# LFM 기법을 이용한 플래시 메모리 스와핑 파일 시스템 설계

## A Design of a Flash Memory Swapping File System using LFM

한 대 만* 구 용 완**
Han Dae Man Koo Yong Wan

### 요 약

플래시 메모리는 NOR 형과 NAND 형의 플래시 메모리 형태로 구분 할 수 있다. NOR 형태의 플래시 메모리는 빠른 읽기 속도와 Byte I/O 형태를 지원하기 때문에 ROM BIOS 와 같은 코드저장용으로 개발되어 진다. NAND 형태의 플래시 메모리는 NOR 형태의 플래시 메모리 보다 값이 싸고, 임베디드 리눅스 시스템의 대용량 처리 장치 등에서와 같이 폭 넓게 사용되고 있다.

본 논문에서는 NAND 형태의 플래시 메모리를 이용하여 시스템의 성능을 저하 시키는 Swapping을 감소시키고, 수행시간을 보장할 수 있는 플래시 메모리 Swapping 알고리즘을 제안하여, 임베디드 시스템을 기반으로 하는 파일시스템을 설계한다. 실험 과 플래시 파일 시스템 구현을 통하여 임베디드 시스템에서 요구하는 NAND 형 플래시 파일 시스템의 성능을 개선한다.

### Abstract

There are two major type of flash memory products, namely, NAND-type and NOR-type flash memory. NOR-type flash memory is generally deployed as ROM BIOS code storage because if offers Byte I/O and fast read operation. However, NOR-type flash memory is more expensive than NAND-type flash memory in terms of the cost per byte ratio ,and hence NAND type flash memory is more widely used as large data storage such as Embedded Linux file systems.

In this paper, we designed an efficient flash memory file system based an Embedded System and presented to make up for reduced to Swapping a weak System Performance to flash file system using NAND-type flash memory, then proposed Swapping algorithm insured to an Execution time. Based on Implementation and simulation studies, Then, We improved performance bases on NAND-type flash memory to the requirement of the embedded system

☞ Keyword : Flash File System, Embedded Linux, File System, Yaffs, JFFS

## 1. 서 론

임베디드 시스템은 시스템의 자동화, 무인 제어 등을 위해 도입된 이후 공장 등 산업 생산 시설 등에서 오늘까지도 많이 이용되고 있다. 하지만 이러한 임베디드 시스템은 정해진 분야에서 정해진 일만을 하는 시스템으로 오랫동안 인식되어 왔기 때문에, 일반인들에게는 매우 생소한 단어로 실제 집안 곳곳에서 사용되고 있음에도 불구하고 본인들과는 전혀 관계가 없는 시스템만으로 생각되어 왔다. 최근 모바일 컴퓨팅 환경으로 컴퓨터의 사용범위가 전이함에 따라, 휴대가 용이하고 저전력으로 동작하며 용이하게 활용할 수 있는 임베디드 시스템이 요구되어 지고 있다. 임베디드 시스템으로 PDA, 무선 인터넷 및 통신 단말기 등 많은 제품들이 시장에 출시되고 있는데 이러한 제품들은 크기 및 전원 공급 장치의 용량이 작기 때문에 시스템을 제어하는 커널 과 비 휘발성 데이터를 저장하는데 있어서 플래시 메모리를 데이터 저장장치로 많이 사용하고 있다. 플래시 메모리가 많이 사용되고 있는 이유는 비 휘발성 메모리로서

* 정 회 원 : 수원과학대학 강사
han38@hanmail.net
** 종신회원 : 수원대학교 IT대학 학장 , 컴퓨터학과 교수
ywkoo@suwon.ac.kr
[2005/02/17 투고 - 2005/03/09 1차 - 2005/04/25
2차 - 2005/06/22 심사완료]

램과 비교했을 때 집적도가 높으며 시스템에서의 쓰기 및 삭제가 가능하고, 낮은 소비 전력, 비교적 빠른 읽기와 쓰기 그리고 온도 및 충격에 강한 내구성을 보이기 때문이다. 이러한 플래시 메모리의 특징을 구현하기 위하여 각 제조 회사에서는 서로 상이한 기술을 이용하여 플래시 메모리를 제조 하였으며 어떤 기술을 기반으로 하였느냐에 따라 쓰기 및 삭제에 소요되는 시간, 데이터 블록 및 페이지의 크기 등에서 차이가 난다. 하지만 플래시 메모리를 저장장치로 활용하기 위해서는 몇 가지 설계상의 고려할 사항이 있다. 플래시 메모리를 이용하여 데이터를 쓰기 위해서는 비교적 수행시간이 길고 큰 단위로 이뤄지는 삭제연산을 선행해야 한다는 단점이 있다. 이러한 단점을 극복하기 위해 플래시 메모리를 설계하는 방법이 하드웨어적 구조의 특징 요소로 인하여 두 가지로 구분되어 진다. 플래시 메모리는 일족의 EEPROM(Electrically Erasable Programmable ROM)으로 크게 바이트 I/O를 지원하는 NOR형과 페이지 I/O만을 지원하는 NAND형 구조로 나뉘어 진다. NOR형 플래시메모리는 읽기 속도가 바른데 반하여 쓰기 속도가 느려 주로 코드 저장용 장치로 사용하고, NAND형 플래시 메모리는 쓰기 속도가 빠르고 단위 공간 당 단가가 낮아서 주로 대용량 데이터 저장 장치로 사용한다[3,13]. 표 1은 플래시 메모리와 다른 메모리 소자와의 성능상의 특성을 비교한 것으로 비교적 낮은 단가로 빠른 읽기 속도를 제공하는 영속성 있는 저장장치임을 보인다.

〈표 1〉 메모리 소자의 특성 비교

| | Read | Write | Erase | Cost/MB |
|---|---|---|---|---|
| DRAM | 60ns(2B)<br>2.6us(512B) | 60ns(2B)<br>2.6us(512B) | - | 30-40 |
| NOR-type Flash | 150ns(1B)<br>15us(512B) | 211us(1B)<br>3.5ms(512B) | 1.2s<br>(128KB) | 20-30 |
| NAND-type Flash | 10us(1B)<br>36us(512B) | 226us(1B)<br>266(512B) | 2ms<br>(16KB) | 10-20 |
| Disk | 12.4ms(512B) | 12.4ms(512B) | - | 1 |

그러나, 플래시 메모리에서 데이터를 쓰기 위해서는 삭제연산을 선행해야 하며 삭제 단위는 쓰기 단위보다 크다. 이러한 특성은 플래시 메모리를 주 메모리로 사용하는 것을 어렵게 하고, 보조기억장치로 사용하는 경우에도 일반 하드디스크용 파일 시스템을 그대로 활용하는 것을 저해한다. 그래서 삭제연산을 감추기 위해서 파일 시스템과 플래시 메모리 사이의 미들웨어인 플래시 변환 계층(Flash Translation Layer, FTL)이 제안되어 사용하고 있다[4-7]. FTL은 쓰기 연산 시에 파일 시스템이 생성한 논리주소를 플래시 메모리상의 이미 삭제연산을 수행한 영역에 대한 물리 주소로 변환하는 역할을 수행한다. 빠른 주소변환을 위해 주소변환 테이블은 비교적 단가가 높은 SRAM을 이용해 구성한다. FTL을 사용함으로써 호스트에서는 FAT과 같은 다른 파일시스템을 이용할 수 있도록 네트워크 파일 시스템을 구성하여 플래시 메모리 파일 시스템을 실시간 데이터베이스를 이용할 수 있도록 설계한다. 하지만 플래시 메모리는 실시간 데이터베이스와 같은 특수한 응용을 제외하면 주 메모리가 아닌 보조기억 장치로 사용한다. 이 경우 FTL을 사용하지 않고도 Log- Structured File System(LFS)와 같은 구조의 파일 시스템을 이용하여 플래시 메모리의 성능을 개선할 수 있다[8,9,11,12]. 플래시 메모리를 구성하는 파일 시스템 설계 방법의 다른 한 방법으로 전용 플래시 파일 시스템을 구성할 수 도 있다. 플래시 메모리 전용 파일 시스템을 구성할 경우 부과적인 하드웨어와 추가 비용 없이 연산 속도를 개선하였다. 플래시 메모리의 쓰기 연산은 삭제 연산을 선행하기 때문에 많은 오버헤드를 갖는다. 본 논문에서는 플래시 메모리의 쓰기 연산 시 발생하는 오버헤드를 최소화 하도록 하고, 메모리의 활용도와 접근 시의 문제를 해결하기 위해 사용되고 있는 대표적인 방법 중에서 자연적으로 발생하는 시간적 지역성과 공간적 지역성을 활용한다. 페이지폴트의 발생률을 최소화 하도록 LFM 기법을 적용하여 Swap 영역을 플래시메모리에 설정하고 최근의 데

이터로부터 순차적으로 Swap-in될 수 있도록 LFM 기법을 사용하도록 한다. LRM 기법 사용 시 페이지 폴트 된 데이터를 블록 재배치 방법을 사용하면 Swap 영역의 최근 데이터는 사용 빈도가 높은 페이지를 항상 빠른 접근 속도로 Swap- in 할 수 있기 때문에 응용 프로그램의 페이지 폴트로 발생하는 플래시 메모리의 쓰기 연산을 최소화 할 수 있다. 본 논문에서는 2장 관련연구에서 리눅스에서 사용하고 있는 플래시 전용 파일시스템인 JFFS, JFFS2, YAFFS 파일 시스템에 대하여 살펴보고, 3장 구현부분에서는 각각의 장단점을 분석하고 리눅스를 운영체제로 사용하는 임베디드 시스템에서 사용할 수 있는 플래시 파일 시스템을 설계하며, 4장에서는 구현결과를 기반으로 성능평가를 나타내었다.

## 2. 관련연구

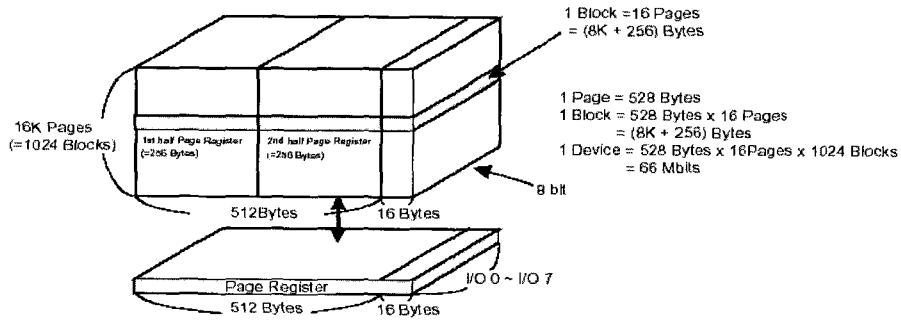
### 2.1 NAND 플래시 메모리 구조

NAND형 플래시 메모리는 다른 플래시 메모리에 비해 가격이 싸고 용량이 크며 연속된 데이터를 읽어오는 속도가 빠르기 때문에, 현재 MP3, 디지털 카메라, 이동식 저장장치 등 많은 디지털 기기에서 사용 되고 있다. 그림 1에서는 NAND 플래시 메모리의 기본 구조를 나타내고 있다. 플래시 메모리는 페이지 단위로 읽거나 쓸 수 있고 블록(여러 개의 페이지)단위로 주울 수 있다. 우리가 사용하는 32MB 플래시 메모리의 경우 한 페이지는 512B + 16B(Spare 영역)이고, 블록은 32개의 페이지이며, 이 블록이 2048개 모여서 전체를 이룬다.

플래시 메모리의 데이터를 읽으면 메모리 어레이에서 해당 페이지의 데이터를 페이지 버퍼로 옮긴다. 그 후 컬럼 주소에 따라 8 bit(1 byte)씩 데이터를 출력하게 된다. 이때, 페이지 버퍼로 데이터를 옮길 때 max 10 sec가 소요 된다. 그러나 한 페이지의 내용을 연속해서 읽어 올 때는 페이지 버퍼에서 바로 바로 데이터를 내 보내므로 최소 50 nsec 주기로 8 bit씩 읽어올 수 있다[1,12].

### 2.2 플래시 디바이스 드라이버 모듈

리눅스에서 플래시 메모리 디바이스를 제어하기 위해 플래시 메모리의 저수준 인터페이스 동작을




| | I/O 0 | I/O 1 | I/O 2 | I/O 3 | I/O 4 | I/O 5 | I/O 6 | I/O 7 | |
|---|---|---|---|---|---|---|---|---|---|
| 1st Cycle | A0 | A1 | A2 | A3 | A4 | A5 | A6 | A7 | Column Address |
| 2nd Cycle | A9 | A10 | A11 | A12 | A13 | A14 | A15 | A16 | Row Address |
| 3rd Cycle | A17 | A18 | A19 | A20 | A21 | A22 | *X | *X | (Page Address) |

NOTE : Column Address : Starting Address of the Register.
00h Command(Read) : Defines the starting address of the 1st half of the register.
01h Command(Read) : Defines the starting address of the 2nd half of the register.
* A is set to "Low" or "High" by the 00h or 01h Command.
* X can be High or Low.

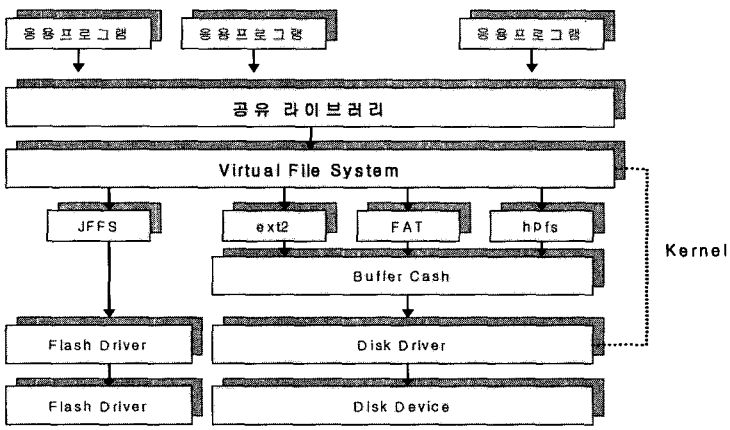
〈그림 1〉 NAND 플래시 메모리의 구조

제공하는 디바이스 드라이버가 필요하며 이를 통해 플래시 디바이스가 서비스할 수 있도록 파일 시스템 인터페이스와 결합시켜 준다. 디바이스 드라이버는 NOR형 플래시와 NAND형 플래시의 특성에 맞게 사용하는 드라이버 모듈이 다르다. NOR형 플래시 메모리의 경우의 읽기/쓰기 동작은 일반 메모리와 유사하며 바이트 단위의 프로그래밍이 가능하다. 따라서 플래시 메모리에 접근할 때 해당 주소에 있는 데이터를 읽어 오기만 하면 된다. 하지만 NAND형 플래시 메모리는 NOR형 플래시 메모리와는 달리 8비트 I/O모드로 동작하는데, 읽거나 쓰기를 할 때는 페이지(512Byte)단위로 동작하며 지우는 동작은 블록 단위(16page)로 동작한다. 플래시 메모리에 대한 쓰기 동작은 NAND형 플래시 메모리 디바이스 특성상 해당 페이지가 속해 있는 블럭을 모두 지운 다음 쓰기를 수행해야 하는 플래시의 특성을 감안해 프로그래밍 해야 한다. 그림 2은 파일 시스템의 구성도이다. 리눅스에서 사용하는 일반적인 파일 시스템은 디스크 드라이버를 기반으로 하며 플래시 파일시스템은 플래시 드라이버를 기반으로 하고 있다. 그림 2의 VFS에서는 지원 가능한 다양한 파일시스템을 등록하여 사용할 수 있도록 구성되어 있다. 본 논문에서는 JFFS를 사용한 부분에 JFFS2와 Yaffs, 그리고 Swap 공간을 가지고 있는 Yaffs를 구성하여 성능을 평가 하도록 한다.

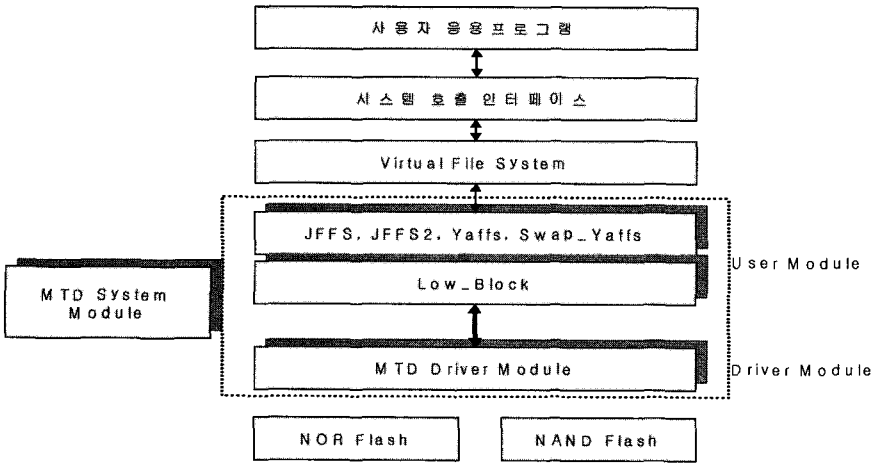
## 2.3 MTD(Memory Technology Device)

임베디드 리눅스 장비에서 램 디스크를 이용하여 루트 파일 시스템을 구현 하였을 경우에는 보드 동작 중에 파일로 기록된 내용이 전원이 꺼짐과 동시에 소실된다. 기록된 내용을 영구 저장하기 위해서는 일반적으로 플래시 메모리에 기록하여야 한다. 플래시 메모리를 리눅스의 루트 파일 시스템으로 사용하기 위해서는 MTD 블록 디바이스 드라이버를 사용한다.

MTD 시스템은 리눅스 시스템에서 플래시 메모리 디바이스를 이용하기 위해 개발되었다. 각 제조사에 따라 다른 플래시 메모리 디바이스에 대해 일관되고 공통된 소프트웨어 인터페이스를 제공함으로써 플래시 메모리를 이용한 리눅스 시스템 개발에 개발자가 손쉽게 대처할 수 있게끔 하고 있다. 하드 블록 디바이스 같은 일반적인 저장매체를 사용하기 힘든 임베디드 시스템에서 플래시 메모리는 유일하게 사용할 수 있는 저장매체이며, 플래시 메모리를 이용한 리눅스 임베디드 시스템 개발에 MTD는 좋은 인터페이스로 활용할 수 있다. MTD는 사용자 모듈과 드라이버 모듈로 구성되어 있으며 전용 플래시 메모리 파일 시스템




〈그림 2〉 파일 시스템의 구성도




〈그림 3〉 플래시 메모리 블록 디바이스를 위한 소프트웨어 구성

을 적용할 수 있다. 그림 3에 MTD의 구성도를 나타내었다. MTD시스템 구성 모듈로 드라이버 모듈로는 MTD를 사용하였고 로우 블록에 JFFS와 같은 플래시 전용 파일 시스템을 적용하여 사용할 수 있다. MTD에서는 두 가지 방법으로 사용자 프로그램에 플래시 메모리를 접근할 수 있는 인터페이스를 제공한다. 문자 디바이스로 접근하는 경우와 블록 디바이스로 접근하는 경우이다. 플래시 메모리를 문자 디바이스로 접근할 때는 플래시 메모리를 하나의 파일 인식해 순차적으로 읽고 쓸 수 있다. 플래시 메모리를 블록 디바이스로 접근한다면 블록 단위의 임의 접근이 가능하다. 블록 디바이스의 경우 마운트를 통해 메타 정보 데이터를 메모리에서 읽어오며 해당 파일 시스템의 API의 모듈을 로드 한다. 그림 3는 플래시 메모리를 블록 디바이스로 접근하는 경우의 소프트웨어 계층 구조다. 그림 3 처럼 사용자 응용 프로그램이 시스템 호출을 사용해 사용자 요구는 가상 파일 시스템을 통해 플래시 메모리를 위해 구축된 파일 시스템(JFFS)에 접근하게 된다. JFFS는 갑자기 전원 공급이 중단되는 상황에서도 신뢰성 있는 시스템을 제공하기 위한 것이다. 물론 ext2등의 다른 파일 시스템을 통해서도 접근할 수 있지만 JFFS를 사용해 플래시 메모리에 접근하는 이유는 JFFS가 플래시 메모리의 특성에 알맞게 설계됐기 때문이다. 드라이버 모듈은 플래시 메모리를 읽고 쓸 수 있도록 하는 장치 구동기다. JFFS를 비롯한 일반적인 파일 시스템은 블록 디바이스를 필요로 하는데, 플래시 메모리는 일반적으로 블록 디바이스가 아니므로 로우블럭(Low-Block)모듈을 두어 플래시 메모리가 블록 디바이스로 보이도록 했다.

## 2.4 JFFS(Journaling Flash File System)

JFFS는 그 이름에서 알 수 있듯이 쓰기의 경우에는 데이터를 로그형태로 순차적으로 기록하고 읽기의 경우에는 로그를 역순으로 검색해 가장 최신의 데이터를 읽는다[6]. 쓰기연산 이후에 임계용량 이상을 사용하게 되면 앞부분 블록부터 순차적으로 삭제연산을 할 경우 쓰레기 모음을 수행해 빈 공간을 확보한다. 이때 삭제 블록내부에 존재하는 유효한 페이지는 다시 로그형태로 기록해 보존한다. 하지만 이와 같이 삭제연산을 순차적으로 수행하는 것은 유효한 데이터의 비율 등을 고려해 최적의 블록을 선택적으로 삭제하는 방법에 비하여 비효율적이다. 이를 개선하기 위해 JFFS2가 개발되었다. 삭제연산을 효율적으로 수행하는 것 이외에도 JFFS2 에서는 단위 공간 당 단가가 높은

플래시메모리의 공간을 효율적으로 활용하기 위해 압축기능을 사용하고, 보다 다양한 종류의 inode 구조를 지원해 몇 가지 장점을 취한다[7].

### 2.5 Yaffs(Yet another Flash File System)

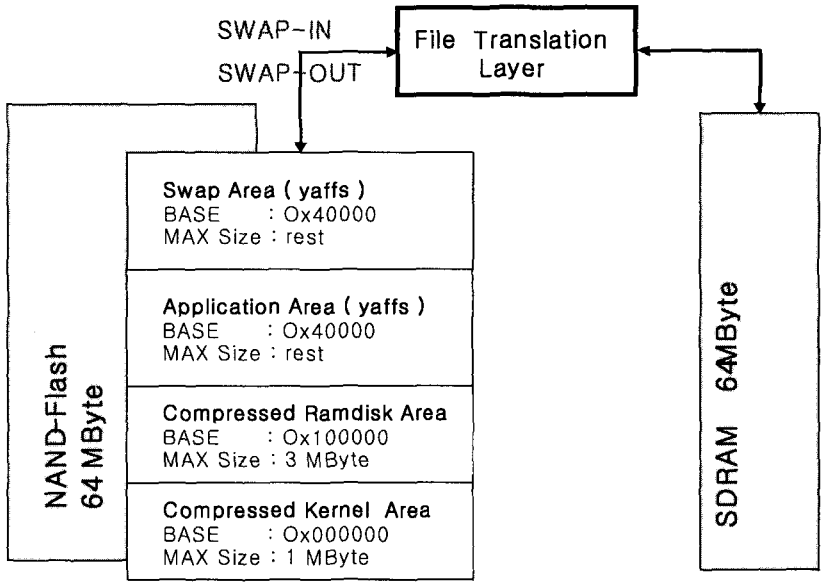
기존의 임베디드 시스템에 리눅스를 사용하게 된 것은 파일시스템을 사용자 임의로 등록하여 사용할 수 있다는 장점이 있기 때문이다. 대부분의 NOR 타입의 파일 시스템은 MTD와 JFFS 시스템을 등록하여 사용한다. 하지만 일부에서 실제 사용중에 JFFS 파일 시스템의 불안정성이 지적되고 있다. 특히 용량이 커지면 쓰기 속도에 문제가 발생하고 메모리 점유율이 상당히 증가하는 단점을 가지고 있어. 임베디드 시스템의 응용 프로그래머에게 제한적인 프로그래밍을 요구한다. 이러한 단점을 Yaffs 파일 시스템에서는 JFFS의 단점을 극복할 수 있도록 구성하였다. 메모리 사용 효율이 JFFS2 보다는 뒤떨어지지만 시스템 마운트 속도에서는 월등하다는 것이 증명되었다. 따라서, 본 논문에서는 Yaffs 파일 시스템을 도입하여 Yaffs 파일시스템의 메모리 사용 효율을 높일 수 있는 방안보다는 NAND 플래시 메모리의 특징적 단점을 보안할 수 있는 파일 시스템을 설계하도록 한다.

### 2.6 블록 재사용 기법

플래시메모리는 블록 단위로 지워지기 때문에 한 블록의 일부분만 지울 수 있는 방법을 제공해야만 한다. 이 과정을 블록 재배치 또는 쓰레기 모음이라고 하는데, 이는 파일 시스템과 미디어의 포맷에 따라 블록 기반으로 이루어지거나 파티션 기반으로 이루어진다. LFM에서의 블록 재배치는 블록내의 데이터를 가지고 있는 파일 시스템의 재배치 보다 훨씬 복잡하다. 각 블록을 스페어 블록에 복사하고 원래의 블록을 지운 뒤 스페어 블록내의 유효한 정보들을 다시 리스트의 뒤쪽에 연결해야 한다. 데이터나 헤더가 블록 경계에 의해서 제한되지 않으므로 블록 재배치 알고리즘은 블록 경계를 지나는 것을 알리는 정보들을 찾아내어야 한다. 본 논문에서도 블록 경계를 나타내기 위하여 헤더정보를 삽입하였다. 또한, 블록 재배치는 시간이 많이 소모되기 때문에 후순위로 수행되도록 하여 수행효율을 높이도록 하였다.

## 3. 플래시 메모리 전용 파일 시스템 설계

플래시메모리 전용 파일시스템과 FTL을 설계하는데 있어서 성능에 큰 영향을 끼치는 설계 요소는 '언제 어떤 블록을 삭제할 것인가'라는 정책결정 과정이다[13]. 너무 일찍 삭제연산을 수행하면 삭제블록 내부에 유효한 페이지의 수가 적어 총 수행하는 삭제연산의 수가 늘어나게 된다. 반대로 삭제연산 수행시점을 최대한 지연시키면 공간 효율성 측면에서는 좋지만 이후에 쓰기연산이 요청되고 기 삭제된 블록이 없는 경우에 그 동안 지연시킨 삭제연산을 한번에 수행해야 하기 때문에 입출력 성능을 크게 저하시킬 수 있다. 왜냐하면 최근에 플래시메모리 중에는 내부를 뱅크 형태로 분할하고 서로 다른 뱅크에 대해 삭제와 쓰기 또는 쓰기와 읽기연산을 동시에 수행할 수 있는 기능을 지원하는데, 삭제연산을 지연해 특정시점에 동시에 수행하게 되면 이러한 구조적 병렬성을 활용하기 어렵기 때문이다. 그리고 삭제할 블록을 선택하는 문제는 LFS의 기법을 응용할 수 있다[13,14] 본 논문에서는 플래시 전용 파일 시스템으로 사용되고 있는 Yaffs를 EZ-X5 보드의 커널에 이식하고 Yaffs의 소스 코드를 개선하여 기존의 Yaffs 와 SWAP-Yaffs, 그리고 JFFS , JFFS2와 비교 분석한다. MTD 시스템에 JFFS를 사용하는 것이 Yaffs를 사용하는 것보다 시스템 안전성 면에서 불안정하여 Yaffs를 사용하였다. FTL을 이용하여 SWAP-Yaffs에서는 합리적으로 메모리를 사용할 수 있는 알고리즘을 구성하여 적용하였다.



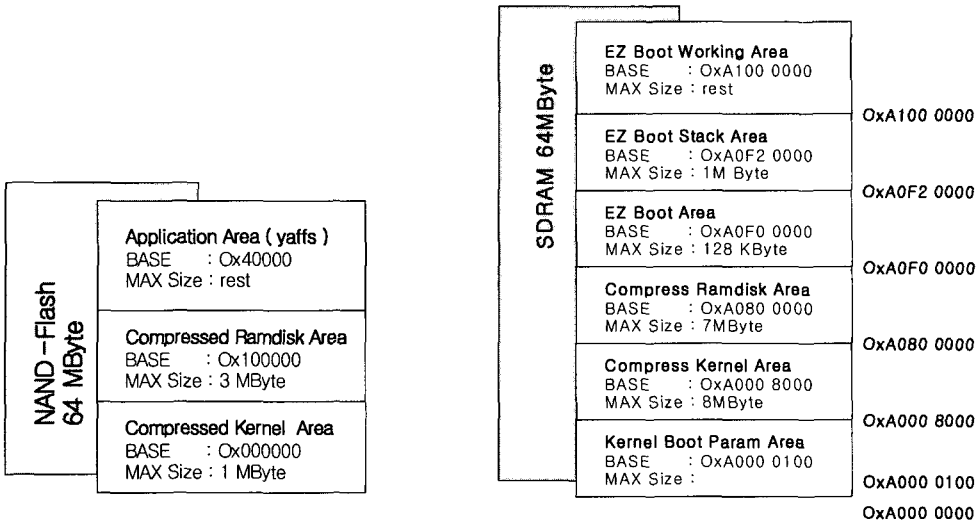

〈그림 4〉 Swap 공간을 만든 NAND 플래시 메모리

### 3.1 MTD 시스템 구성

MTD가 지원하는 플래시 메모리는 여러 가지 형태가 있다. 그 중 플래시 메모리에서 대용량으로 설계된 NAND 플래시를 지원하는 Yaffs 파일 시스템을 지원하도록 한다. MTD는 두 가지 형태의 디바이스 드라이버를 지원하는데 캐릭터 형과 블록 형이다. 캐릭터 형은 플래시 자체를 접근 가능하게 하는 부분이 구현되어 있다. 그러므로 플래시를 직접 지우거나 할 경우에는 캐릭터 형으로 접근해야 한다. 장치 파일을 만들때 주 번호는 90으로 시작하며 짝수의 부 번호는 읽기 쓰기가 가능하고 홀수의 번호는 읽기만 가능하다. 이렇게 구별한 이유는 플래시 보호 정책에 의해서 만들어 진 것이다[9].

### 3.2 SWAP-Yaffs파일 시스템 설계

임베디드 시스템에 리눅스를 사용하는 이유 중 하나는 파일 시스템을 응용 프로그래머의 의도에 따라 선택하여 사용할 수 있는 장점이 있기 때문이다. 현 임베디드 시스템에 사용되는 NOR 형 시스템의 파일 시스템은 MTD와 JFFS2를 통상적으로 사용한다. 그러나 실제 사용 중에 JFFS2 시스템의 불안정성이 문제가 된다. 특히 용량이 커지면 쓰기 속도에 문제가 발생하고 점유 메모리가 크게 늘어난다. 이런 단점을 극복하기 위하여 MTD에 Yaffs 파일 시스템을 적용하여 구현하고, Yaffs 파일 시스템을 FTL을 이용한 로직을 이용하여 파일시스템을 설계한다. MTD의 설치 방법은 각 하드웨어 기업에서 제공하는 매뉴얼을 참고 하도록 한다. 본 논문에서는 64 M Byte NAND 플래시 메모리 중 Swap 영역을 지정하여 FTL에서 실행 파일을 어플리케이션 영역으로 검색하는 것이 아니라 Swap 영역을 미리 검색하여 NAND 플래시의 검색영역을 최소화 하였다. 그림 4에 Swap 파티션을 생성한 플래시 메모리 의 구조를 나타내었다. 이는 NAND 플래시 메모리의 쓰기 연산 시 NAND 의 어플리케이션 영역으로 접근하는 시간을 최소화 하였다. 이는 NAND 플래시에서 발생하는 쓰기 연산 시 접근시간의 영향으로 시스템에 많은 영향을 주기 때문에 쓰기 연산이 최소화 되도록 Swap 영역을 할당하여 읽기 연산만을 수행




〈그림 5〉 EZ-x5 의 메모리 구조

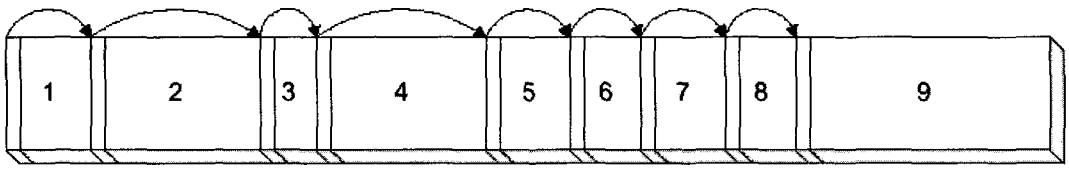
하도록 한다. 이것은 응용프로그램의 프로그램 들은 일반적으로 유사한 메모리 지역에 필요한 프로그램을 저장하는 공간적 지역성에 근거하였으며, 유사한 NAND 메모리 지역의 값을 Swap 지역으로 배치함으로서 쓰기 연산이 NAND 플래시 응용 프로그램을 순차적으로 처음부터 검색하여 메인메모리로 쓰기 하는 연산을 최소화 하였다. 따라서, Swap 영역의 값을 읽기 연산만으로 내용을 검색할 수 있도록 하였다. 플래시 메모리에서의 읽기 연산은 메인메모리에서의 연산속도와 유사한 결과를 나타낸다.

그림 5에 나타낸 바와 같이 본 연구에서는 NAND 플래시 메모리 64M Byte를 사용하고, 최초로 부팅이 되면 MCU(Memory Control Unit)의 NAND 플래시 컨트롤러에 의해 NAND 플래시 메모리의 커널 블록을 SDRAM으로 복사를 하여 커널을 실행한다. NAND 플래시의 응용프로그램 영역에 있는 프로그램은 모두 SDRAM 영역으로 복사되어 실행되는 것이 아니라 필요한 부분만 램으로 복사하여 수행하도록 한다. JFFS나 JFFS2는 NAND 형과 NOR 형 플래시 메모리를 모두 지원하고 SDRAM에 파일시스템 수행 시 요구되는 크기가 4M Byte 이며 부팅 속도는 평균 25 sec를 나타낸다.

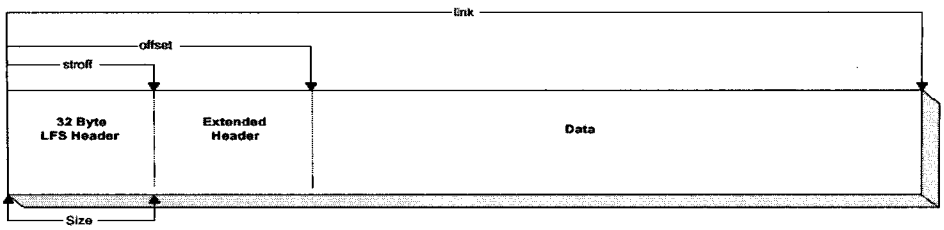
Yaffs 파일 시스템은 오직 NAND 형 플래시만을 지원하고 SDRAM에 요구되는 크기가 512 KByte 이다. 부팅속도는 평균 3sec 정도로 나타난다. 따라서, 속도 면이나 공간적인 면이나 많은 장점을 가지고 있다.

## 3.3 LFM 설계

FTL과 달리 LFM은 완전한 하나의 파일 매니




〈그림 6〉 연결리스트로 구성한 LFM



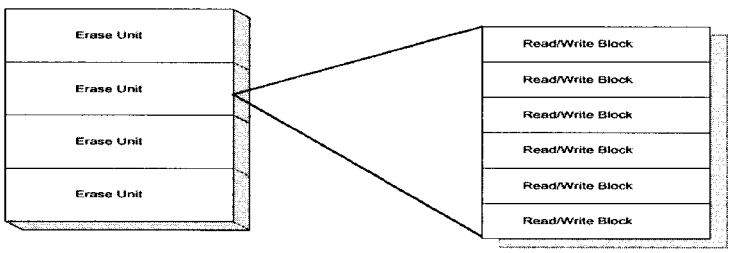

```
typedef struct lfs_header {
   ULONG    link;
   ULONG    size;
   ULONG    type;
   ULONG    offset;
   ULONG    flags;
   ULONG    stroff;
   ULONG    id;
   ULONG    reserved;
} LFS_HEADER;
```

〈그림 7〉 LFM 형태와 헤더 구조

저로 구성하였다. LFS 구조는 가변 길이의 파일 오브젝트들을 저장할 수 있는 PCMCIA에서 정의된 자료구조이다. LFM은 이러한 자료 구조를 사용할 수 있는 파일 매니저로 설계하였다. 파일 오브젝트들은 파티션 내에서 연속적으로 저장되어 있으며 단일 연결 리스트로 연결하였다. 파일 오브젝트의 처음 32byte는 LFM 헤더를 담고 있다. 헤더는 다음 파일 오브젝트에 대한 링크를 포함한 기본적인 파일 정보를 담고 있다. 그림 6에 플래시 메모리에서 LFM를 구현하고 연결리스트로 구성한 것을 나타내었다.

그림 7은 LFM 형태로 사용한 LFM 파일 오브젝트의 형태와 헤더의 필드들을 나타내고 있다. 헤더 필드의 구성내용은 데이터의 길이, off-set 위치, id, flags, 그리고, LFS 헤더를 나타내기위한 자료구조 등으로 구성하였다.

플래시 메모리에서 쓰기 연산이 발생하였을 경우, 즉 Swap 연산이 발생하여 이에 따르는 삭제 연산이 발생하였을 경우 그림 8과 같이 삭제 장치 블록이 지정되고, 삭제 장치의 연산을 위해 EUH (Erase Unit Header)가 존재한다. EUH에는 FTL 전체 파티션과 EUH가 위치한 삭제 장치에 대한 데이터를 저장 하도록 하였다. 각 삭제 장치는 자신만의 고유한 EUN를 가지고 있으며, EUH에서만 찾을 수 있다. 그리고, 일기/쓰기 블록과 마찬가지로 삭제 장치들도 성능상의 이유로 재배치되고 순차적으로 저장되지 않기 때문에 FTL이 연결을 등록 하도록 논리적 EUN도 각 삭제 장치




〈그림 8〉 Swap 발생시 생성되는 Erase Unit의 구성

의 EUH에 저장하도록 설계하였다. 또, 삭제 장치 내에 BAM(Block Allocation Map)이라는 구조를 설계하였으며, 이는 각 삭제 장치 내의 읽기/쓰기 블록 배치에 대한 정보를 나타내도록 하였다. EUH의 BAM-Offset을 참조함으로써 BAM의 위치를 찾을 수 있다. 일반적으로는 BAM은 EUH의 바로 다음에 위치한다. 그리고 운영체제의 요구를 논리적 위치로 기본적인 테이블을 VBM (Virtual Block Map)이라고 하고. BAM에서 VBM 페이지의 위치가 표시되도록 하였다.

Swap 발생시 수행해야 하는 삭제 연산을 최소화하기 위해서는 Swap 영역에 보다 많은 페이지를 위치 시켜야 한다. 높은 Hit 율을 유지하기 위해 구성한 LFM은 Swap 영역의 크기가 상당한 부분을 차지하기 때문에 성능평가에 Hit 율에 따른 SDRAM의 크기 별로 발생하는 수행시간을 나타내었다. 본 논문에서 구현한 LFM 방식은 Swap영역에서 높은 Hit 율을 유지하기 위해 재배치 알고리즘을 사용하였다.

## 4. 성능분석 및 평가

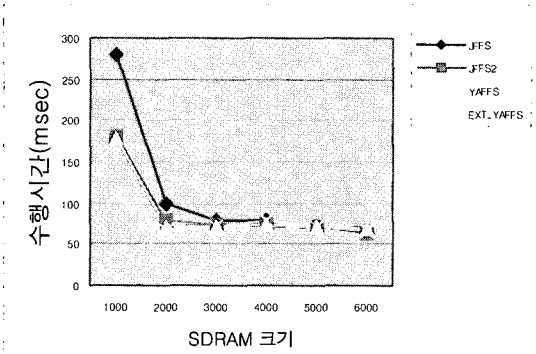
본 장에서는 NAND 플래시 메모리로 시스템을 구성할 경우, 고려해야 할 사항에 대한 성능평가를 한다. 최초의 부팅을 하면 내부 NAND 플래시 메모리 컨트롤러가 NAND 플래시 메모리의 처음 블록을 내부 램으로 복사하여 실행 시킨다. 실험용으로 사용한 EZ-X5 보드는 [그림5]와 같은 하드웨어 구조를 가지고 있다. 램으로 복사된 프로그램에 의해 커널을 NAND 플래시 메모리에서 DRAM으로 복사하고 커널을 실행하게 된다. 실행 중, 커널은 램에 상주하면서 실행되지만 응용프로그램은 RAM으로 전부 복사되지 않고 실행에 필요한 일부분만 램에 복사하여 사용한다. 실험에 사용한 하드웨어의 사양은 다음 표 2와 같다.

〈표 2〉 하드웨어 사항

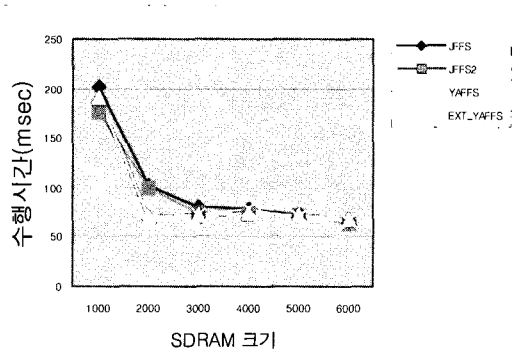
| 하드웨어 | 세부사항 | 비고 |
|---|---|---|
| MCU | 400 MHz PXA255 ARM RISC Chip | ARM10 |
| RAM | 64 Mbyte SDRAM | |
| ROM1 | 512 Kbyte Boot Flash | |
| ROM2 | 64 Mbyte NAND-Flash | |

실험방법은 Yaffs 시스템과 개선된 Yaffs시스템사이의 Swap 영역의 크기에 따라 수행시간을 측정하는 방법과 JFFS와 JFFS2 파일과의 성능평가를 보이도록 한다. 성능평가 방법 중 SDRAM의 크기조절은 하드웨어 설계 시 불가능 하므로 가상으로 조절하도록 하였으며 Yaffs 등 플래시 파일 시스템에서 메모리의 Swap 알고리즘을 제공하지 않기 때문에 성능평가 알고리즘에서 임의의 리눅스 파일 시스템에서 제공하는 Swap 알고리즘을 사용하도록 한다.

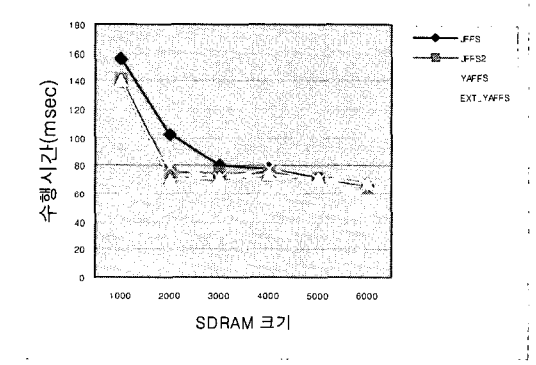
그림 9~12 의 내용은 JFFS, JFFS2, Yaffs 파일 시스템과 Swap 파티션을 사용하는 Yaffs 파일시스템간의 평가를 나타낸 것이다. Swap 영역의 크기에 따라 Swap 영역이 128KByte 인 경우와 512KByte 경우는 SDRAM의 크기가 증가함에 따라 수행시간의 효율이 증가하지 않았다. 이 결과는 Swap 파티션의 크기가 응용프로그램에서 요구하는 메모리의 크기가 작기 때문이다. 이에 반하여 Swap 크기를 1MByte, 2Mbyte로 증가시 이후의 결과를 보면 파일시스템의 종류에 관계없이 모두 성능이 향상된 것으로 나타났으며, SDRAM의 크기가 증가 하면 전체 파일 시스템이 모두 유사한 수행속도를 나타내었다. Swap 파티션의 크기가 256KByte, 512KByte인 경우 SDRAM의 크기가 2KByte 미만인 부분에서만 SWAP-Yaffs 파일 시스템의 성능이 개선된 것으로 나타난다. 위와 같은 실험결과로 볼때 메모리 요구량이 많은 경우에만 Swap 파일 시스템을 사용하는 것이 성능을 개선할 수 있는 것으로 나타났으며 메모리의 요구량이 적은 응용프로그램의 경우 즉, 정적으로 많은 데이터의 변화가 없는 응용프로그램에서는 차이가 적은 것으로 나타났다. 따라서, Swap 파티션을 사용하는 것은 동적인 메모리 사용 비율이 높을 경우에만 장점이 있을 것으로 판단된다.



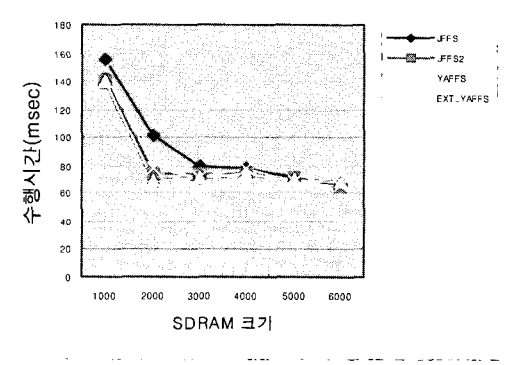

〈그림 9〉 SWAP 크기가 128KByte




〈그림 10〉 SWAP 크기가 512KByte




〈그림 11〉 SWAP 크기가 1MByte




〈그림 12〉 SWAP 크기가 2MByte

## 5. 결론 및 향후연구 과제

본 논문을 통하여 플래시 메모리를 이용하여 플래시 파일시스템을 구축하는 방법을 알아보았고, NAND 형 플래시 메모리를 구성하였으며, 연구과제로 부터 적절한 플래시 파일 시스템을 설계하였다. Swap 기능을 Yaffs 파일 시스템에 접목함으로서 JFFS2보다 메모리의 사용면에서는 성능이 개선되지 못하였지만, 성능평가 결과 플래시 메모리를 이용하는 응용프로그램의 수행속도 면에서 성능이 개선되었다. 또한 NAND 형 플래시 메모리가 순차검색에서 다른 메모리 보다 빠르다는 장점을 이용하였다. 성능평가 결과 SDRAM의 크기에 따라 각각의 파일 시스템은 Swap 영역의 크기에 상관없이 거의 유사한 결과를 나타내었다.

성능평가에 사용한 응용프로그램의 크기나 프로그램 수행 부분이외의 추가적으로 필요로 하는 메모리의 크기를 비교하는 방법이 요구된다. 또한 플래시 파일 시스템에서 제공하지 않는 Swap 기능을 Yaffs 파일 시스템에 구현하는 방법을 알아볼 수 있도록 한다.

## 참 고 문 헌

[1] AMD, Flash Memory Technical Documentation.http://www.amd.com/us-en/FlashMemory/Tech.nicalResources, August 2001

[2] D. Woodhouse, *"JFFS: The Journaling Flash File System,"* Ottawa Linux Symposium (http://sources.redhat.com/jffs2/),2001.

[3] Intel Corporation, Understanding the Flash Translation Layer(FTL) Specification, http:// dev eloper.intel.com/design/flash, December 1998.

[4] J. Kim, J.M.Kim, S.H.Noh, *"A Space-Efficient Flash Translation Layer for Compact Flash Systems."* IEEE Transactions on Consumer Electronics, Vol.48, No.2, pp.366-375, 2002.

[5] J. L. Hennessy, D. A. Patterson, David Goldberg, Computer Architecture: A Quantitative Approch, 3rd Edition, Morgan Kaufmann Publishers, 2002.

[6] JFFS, http://devleoper.axis.com/s oftware/jffs/.

[7] JFFS2, http://sources.redhat.com/jffs2/

[8] K.S.Yim, H. Bahn, K. Koh, *"A Compressed Page Management System NAND-type Flash Memory,"* In Proceedings of the International Conference on VLSI, pp. 266-271, 2003.

[9] MTD, *"Memory Technology Device (MTD) sub-system for Linux,"* http://www.linux-mtd.infradead.org/.

[10] N. Webber, *Operating System Support for Portable File system Extensions*, Proceedings of the Winter USENIX 1993 Technical Conference, 219-228, January 1993.

[11] Samsung Electronics, SAMSUNG NAND Flash Memory, Memory Product & Technology Divi sion, 1999.

[12] Samsung Electronics, "128M x 8 bit / 64M x 16 bit NAND Flash Memory," http:// www.san sungelectronics.com/

[13] 임근수, 고건 " *A Study on Flash Memory Based Storage Systems Depending on Design Techniques."* 한국정보과학회 논문집 Vol 8, No 1, pp, 36-45, 2003.

[14] 임근수, 양훈모, 차호정 "An Extended Flash Memory Swapping System Based on Selective Compression." 한국정보과학회 논문집 Vol. 5, No2. pp 304-308 ,2002.

## ◐ 저 자 소 개 ◑

**한 대 만 (Han Dae Man)**
1998년 독학사 대학교 전자계산학과 졸업(학사)
2000년 수원대학교 대학원 전자계산학과 졸업(석사)
2003년 수원대학교 대학원 전자계산학과 수료(박사)
2002 ~ 현재 수원과학대학 강사
관심분야 : 분산 및 운영체제, 임베디드 시스템, 실시간 리눅스 시스템,
E-mail :han38@hanmail.net

**구 용 완 (Koo Yong Wan)**
1976년 중앙대학교 전자계산학과 졸업(학사)
1980년 중앙대학교 대학원 전자계산학과 졸업(석사)
1988년 중앙대학교 대학원 전자계산학과 졸업(박사)
1983년 ~ 현재 수원대학교 IT대학 학장 , 컴퓨터학과 교수
관심분야 : 분산 및 운영체제, 실시간 리눅스 시스템, 시스템 네트워크 관리,
임베디드 시스템, 인터넷 응용 등
E-mail : ywkoo@suwon.ac.kr