|  |
| --- |
| skku |
| 인턴십 종합 보고서 |
| Sector Level Mapping FTL 구현 |
|  |
| **오기환, 한규화, 홍경환** |
| **2012-01-28** |

|  |
| --- |
| 동계 방학 중 해외 인턴십 목적으로 미국 캘리포니아 산 호세에 위치한 OCZ Technology내의 인디링스 팀에서 Open-SSD 프로젝트의 자스민 보드(Jasmine board)를 활용하여 섹터 단위 매핑 FTL(Sector Level Mapping FTL)을 구현하였다. |

Index

[그림 목차 3](#_Toc315364850)

[프로젝트 진행 일정 및 역할 5](#_Toc315364851)

[구현 상세 설명 6](#_Toc315364852)

[배경 지식 6](#_Toc315364853)

[구현 목표 8](#_Toc315364854)

[상세 설명 8](#_Toc315364855)

[Merge Buffer 및 기본 구조 8](#_Toc315364856)

[Write Module 9](#_Toc315364857)

[Read Module 11](#_Toc315364858)

[Mapping Table의 관리 11](#_Toc315364859)

[구현 이슈 12](#_Toc315364860)

[개요 12](#_Toc315364861)

[Write Buffering Technique 12](#_Toc315364862)

[Mapping Table Manage 12](#_Toc315364863)

[DMA vs CPU 12](#_Toc315364864)

[성능 평가 13](#_Toc315364865)

[결론 14](#_Toc315364866)

[인턴십 소감 15](#_Toc315364867)

[참고 문헌 16](#_Toc315364868)

[현재 문서에 출처가 없습니다. 16](#_Toc315364869)

그림 목차

[Figure 1. 인턴십 진행 일정 5](file:///C:\Users\ogh\Desktop\src\tutorial2-dev\docs\보고서.docx#_Toc315366622)

[Figure 2. FTL기본 구조 8](#_Toc315366623)

[Figure 3. Dram 사용 구조 9](#_Toc315366624)

[Figure 4. Bank별 Flash Memory 사용 구조 9](#_Toc315366625)

[Figure 5. Merge Buffer로의 Write 10](#_Toc315366626)

표 목차

[Table 1. Merge Buffer의 선언 8](#_Toc315357997)

프로젝트 개요

본 프로젝트는 성균관대학교 정보통신공학부 컴퓨터공학과 학생들의 2011년도 동계 방학 해외 인턴십 목적으로, 미국 캘리포니아 산 호세(San Jose)에 위치한 컴퓨터 하드웨어 업체 오씨젯 테크놀로지(OCZ Technology)에서 진행되었다. 이 해외 인턴십은 정보통신산업진흥원(NIPA; National IT Industry Promotion Agency)의 지원을 받았다.

본 프로젝트에서는 인디링스(Indilinx)에서 공개한 [OpenSSD 플랫폼](http://www.openssd-project.org)을 이용하여 섹터 매핑 FTL(Sector Mapping FTL)을 구현하였고, 기존에 공개된 튜토리얼 FTL(Tutorial FTL)과의 성능을 비교하였다. 또한 튜토리얼 FTL 에는 구현되어 있지 않은 POR(Power OFF Recovery)과 GC(Garbage Collection)를 구현하여 실제 SSD(Solid State Drive) 장치로 사용할 수 있도록 하였다.

프로젝트 진행 일정 및 역할



Figure 1. 인턴십 진행 일정

구현 상세 설명

## 배경 지식

본 보고서에서 사용할 기본적인 용어들에 대한 정의는 다음과 같으며 자세한 내용은 [Jasmine OpenSSD 프로젝트 위키 페이지](http://www.openssd-project.org)를 참고하도록 한다.

FTL - Flash Translation Layer의 준말로, SSD를 사용하는 호스트 PC와 실제 데이터가 저장되는 NAND 플래시 메모리 사이를 연결하는 컨트롤러에서 동작하는 소프트웨어의 핵심 로직을 의미한다. 이 소프트웨어는 주소 번역, POR, GC 등 많은 기능을 제공한다.

NAND – NAND 플래시 메모리(NAND Flash Memory)를 일컫는다. 본 보고서에서 언급하는 플래시 메모리는 모두 이 NAND 플래시 메모리를 의미한다.

블록 - 플래시 메모리에서 데이터를 지우는 최소 단위로서 수십 개의 페이지들로 구성되어 있다.

페이지 - SATA 호스트와 펌웨어 사이의 기본 통신 단위로서 수십 개의 섹터들로 구성되어 있다.

섹터 - I/O 요청(request)의 최소 단위로서 Jasmine 보드에서의 크기는 512Bytes다.

DRAM - Jasmine 보드 상의 DRAM을 의미한다.

SRAM - Jasmine 보드 상의 SRAM을 의미한다.

<W/R , 0 , 0 > - Host PC 로부터의 I/O 요청을 간단히 표현한 것으로 W와 R는 각각 Write, Read 명령을 나타낸다. 가운데의 숫자는 I/O 요청의 시작 섹터 주소 이고 마지막 숫자는 시작 주소로부터 순차적으로 처리해야 할 섹터의 개수를 나타낸다.

SSD는 크게 컨트롤러, NAND 플래시 메모리 두 부분으로 나뉜다. 이 중 컨트롤러가 호스트 PC와 NAND 사이에서 교량 역할을 한다. 그만큼 컨트롤러에서 작동하는 소프트웨어인 FTL의 역할은 SSD에서 굉장히 중요하다. FTL의 기능으로는 주소 변환, GC, POR 등 여러 가지가 있는데, 이번 프로젝트에서는 주소 변환에 초점을 맞추었다.

기존 FTL의 주소 변환 알고리즘에서는 대부분 페이지 단위로 주소를 변환했다. 그러나 실제로 SSD를 사용할 때는 페이지 단위보다 더 작은 크기의 입출력을 자주 하였고, NTFS에서는 페이지 내에 데이터를 쓰고 나머지 부분은 쓸모 없는 데이터로 채우곤 했다. 이로 인해 NTFS에서 SSD를 사용하면 성능 저하가 일어났다. 이를 극복하기 위해, 본 프로젝트에서는 섹터 단위 주소 번역 알고리즘을 개발하고자 한다.

SSD는 주소 변환, POR과 GC 중 하나라도 빠지면 완전히 동작할 수 없다. OpenSSD 프로젝트의 튜토리얼 FTL에는 주소 변환 기능은 있었지만 POR, GC 기능은 없어서 실제 SSD 장치로 사용하는 것이 불가능했다. 이 때문에 기존 핵심 목표보다 더 나아가 POR과 GC 기능을 추가하기로 하였다.

SSD의 전원이 꺼졌을 경우, NAND에 저장된 데이터는 무사하지만 SRAM과 DRAM에 저장된 데이터는 소멸된다. 특히 SSD의 핵심 데이터인 메타데이터(metadata)는 SRAM과 DRAM에 저장되기 때문에 이 데이터를 보호하는 것은 굉장히 중요하다. 이러한 메타데이터들을 수시로 NAND에 저장하여 전원 종료에도 대응할 수 있도록 하는 기능을 POR이라고 한다.

NAND에는 HDD(Hard Disk Drive; 하드 디스크 드라이브)와는 달리 덮어 쓰기를 할 수 없어, 데이터를 페이지 별로 쓰고 블록 별로 지운다. 데이터를 지울 때는 모든 페이지가 유효하지 않은 블록을 지워야 하는데, 실제 상황에서는 그러한 블록이 자주 있는 것이 아니다. 거의 모든 블록이 일부 페이지만 유효하지 않다. 따라서 유효하지 않은 페이지들을 한 블록으로 모아서 지우는 과정이 필요한데, 이 과정을 GC라 한다. SSD가 이 과정을 거치지 않으면 NAND를 전부 활용할 수 없게 되기 때문에 실제 사용 시 FTL에서 GC는 반드시 필요하다.

## 구현 목표

* 섹터 단위 주소 변환 알고리즘을 사용하는 FTL을 구현하여, 작은 단위의 I/O 요청을 최대한 빨리 처리할 수 있도록 한다.
* 기존 튜토리얼 FTL에 없었던 POR 기능을 추가하여, POR에 대응할 수 있도록 한다.
* 이 FTL과 기존의 튜토리얼 FTL의 성능을 비교하고 결과를 분석한다.
* 기존 튜토리얼 FTL에 없었던 GC 기능을 추가하여, NAND 플래시 메모리 관리를 완전하게 한다.

## 상세 설명

### 기본 구조

본 FTL에서는 섹터 단위의 명령들을 페이지크기만큼 모아서 한번에 처리 하기 위해서 Jasmine board의 DRAM 영역에 Merge buffer라는 영역을 따로 두었다. Merge buffer 하나의 크기는 Page의 크기와 같고, 총 Merge buffer의 개수는 활성화된 Bank의 수와 같다.

Table . Merge Buffer의 선언

|  |
| --- |
| #define MERGE\_BUFFER\_ADDR (SCAN\_LIST\_ADDR + SCAN\_LIST\_BYTES)  #define MERGE\_BUFFER\_BYTES (((NUM\_BANKS \* BYTES\_PER\_PAGE + BYTES\_PER\_SECTOR - 1) / BYTES\_PER\_SECTOR )\* BYTES\_PER\_SECTOR) //Align to sector size |

본 FTL의 기본구조는 Figure 2. FTL기본 구조 그림과 같다.

Figure . FTL기본 구조



Figure 3. Dram 사용 구조는 본 FTL의 DRAM 사용 구조를 나타낸다.

Figure . Dram 사용 구조



Figure . Bank별 Flash Memory 사용 구조



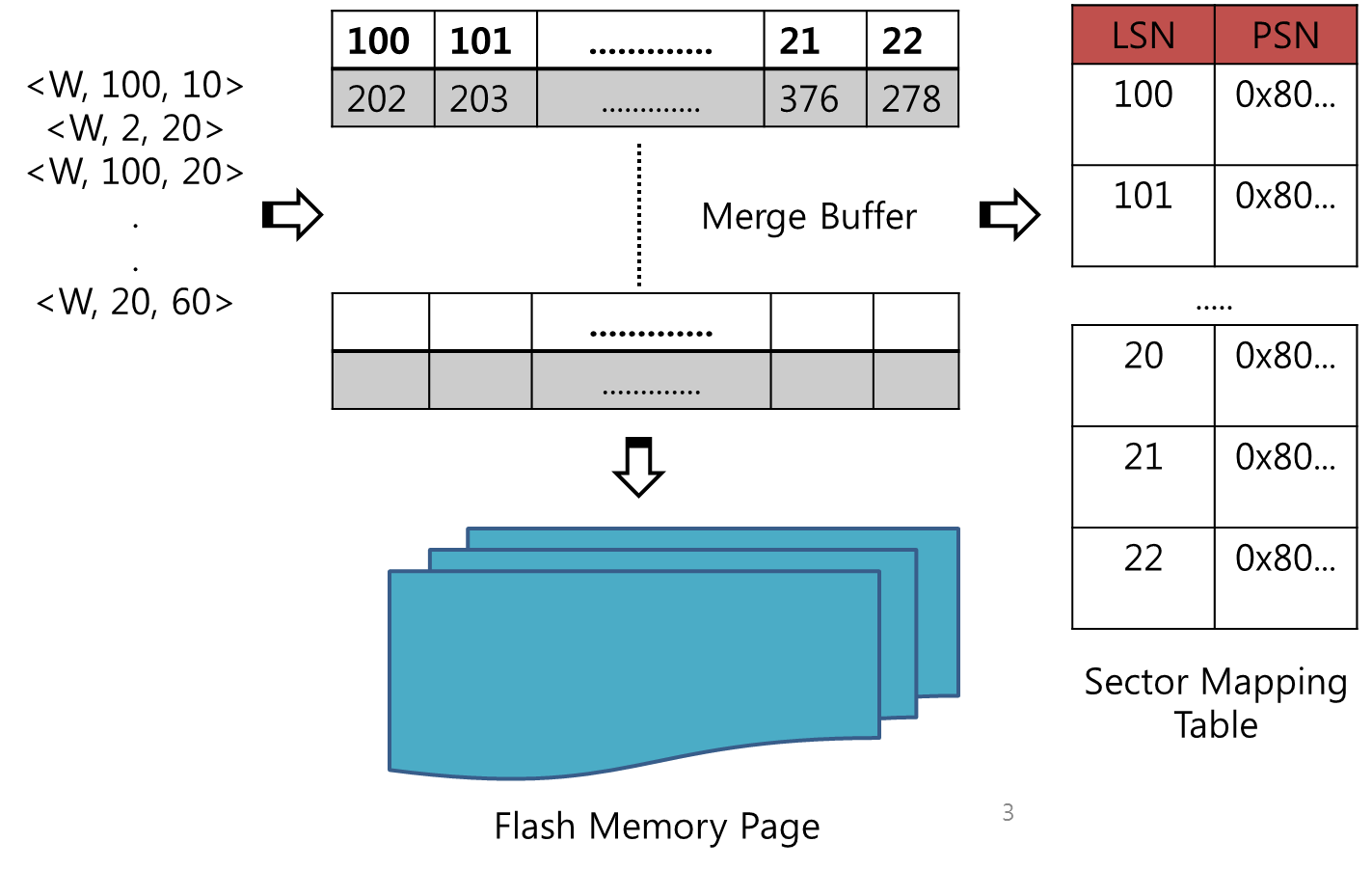
### Mapping Table의 관리

### Write Module

Host PC로부터 SSD로 <W, 100, 30>과 같은 요청이 들어올 경우 요청 처리 순서는 다음과 같다.

1. Lsn 번호 100 번 섹터에 대해서 PSN을 얻어와 해당 섹터가 Merge Buffer에 존재하는지를 판별 한다.
   1. 해당 섹터가 Merge buffer에 존재할 경우 같은 위치에 100번 섹터에 대한 데이터를 복사하고, PSN 업데이트는 진행하지 않는다.
   2. Merge buffer에 존재하지 않을 경우 Merge Buffer의 다음 위치에 100번 섹터에 대한 데이터를 복사하고, PSN을 업데이트 한다.

Figure . Merge Buffer로의 Write



### Read Module

읽기 요청에 대해서는 이렇게 처리한다.

매핑 테이블은 이렇게 관리한다.

구현 이슈

## 개요

본 프로젝트에서 FTL을 구현하는 데 여러 가지 이슈를 고려하여야 했다. 다음은 주요 구현 이슈들이며, 각 이슈와 그에 대한 구현 사항을 자세히 설명한다.

* Write Buffering Technique
* Mapping Table Manage
* DMA vs CPU – 얘는 2GB 에서만 구현 실험됨.

## Write Buffering Technique

## Mapping Table Manage

## DMA vs CPU

성능 평가

결론

인턴십 소감

참고 문헌

# 현재 문서에 출처가 없습니다.