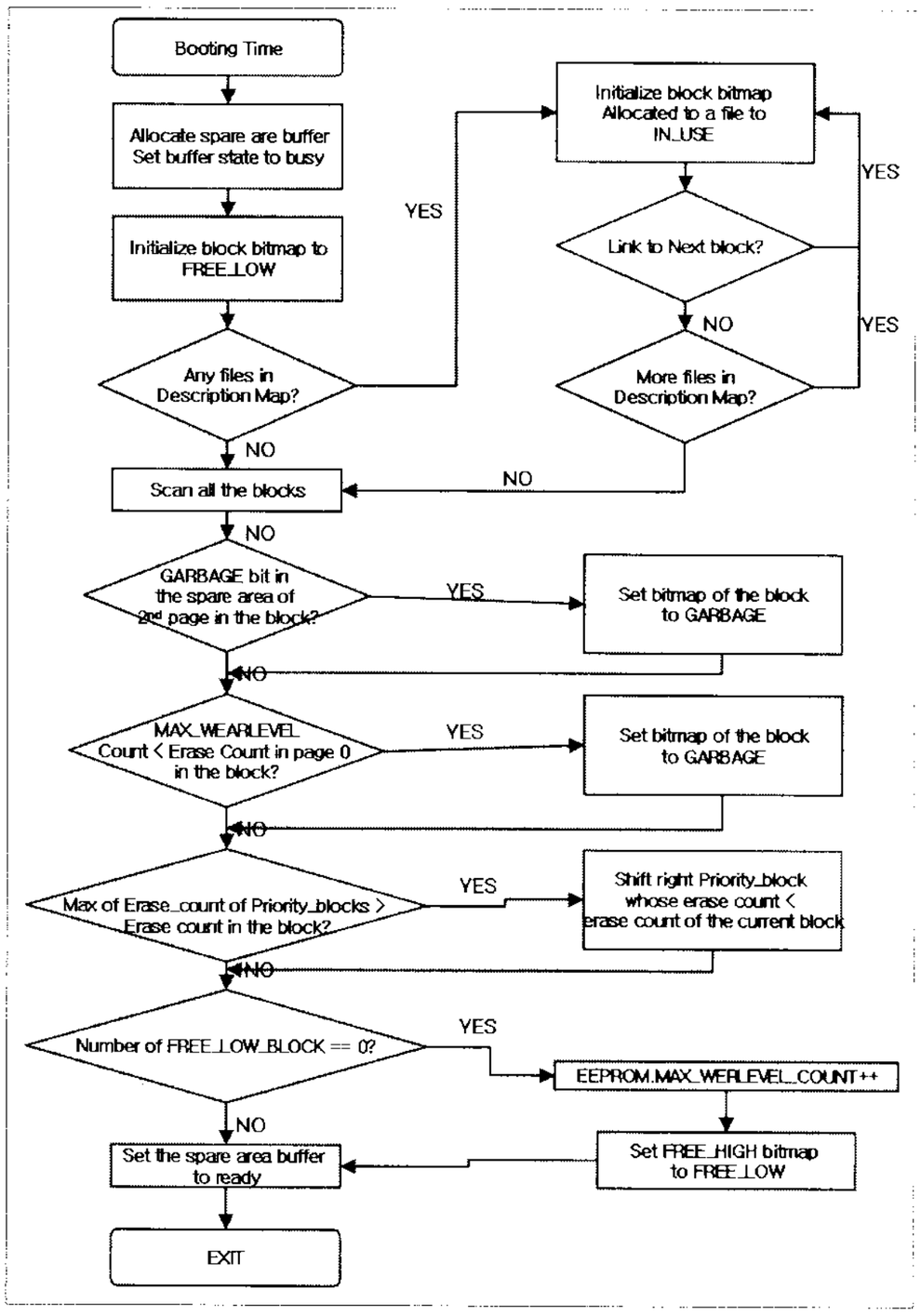


그림 4. Booting time flow chart
Fig. 4. Booting time flow chart.

(2) free block allocation

WRITE_FIXED 형식의 파일 CREATE나 WRITE에서 새로운 블록이 필요할 경우 free_block_allocate 모듈에 의하여 블록 할당이 이루어진다. free block allocate 모듈은 priority Queue에 저장된 블록 번호를 우선적으로 할당하고, priority Queue에 저장된 블록 번호가 없을 경우엔 free block bitmap을 탐색하여 가장 먼저 발견되는 FREE_LOW 상태인 블록을 할당한다. 할당을 하면 마지막으로 할당한 블록을 기억하며, 다음 할당이 요청될 때 그 블록의 뒤부터 검색하여 할당한다.

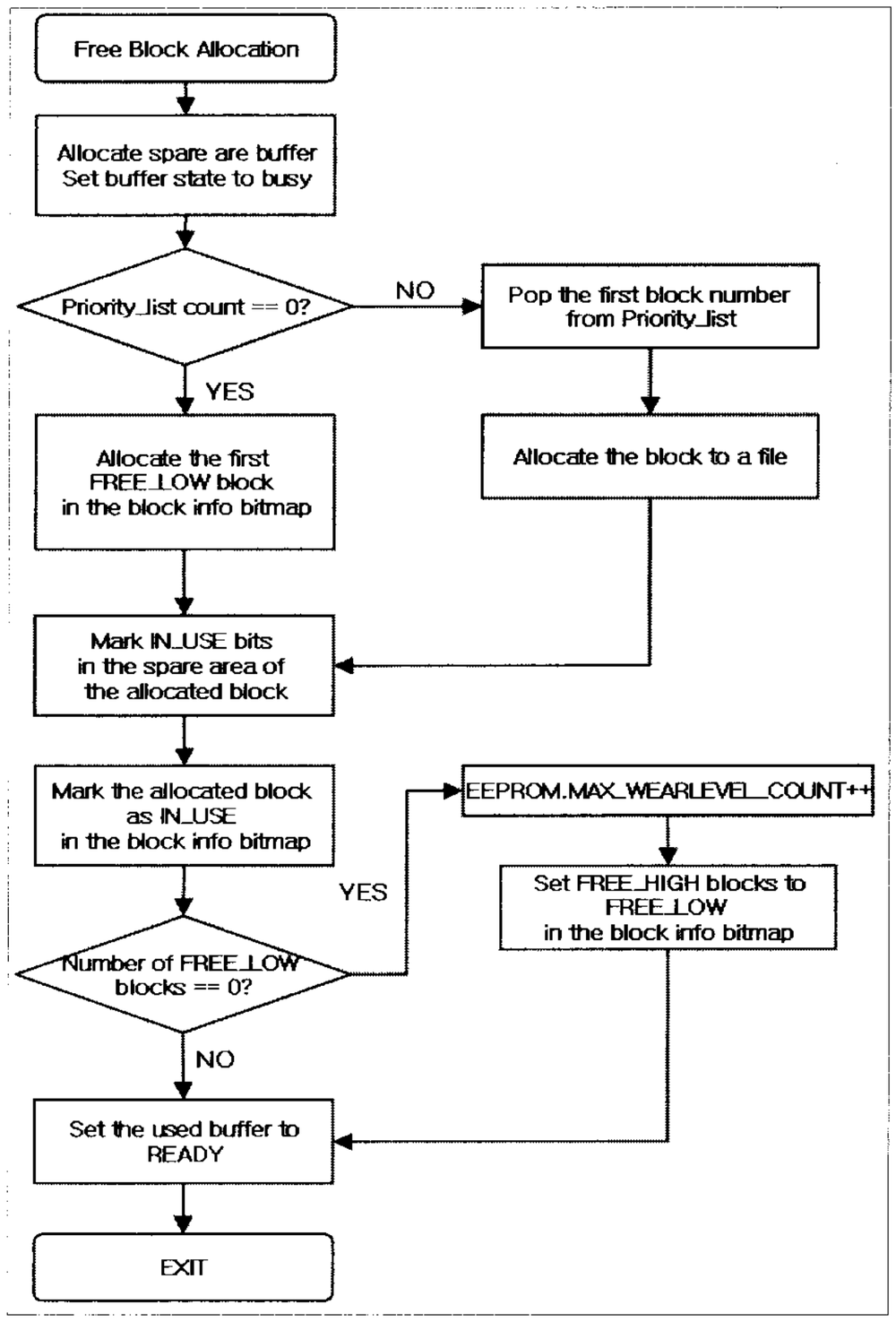


그림 5. Free block allocate flow chart
Fig. 5. Free block allocate flow chart.

4. 구현

본 파일 시스템은 TinyOS 1.x 상에서 NesC를 이용하여 Eclipse 3.1 환경에서 구현하였다.

가. 지원하는 명령어

(1) Create

Create는 새로운 파일을 생성하는 파일 시스템 명령어이다. WRITE_FIXED, WRITE_APPEND 모드 중 하나를 선택하여 파일을 생성시킨다.

WRITE_FIXED는 사용하고자 하는 데이터 크기를 고정하여 사용하는 옵션이고 WRITE_APPEND는 데이터가 입력될 때마다 데이터 기록 영역을 자동으로 넓히는 옵션이다.

두 가지 모드는 파일 생성 시 EEPROM에 있는 파일 디스크립션 맵을 통해 파일의 생성이 가능한지를 확인하고 파일을 생성한다. WRITE_FIXED는 파일 생성 시 원하는 사이즈를 블록 단위로 환산하여 할당하며 할당된 첫 번째 블록을 initial블록으로 설정하고 할당된 블록들의 3번째 페이지에 각각 다음 할당받은 블록의 번호를 기록한다. WRITE_APPEND는 첫 번째 블록만을 할당하고 블록을 다 쓰면 그 순간에 다음 블록을 할당하는 방법을 사용한다.

(2) Delete

Delete는 파일을 삭제하는 파일 시스템 명령어이다.

파일명을 입력받아 현재 OPEN되어 있지 않으면 해당 파일을 버퍼에 있는 디스크립션 맵과 EEPROM에서 삭제 후 해당 파일이 가진 블록들의 스페어 영역에 GARBAGE 마킹한다. 실제 블록의 삭제는 추후 garbage collection에 의해서 이루어진다.

(3) Open

Open은 Open되어 있는 동안 해당 파일에 대한 파일 연산을 허용하겠다는 명령어로 동시에 2개의 파일 Open이 가능하다. READ나 WRITE용으로 OPEN하면 파일의 이름을 받아 파일이 존재하는지를 파일 디스크립션 맵에서 검색 후 파일이 존재한다면 같은 파일이 Open되어 있는지를 확인한다. Open이 되어있지 않다면 버퍼와 EEPROM에 Open될 파일의 정보를 저장하고 파일의 기본정보를 버퍼에 상주시킨다. 만약 파일이 CRASH 모드로 되어있다면 기존에 파일이 WRITE하는 도중에 모트의 전원이 꺼진 상황이므로 WRITE용으로는 Open되지 못하게 한다.

(4) Close

Close는 Open되어 있는 파일을 닫음으로써 해당 파일에 대한 파일연산을 더 이상 허용하지 않는다. 파일을 Close하기 전에 WRITE연산을 하던 중이라면 버퍼 안의 데이터의 사이즈가 0이 아니거나 버퍼의 데이터가 EEPROM에 적혀있는 데이터로 나눈 나머지보다 크기가 크다면 Sync를 한 것이 아닌 상태에서 Close를 시도한 것이거나 데이터를 쓰다가 Sync를 하고 그 후에 버퍼에 다시 데이터를 Write한 경우이므로 Sync연산을 수행한 후에 EEPROM과 버퍼에 Open된 정보를 삭제한다.

(5) Write

Write는 파일에 Write를 수행하는 연산으로 Open상태를 전재로 한다. 파일이 WRITE_APPEND 또는 WRITE_FIXED로 Create되어 있는지를 확인하고 해당 모드에 따라 Write를 진행한다. 두 모드 공통으로 우선 버퍼에 데이터가 꽉 찰 경우 페이지에 Write를 실행하며 해당블록이 다 찼을 경우 WRITE_APPEND의 경우 새로운 블록을 할당 받아서 Write를 계속하고 WRITE_FIXED의 경우 지정된 블록에 계속 Write연산을 수행하고 지정된 블록을 전부 쓰면 더 이상 Write연산은 마친다.

(6) Read

Read는 파일의 데이터를 특정 Byte만큼 읽는 명령어로 Open된 상태를 전제로 한다. 파일이 Open되어 있다면 파일을 Read하면서 블록의 끝에 오면 다음블록의 번호를 보고 해당블록의 첫 번째 페이지부터 계속 Read한다.

(7) Seek

Seek은 파일 포인터를 특정 위치로 이동하는 명령어로 Open된 상태를 전제로 한다. 파일이 Open되어 있고 주어진 Offset에 따라 해당 Byte만큼 Read를 한다.

(8) GetRemaining

GetRemaining은 파일이 현재 위치부터 얼마나 더 읽을 수 있는지 Byte의 크기를 반환해 주는 명령어로 Open된 상태를 전제로 한다. 파일의 전체 크기에서 현재 읽은 만큼의 Byte를 뺀 값을 반환한다.

(9) Sync

Sync는 버퍼의 내용을 플래시 메모리에 기록하는 명령어 로 Open된 상태를 전제로 한다. 현재 페이지를 플래시 메모리에 Write하고 버퍼에 파일의 총 사이즈를 기록한다.

(10) Dir

Dir은 파일 시스템에 저장되어 있는 파일의 리스트를 반환해 주는 명령어이다. 파일 디스크립션 맵을 순환하며 파일의 리스트를 반환한다.

나. 파일시스템 컴포넌트 구조

파일시스템의 컴포넌트 구성도는 Eclipes 3.1버전에서 TinyOS 플러그인을 설치하여 아래와 같은 그래프로 나타내었다.

그림 6은 NAND플래시 파일시스템의 컴포넌트 구성도이다. Create, Delete, Read, Write의 파일시스템 관련 컴포넌트와 EEPROMAccess, NandAccess의 EERPOM과 NAND관련 컴포넌트 그리고 파일 디스크립션의 정보를 담당하는 MetaData 컴포넌트, FreeBlock 맵과 Garbage 프로세싱을 담당하는 FreeList 컴포넌트, 마지막으로 BufferAllocate컴포넌트로 구성되어 있다.

BufferAllocate컴포넌트는 실제로 사용할 버퍼의 생성과 버퍼관리를 담당한다. MetaData컴포넌트는 디스

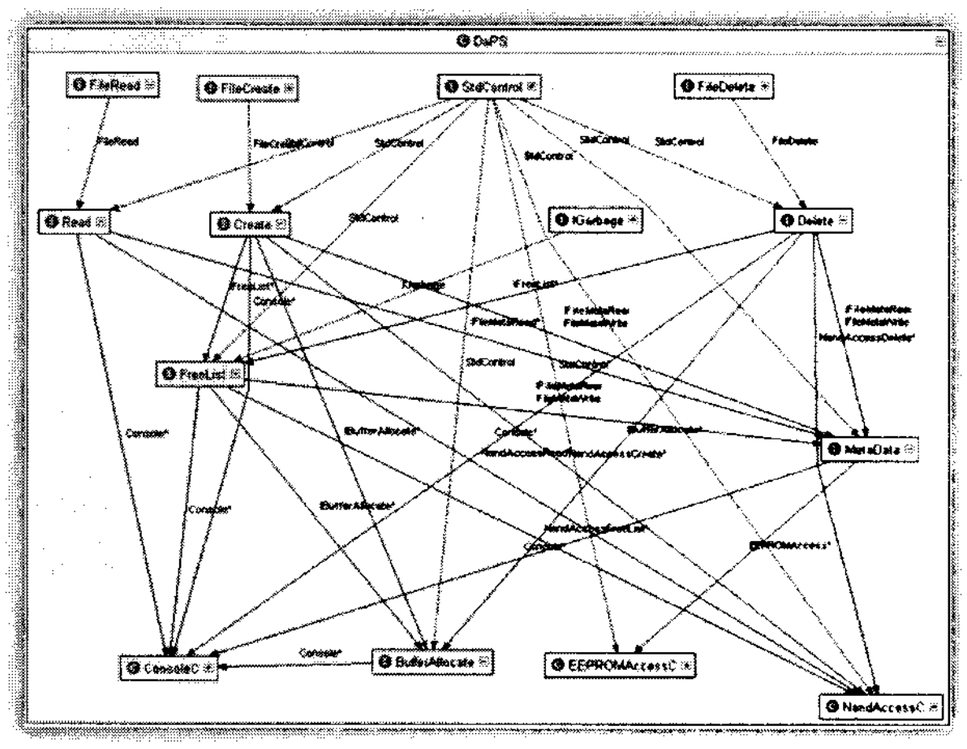


그림 6. 파일시스템을 위한 TinyOS 컴포넌트
Fig. 6. TinyOS components for the file system

크립션의 정보를 관리하고 EEPROM의 Read와 Write 쪽을 담당하는 컴포넌트이다.

Write의 경우 파일 디스크립션의 값을 저장하고 변경하기에 MetaData컴포넌트와의 관계를 가지고, NAND 플래시 메모리의 스페어 영역에 값을 저장하기에 NandAccess컴포넌트와의 관계도 가진다. 그리고 사용할 버퍼를 할당받기위해 BufferAllocate 컴포넌트와의 관계도 가진다. 이렇듯 각각의 컴포넌트들은 사용하는 관계에 따라서 그림 6과 같은 그래프로 나타난다.

다. 파일 시스템의 사용

위에 구현된 파일시스템을 사용하여 온도 및 습도 데이터를 저장하는 어플리케이션을 테스트해보았다.

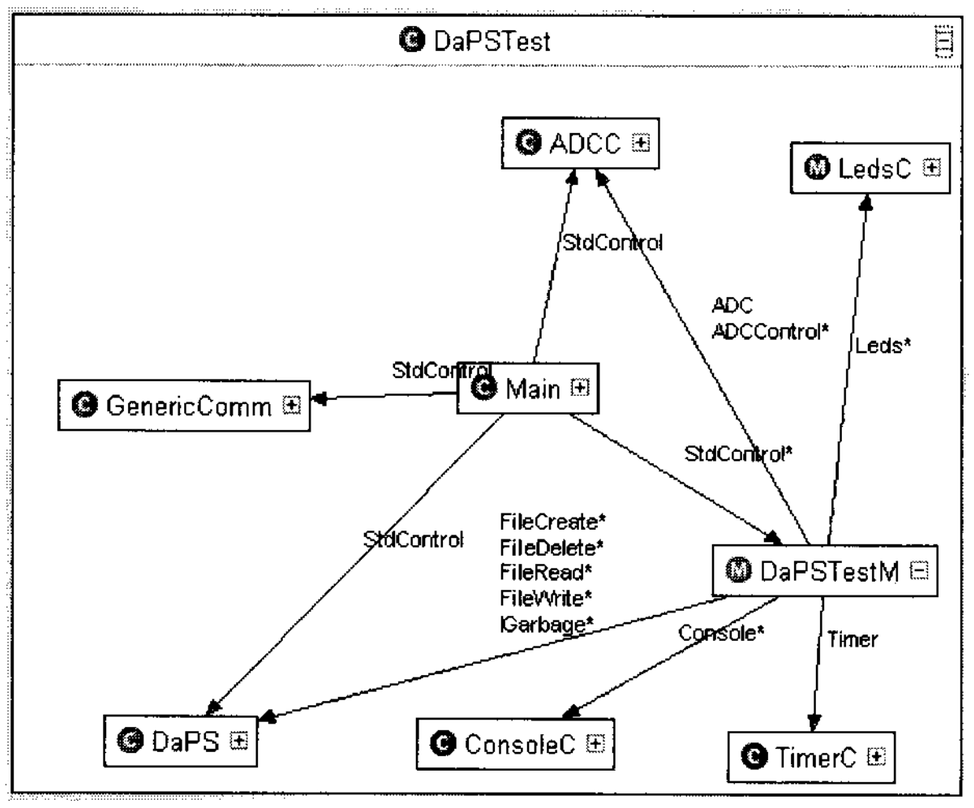


그림 7. 파일 시스템을 사용하는 전형적인 TinyOS 응용
Fig. 7. Typical TinyOS application to use the file system.

그림 7은 본 파일 시스템을 이용해서 측정된 온도 및 습도 데이터를 저장하는 프로그램의 컴포넌트 관계 그래프이다.

DaPSTestM이라는 컴포넌트는 NAND플래시 파일 시스템을 사용하여 파일의 생성과 삭제 그리고 Read와 Write를 사용한다. 파일을 생성하고 해당 파일에 센서 데이터를 Write하는 방식으로 이루어진다.

Ⅳ. 결 론

현재 센서 네트워크에서는 점차 센서 노드에 대용량의 저장 시스템이 요구되는 추세이므로 기존의 저장소로 많이 사용되었던 NOR 플래시 보다 대용량화가 쉬운 NAND 플래시 기반의 센서 노드가 늘어날 것으로 예상된다. 본 논문에서는 앞으로의 센서 노드용 저장소로 가장 적합한 대용량 NAND 플래시 기반 파일 시스템의 설계에 대해 논하였다.

본 파일 시스템은 자원과 에너지의 효율성, 대용량의 지원 및 신뢰성을 극대화하는 방향으로 설계되었으며, 매우 작은 크기의 EEPROM을 사용하여 센서 노드에서 제공하는 작은 크기의 메인 메모리의 한계를 극복하였다. 특히 효과적인 플래시 메모리의 쓰기방법과 웨어 레벨링 방법을 제안하였는데, 이는 플래시 메모리의 덮어쓰기가 불가능한 물리적인 단점을 극복하였으며, 쓰기 횟수가 제한되어 있는 플래시 메모리의 페이지 쓰기 횟수를 절약한다. 우리는 이러한 파일 시스템을 시뮬레이터를 통해 구현하였다.

본 논문에서 제안한 효율성을 높이는 여러 가지 방법을 센서 노드용 파일 시스템에 적용한다면, 앞으로의 센서 노드용 파일 시스템의 성능 향상에 적지 않은 기여를 할 것으로 기대한다.

참 고 문 헌

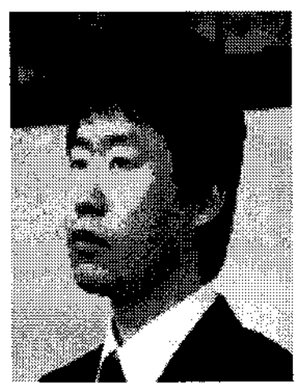
[1] Eran Gal and Sivan Toledo, “Algorithms and Data Structures for Flash Memories”, ACM Computing Surveys Vol 37, Issue 2, pp138-163, 2005.

[2] Gaurav Mathur, Peter Desnoyers, Deepak Ganesan and Prashant Shenoy, “Capsule: An Energy-Optimized Object Storage System for Memory-Constrained Sensor Devices”, Proceedings of the Fourth ACM Conference on

Embedded Networked Sensor Systems (SenSys), Boulder CO, November 1-3, 2006.
[3] D.Gay. "Design of Matchbox : The simple Filing system for Motes". In TinyOS 1.x distribution, http://www.tinyos.net, Aug. 2003.
[4] P. Levis, S. Madden, J. Polastre, et al. "TinyOS: An Operating System for wireless Sensor networks". In Ambient Intelligence, Springer-Verlag, 2005.
[5] H.Dai, M.Neufeld, and R.Han. "ELF: An efficient Log-structured flash file system for micro sensor nodes". In SenSys,page 176-187,New York NY, 2004.
[6] TIP7xx Series manual, http://www.maxfor.co.kr
[7] 128M x 8Bit NAND flash memory, Samsung Electronics.

저 자 소 개

손 기 락(정회원)
1984년 서울대학교 계산통계학과 학사.
1986년 서울대학교 계산통계학과 석사.
1993년 미국, Univ. of California, Santa Cruz, 전산학 박사.
1994년~1996년 전자통신연구원 선임연구원.
1996년~현재 한국외국어대학교 컴퓨터및정보통신공학부 교수.
<주관심분야 : 데이터베이스, 데이터마이닝>



한 경 훈(학생회원)
2002년 한국외국어대학교 컴퓨터공학과 학사.
<주관심분야 : 알고리즘, 디자인 패턴>

최 원 철(학생회원)
2006년 한국외국어대학교 컴퓨터공학과 학사.
2008년~한국외국어대학교 컴퓨터공학과 석사.
<주관심분야 : 데이터베이스, 센서네트워크, 파일시스템>

한 형 진(학생회원)
2007년 한국외국어대학교 컴퓨터공학과 학사.
2008년~한국외국어대학교 컴퓨터 공학과 석사.
<주관심분야 : 데이터베이스, 유비쿼터스>

한 지 연(학생회원)
2005년~한국외국어대학교 컴퓨터공학과 학사.
<주관심분야 : 알고리즘, 웹, 유비쿼터스>

이 기 혁(학생회원)
2007년 한국외국어대학교 컴퓨터공학과 학사.
2008년~한국외국어대학교 컴퓨터공학과 석사
<주관심분야 : 데이터베이스, 유비쿼터스>