|  |
| --- |
| 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 2](#_Toc316023522)

[프로젝트 진행 일정 및 역할 5](#_Toc316023523)

[구현 상세 설명 6](#_Toc316023524)

[배경 지식 6](#_Toc316023525)

[구현 목표 8](#_Toc316023526)

[상세 설명 8](#_Toc316023527)

[기본 구조 8](#_Toc316023528)

[Mapping Table의 관리 10](#_Toc316023529)

[Write Module 11](#_Toc316023530)

[Read Module 12](#_Toc316023531)

[구현 이슈 13](#_Toc316023532)

[개요 13](#_Toc316023533)

[Write Buffering Technique 13](#_Toc316023534)

[성능 평가 15](#_Toc316023535)

[평가 환경 15](#_Toc316023536)

[평가 결과 16](#_Toc316023537)

[결과 분석 20](#_Toc316023538)

[결론 22](#_Toc316023539)

[향후 계획 22](#_Toc316023540)

[인턴십 소감 23](#_Toc316023541)

[참고 문헌 25](#_Toc316023542)

그림 목차

[Figure 1. 인턴십 진행 일정 6](file:///C:\Users\한규화\Desktop\OpenSSD\tuto\docs\보고서.docx#_Toc316023483)

[Figure 2. FTL기본 구조 10](#_Toc316023484)

[Figure 3. Dram 사용 구조 10](#_Toc316023485)

[Figure 4. Bank별 Flash Memory 사용 구조 11](file:///C:\Users\한규화\Desktop\OpenSSD\tuto\docs\보고서.docx#_Toc316023486)

[Figure 5. Merge Buffer로의 Write 및 Nand로의 Flush 12](#_Toc316023487)

[Figure 6. Read 요청 처리 13](#_Toc316023488)

[Figure 7. Write buffering의 여러가지 방법 15](#_Toc316023489)

[Figure 8. Random Write Test 18](#_Toc316023490)

[Figure 9. Sequential Write/Read Test 18](#_Toc316023491)

[Figure 10. Multi User Test 19](#_Toc316023492)

[Figure 11. 1\_buffer ATTO Test (32GB / 64GB) 19](#_Toc316023493)

[Figure 12. Dynamic ATTO Test(32GB / 64GB) 20](#_Toc316023494)

[Figure 13. Static ATTO Test(32GB / 64GB) 20](#_Toc316023495)

[Figure 14. Multi\_copy ATTO Test(32GB / 64GB) 21](#_Toc316023496)

[Figure 15. Tutorial\_1 ATTO Test(32GB / 64GB) 21](#_Toc316023497)

[Figure 16. Greedy ATTO Test(32GB / 64GB) 21](#_Toc316023498)

표 목차

[Table 1. Merge Buffer의 선언 8](#_Toc316023543)

[Table 2. 32GB 대상 Random Write 16](#_Toc316023544)

[Table 3. 64GB 대상 Random Write 16](#_Toc316023545)

[Table 4. 32GB 대상 Sequential Read/Write 16](#_Toc316023546)

[Table 5. 64GB 대상 Sequential Read/Write 16](#_Toc316023547)

[Table 6. Multi User Random Write Test 17](#_Toc316023548)

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

기간 : 2012. 1. 3 ~ 2012. 2. 3 (약 32일)

장소 : OCZ Technology (6373 San Ignacio Avenue, San Jose, CA)



Figure . 인턴십 진행 일정

담당 멘토 : 정현모 이사님

|  |  |
| --- | --- |
| 이름 | 역할 |
| 오기환 | 대장 |
| 한규화 |  |
| 홍경환 |  |

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. Host PC로부터 쓰기 명령이 들어 올 경우에 바로 Nand로 쓰지 않고 Dram Merge Buffer에 저장을 한 후에 Merge buffer의 한 페이지가 꽉 찼을 경우에 해당 Merge buffer 페이지를 Nand로 flush한다. 각 뱅크의 0번 블록은 펌웨어 정보와 bad 블록 정보를 담고 있고, 1번 블록은 POR을 위한 메타 데이터들을 저장하는 데에 사용된다. 이후에 2번 블록부터 32개의 블록은 섹터 맵 테이블을 저장하기 위한 용도로 사용된다.

Figure . FTL기본 구조



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

Figure . Dram 사용 구조



### Mapping Table의 관리

매핑 테이블은 기본적으로 LSN to PSN 정보를 저장/관리한다. 본 프로젝트에서 구현한 섹터 매핑 테이블의 경우에는 LSN당 4바이트의 매핑 정보를 기준으로 Nand flash memory 1GB용량당 8MB의 Mapping 공간이 필요하다. Jasmine 보드의 DRAM memory 용량이 64MB인 점을 감안하면, 약 8GB이상의 Nand flash memory를 사용할 경우는 모든 매핑 정보를 DRAM에서 관리할 수 없다. 8GB 이상의 Nand flash memory를 관리하는 섹터 매핑 테이블을 구현하기 위해서는 매핑 테이블의 일부만을 DRAM영역에 캐싱하고 나머지 매핑 테이블은 Nand flash memory의 특정 영역에 저장해놓고, 필요한 순간에 읽어오는 방식의 구현이 필요하다. 본 프로젝트에서는 DFTL 논문[1]에서 설명된 방식에 따라 섹터 매핑 테이블을 구현하였다. Nand flash memory의 구조는 Figure 4. Bank별 Flash Memory 사용 구조와 같다.



Figure . Bank별 Flash Memory 사용 구조

### Write Module

Host PC로부터 SSD로 <W, 100, 30>(p. 7)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. Merge Buffer에 있을 경우의 PSN은 최상위 비트를 1로 표기하여 Merge buffer에 있음을 나타낸다.
2. Merge buffer가 가득 찼을 경우에는 해당 Merge buffer 페이지를 Nand로 flush하고 PSN을 업데이트 한다.
3. 1~2의 내용을 30개의 섹터에 대해서 반복 한다.

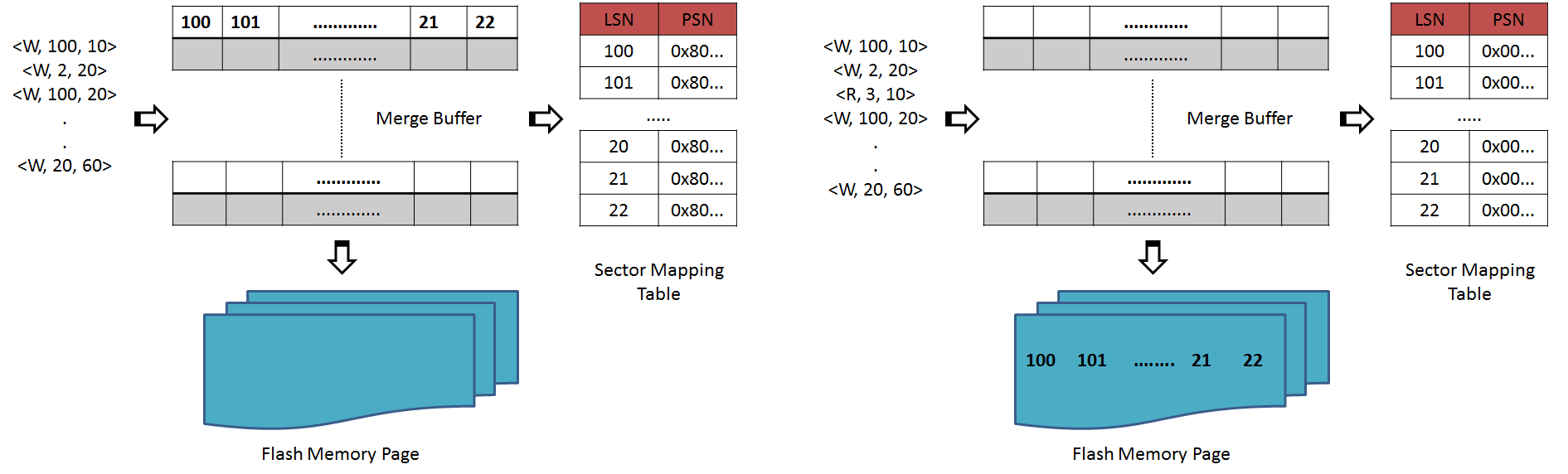
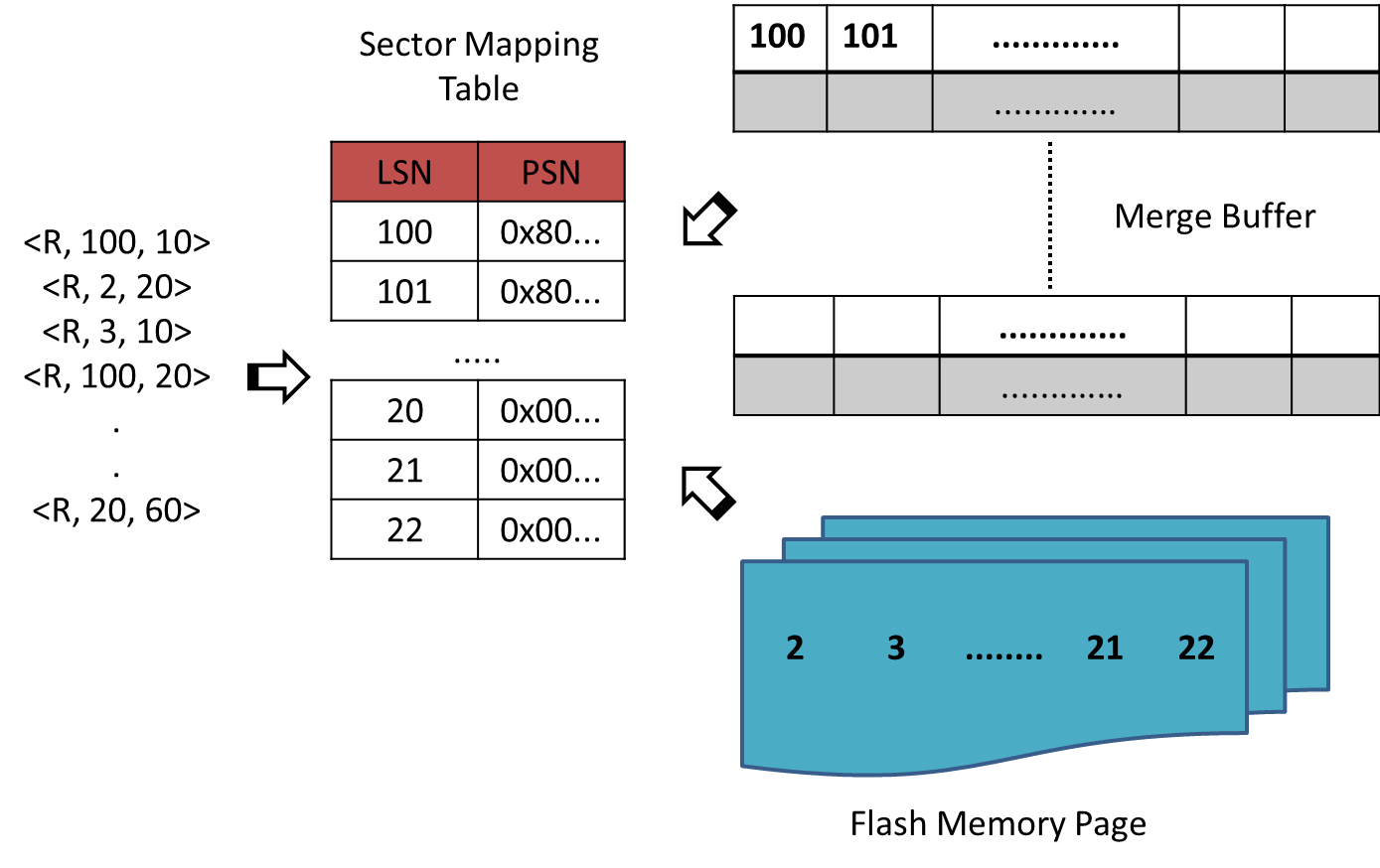


Figure . Merge Buffer로의 Write 및 Nand로의 Flush

### Read 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 Merge buffer의 해당 위치에서 SATA read buffer로 복사한다.
   2. Merge buffer에 존재하지 않을 경우 PSN에 해당하는 Nand섹터에서 해당 섹터 정보를 읽어온다.
2. 1의 내용을 30개의 섹터에 대해서 반복 한다.

Figure . Read 요청 처리

구현 이슈

## 개요

본 프로젝트에서 FTL을 구현하는 데 여러 가지 이슈를 고려하여야 했다. 다음은 주요 구현 이슈이다.

### Write Buffering Technique

Write 요청시에 섹터 단위로 요청을 처리 할 경우, buffer를 사용하지 않고 곧바로 Nand에 섹터 단위 write를 진행하거나 Merge buffer를 이용하는 경우로 나눌 수 있다.

그 중 Merge buffer에 요청된 데이터를 buffering하는 방법은 여러가지가 있을 수 있다.

1. 첫번째 buffer부터 차근차근 꽉 채워 나가는 방법
   1. 제일 첫 buffer의 제일 앞에서부터 채워나가되, 아직 Merge buffer에 존재하는 LSN에 대한 쓰기 요청이 들어올 경우 같은 위치에 데이터를 저장한다. 이 경우에는 쓰기 요청된 LSN에 대한 정보가 Merge buffer에 존재 하는지 안하는지를 판별하기 위해 PSN을 얻어오는 추가작업이 들게 되지만, Merge buffer의 공간활용도를 높일 수 있다.
   2. 제일 첫 buffer의 제일 앞에서부터 append only로 데이터를 저장한다. 한번에 여러 개의 섹터에 대한 쓰기요청이 발생할 경우 Merge buffer 페이지의 크기를 초과하지 않는 크기 내에서는 한번에 복사하여 Merge buffer로 데이터를 옮긴다. 이 경우에는 DRAM 의 메모리 복사 연산 횟수가 줄고, Merge buffer에 해당 LSN의 존재여부를 판별 하지 않아도 되기 때문에 PSN을 얻어오는 작업이 필요없다.
2. 요청이 들어오는 Sector를 순서대로 각 Bank의 Merge buffer로 병렬 분산 시키는 방법
   1. Sector번호에 따라 특정 Bank로 위치를 고정시켜서 같은 LSN에 대해서는 항상 같은 Bank에 저장을 한다. PSN계산이 쉬운 장점이 있지만, 특정 LSN에 대한 요청이 집중 될 경우 특정 Bank로의 Nand 쓰기 요청이 많이 발생하게 되어 병렬성을 살리지 못하는 단점이 있다.
   2. LSN에 따른 Bank 고정이 없이 들어오는 순서 그대로 모든 Bank로 분산하여 저장한다. 이 경우에는 어떤 상황이든지 Bank간의 병렬성을 최대한 살릴 수 있다.

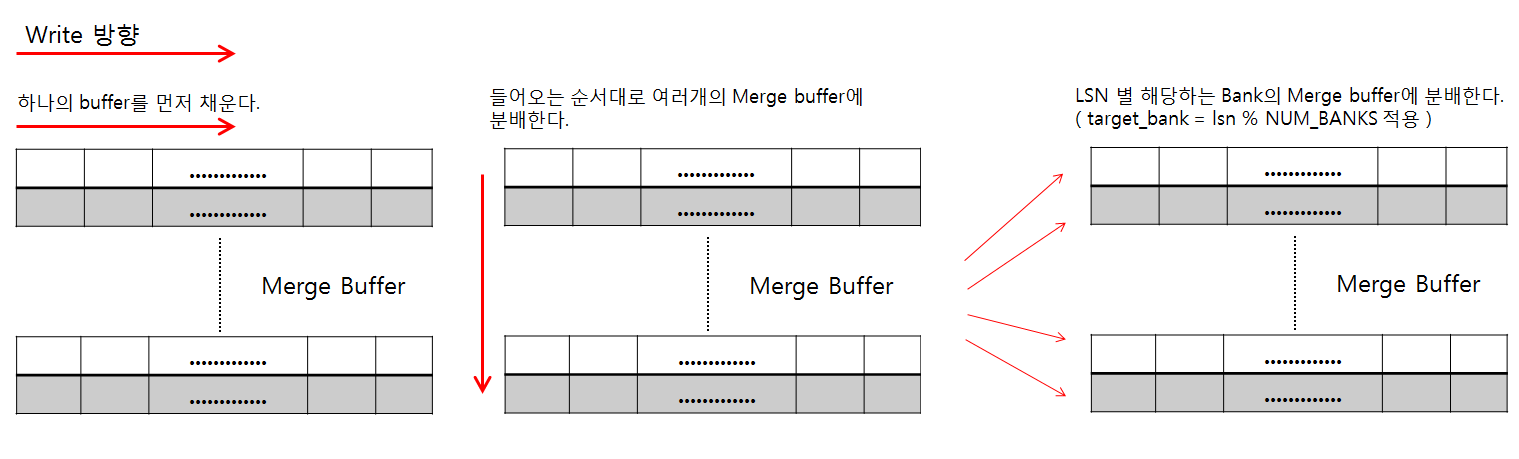


Figure . Write buffering의 여러가지 방법

성능 평가

## 평가 환경

**측정 도구** : IOmeter, ATTO

**측정 방법**

1. IOmeter – Single User Test
   1. Random Write transfer size : 512Bytes, 1KB, 2KB, 4KB
   2. Sequential Write/Read transfer size : 128KB
2. ATTO testing environment
   1. Queue depth : 10
   2. Transfer size : 512Bytes, 1KB, 2KB, 4KB, 8KB, 16KB, 32KB

**측정한 FTL**

1. 1 buffer  
   Merge buffer를 순차적으로 채우는 방법으로 Write Buffering Technique의 1-A의 방법이다.
2. Dynamic   
   Merge buffer에 들어오는 순으로 병렬적으로 분배 하는 방법으로 Write Buffering Technique의 2-A 방법이다.
3. Static  
   Merge buffer에 LBA를 병렬적으로 분배 하되 LBA에 따라 BANK 번호를 고정 시키는 방법으로 Write Buffering Technique의 2-B 방법이다.
4. Multi copy  
   Merge buffer를 순차적으로 채우되 연속된 LBA에 대해서는 한번에 Merge buffer로 옮기는 방법으로 Write Buffering Technique의 1-B 방법이다.
5. Tutorial1, Greedy – 기존 OpenSSD 플랫폼에 구현되어있는 Tutorial FTL과 Greedy FTL이다. 두 FTL은 페이지 매핑 알고리즘을 사용한다.

## 평가 결과

다음 표와 그래프는 IOmeter를 통해 테스트한 결과를 나타낸다. 이 테스트는 모든 Sector Map table이 DRAM에 저장되어 있다고 가정된 상황에서의 결과이다. 각각의 결과는 4번씩 테스트한 결과의 평균값이다. Random 테스트 결과의 단위는 IOPS이고, Sequential 테스트 결과의 단위는 MB/sec이다.

Table . 32GB 대상 Random Write

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| IOPS | 1\_buffer | Dynamic | Static | Multi\_copy | Tutorial\_1 | Greedy |
| 512B | 26930 | 27213 | 27868 | 28370 | 1400 | 1300 |
| 1KB | 22075 | 22773 | 17485 | 24573 | 1340 | 1280 |
| 2KB | 14415 | 13250 | 8858 | 19845 | 1270 | 1250 |
| 4KB | 7377 | 6885 | 4693 | 10470 | 1200 | 1120 |

Table . 64GB 대상 Random Write

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| IOPS | 1\_buffer | Dynamic | Static | Multi\_copy | Tutorial\_1 | Greedy |
| 512B | 28833 | 26680 | 26007 | 27062 | 2466 | 1660 |
| 1KB | 19857 | 23400 | 15343 | 25360 | 2423 | 1610 |
| 2KB | 12604 | 12527 | 8020 | 21287 | 2353 | 1543 |
| 4KB | 6310 | 6607 | 4100 | 11123 | 2228 | 1430 |

Table . 32GB 대상 Sequential Read/Write

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| MB/S | 1\_buffer | Dynamic | Static | Multi\_copy | Tutorial\_1 | Greedy |
| WRITE | 31.5 | 29 | 33 | 62 | 90 | 85 |
| READ | 1.8 | 6.88 | 6.87 | 1.8 | 77 | 77 |

Table . 64GB 대상 Sequential Read/Write

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| MB/S | 1\_buffer | Dynamic | Static | Multi\_copy | Tutorial\_1 | Greedy |
| WRITE | 28 | 28 | 36 | 72 | 170 | 156 |
| READ | 1.8 | 13.8 | 13.4 | 1.8 | 97 | 97 |

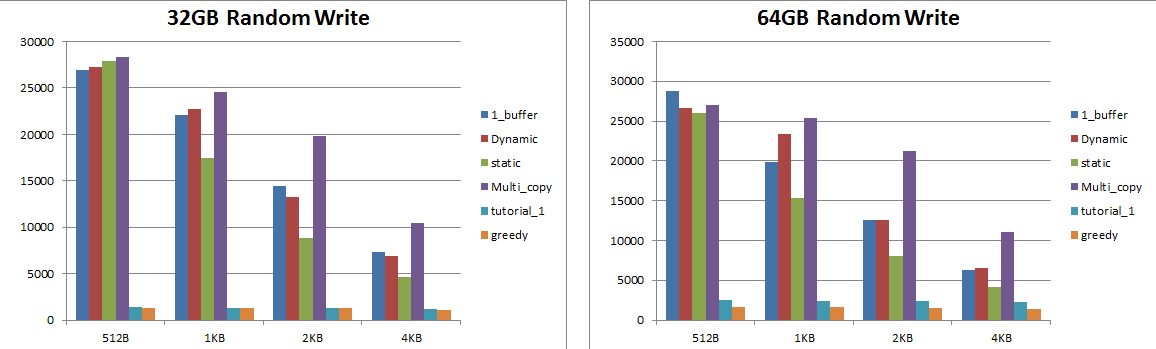


Figure . Random Write Test

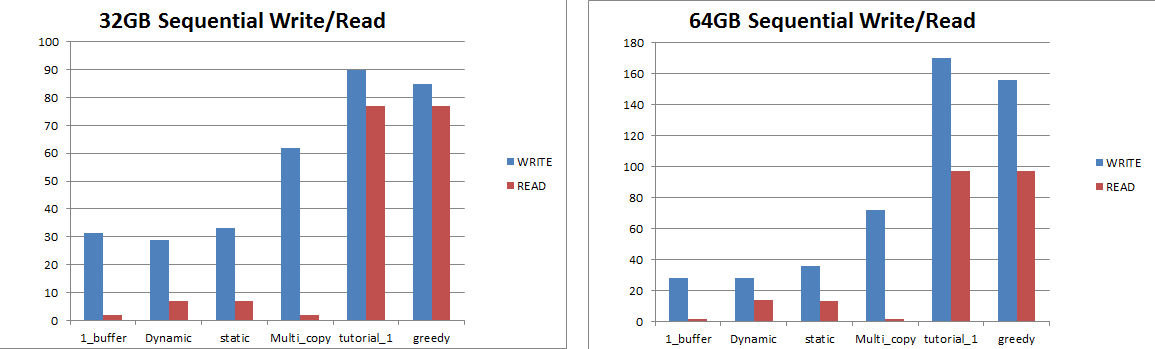


Figure . Sequential Write/Read Test

추가적으로512Bytes size에 대한 Multi User 상황의 Random Write 실험도 진행 하였다. 아래는 그 결과 이다.

Table . Multi User Random Write Test

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| IOPS | 1\_buffer | Dynamic | Static | Multi\_copy |
| 32GB | 47770 | 49200 | 37800 | 56600 |
| 64GB | 43160 | 43390 | 34500 | 47120 |

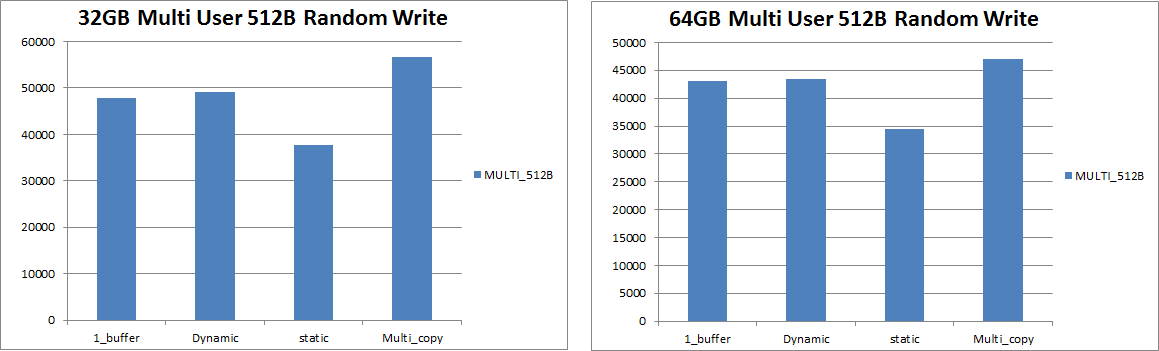


Figure . Multi User Test

아래 결과는 Map Table의 정보가 DRAM에 모두 존재한다고 가정하지 않고, 실제 NAND에 저장되어 있는 데이터를 읽어오는 방법으로 구현된 버전의 성능이다.

Figure . NAND Version Test

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| IOPS | 1\_buffer | Dynamic | static | Multi\_copy |
| Single\_512B | 1293 | 1177 | 1261 | 540 |

Figure . NAND Version Test

아래 결과는ATTO를 이용하여 측정한 결과이다. Sequential Write 명령을 날리기 때문에 Sector Map Table의 영향을 거의 받지 않는다. 이것이 Map Table의 Overhead가 없을 경우의 최대 성능이라고 할 수 있다.

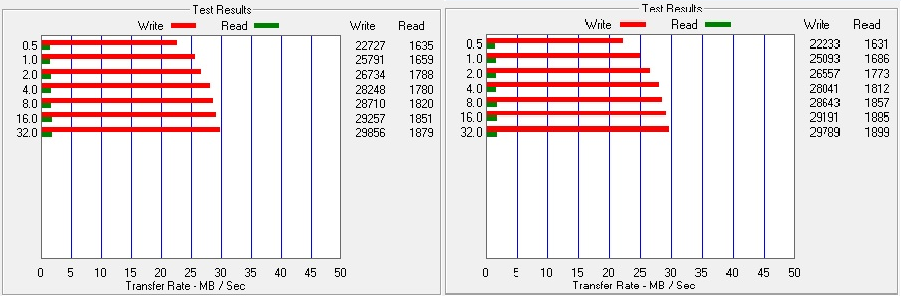
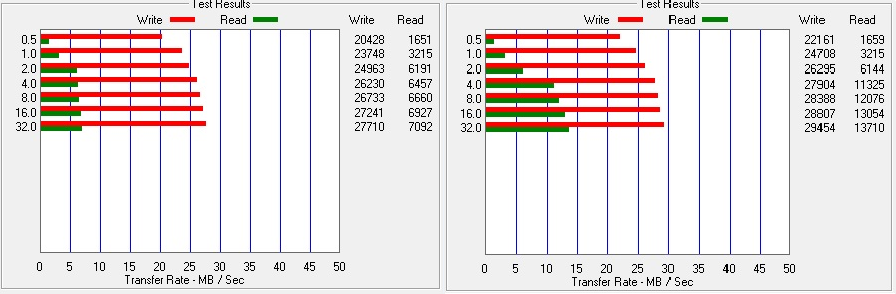
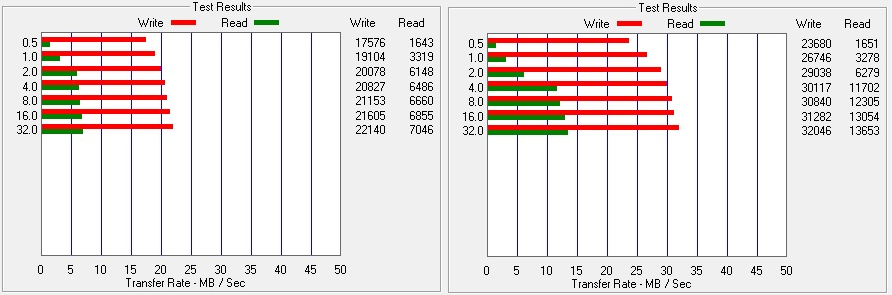


Figure . 1\_buffer ATTO Test (32GB / 64GB)

Figure . Dynamic ATTO Test(32GB / 64GB)

Figure . Static ATTO Test(32GB / 64GB)

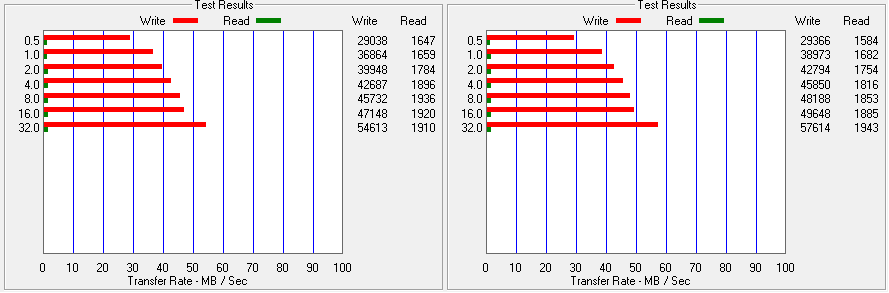


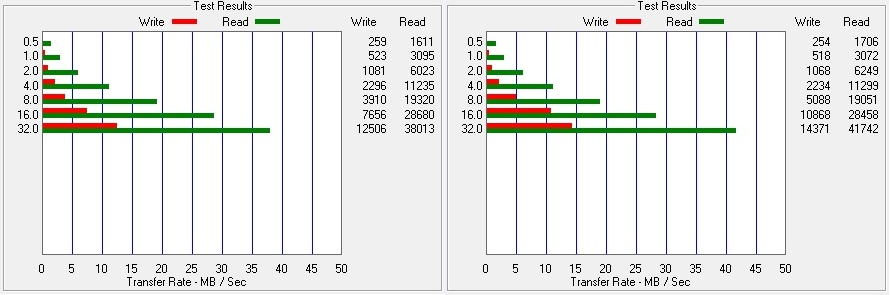
Figure . Multi\_copy ATTO Test(32GB / 64GB)

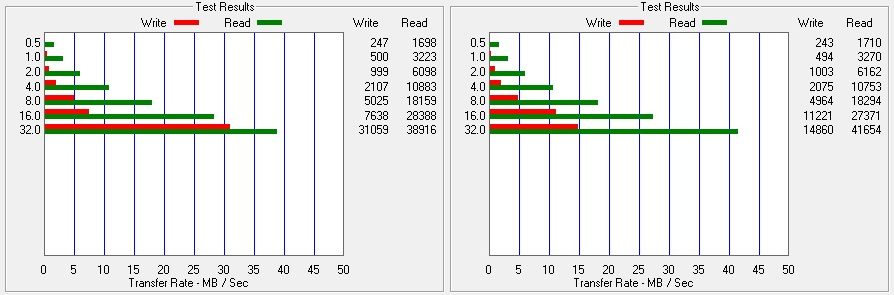
Figure . Tutorial\_1 ATTO Test(32GB / 64GB)

Figure . Greedy ATTO Test(32GB / 64GB)

## 결과 분석

Figure8의 결과를 보면 512B, 1KB, 2KB, 4KB등 작은 단위의 Random Write에 대하여 Sector Level Mapping을 하는 1\_buffer, Dynamic, Static, Multi copy 버전이 Page Level Mapping을 하는 Greedy와 Tutorial\_1에 비하여 IOPS가 높게 나오는 것을 확인 할 수 있다. 이것은 Page Level Mapping과 Sector Level Mapping의 Write 방법의 영향이 크다. Page Level Mapping은 모든 Write를 Page 단위로 하기 때문에 수정이 안된 부분까지 다시 Write하게되어 작은 단위의 Write에 대한 Overhead가 크다. 하지만 Sector Level Mapping은 새로 Write 되는 정보만을 수정하기 때문에 그만큼 NAND의 접근이 적어진다. 하지만 Sector단위로 Map Table을 관리하기 때문에 Map Table의 양이 커지게 되며 이에대한 관리가 중요해진다. Map Table이 모두 DRAM에 존재한다고 가정한다면 512B Random Write에 대하여 Sector Level Mapping이 Page Level Mapping에 비하여 약 100배의 성능을 보이는 것을 알 수 있다.

Sector Level Mapping간에도 Read, Write 방법에 따라 성능의 차이를 보인다. 하나의 Buffer씩 채워나가는 1\_buffer 버전과 Bank 개수만큼의 Buffer를 한 Sector씩 채워가는 Dynamic 버전의 경우는 Write 성능상에서 큰 차이를 보이지 않는다. 하지만 Sequential Read에서는 더 좋은 성능을 보이는 것을 알 수 있다. 이것은 Write 할 때 연속적인 LBA의 Sector들을 각 Bank에 흩뿌려 놓기 때문에 Bank들 간의 병렬성이 발휘되기 때문이다. 이에 반해 Static의 경우 Sector별로 Bank가 고정되어 있기 때문에 Random Write 상황에서 병렬성을 제대로 발휘할 수가 없다. 따라서 Dynamic 버전에 비해 성능이 떨어지는 모습을 확인할 수 있다. 하지만 Read의 경우에는 Dynamic 버전과 같은 성능을 보인다.

Multi copy 버전의 경우 SATA Write Buffer에서 Merge Buffer로 내용을 Copy하는 과정에서 연속적인 Sector를 동시에 Copy하는 방법을 사용하기 때문에 Mem copy의 횟수를 줄일 수 있다. 그래서 Write의 크기가 커지더라도 다른 버전들에 비하여 IOPS의 감소가 적은 것을 확인할 수 있다. 128KB Sequential Write 성능을 보면 다른 Sector Level Mapping 버전들에 비해 많은 성능 향상이 이루어진 것을 확인할 수 있다. 또한 Mapping정보를 읽어오는 get\_psn함수를 생략하기 때문에 DRAM 접근을 SECTORS\_PER\_PAGE번 만큼 줄일 수 있다. 따라서 작은 단위의 Random Write에서도 다른 버전들에 비하여 높은 성능을 보이는 것을 알 수 있다.

Sector Map Table이 모두 Dram에 저장되었다는 가정을 하지 않고, NAND에 저장되어 있는 버전의 경우 Sector Level Mapping의 성능은 Page Level Mapping에 비하여 오히려 감소한 것을 확인하였다. 이것은 Map Table을 NAND에서 DRAM으로 옮기는 Overhead 때문이다. 실제로 Map Table의 Miss Rate가 32GB 기준으로 약 90%정도로 측정이 되었다. 한번에 DRAM으로 옮겨지는 Table의 크기가 크기 때문에 그만큼 성능이 저하되는 것이다. 한번에 옮겨지는 Table의 크기를 Page단위까지 줄여서 실험한 결과가 Table 11이다. Table 11에서 확인할 수 있듯이 이 또한 성능에 많은 저하를 가져온다. NAND Version에서는 Multi copy버전이 다른 버전들에 비하여 성능이 좋지 않게 나오는 것을 확인할 수 있다.

결론

지금까지의 실험 결과를 보면 Map Table의 Overhead를 거의 없는 경우(모든 Map이 DRAM에 존재하는 경우) Sector Level Mapping이 Page Level Mapping에 비하여 작은 단위의 Write에 대한 성능이 매우 뛰어나다는 것을 확인할 수 있다. 하지만 Map Table의 크기가 매우 크기 때문에 DRAM에 모두 저장하기에는 무리가 있다. 이것을 해결하기 위한 가장 간단한 방법으로서 NAND에 Map Table을 저장했지만 큰 성능의 저하가 있었다. Map Table의 관리가 효과적으로 이루어질 수 없다면 Sector Level Mapping은 작은 단위의 Write에서 Page Level Mapping에 비해 크게 좋아지지 못하는 것을 확인할 수 있었다. 또한 Map Table의 관리가 잘 이루어진다면 512B Random Write에서 약 100배의 성능을 낼 수 있다는 것 또한 확인할 수 있었다.

Sector Level Mapping의 성능를 가장 높이기 위해서는 Sector Map Table을 좀 더 효과적으로 관리할 수 있는 방법을 찾아 다시 Jasmine Board에 Test를 해보는 것이 필요할 것이다. 이러한 작업이 이루어진다면 Sector Level Mapping의 장점인 작은 단위의 Write 성능이 효과적으로 발휘될 수 있을 것이다.

## 향후 계획

그동안 완성된 버전들에 대하여 최적화 과정을 진행할 것이다. 변수명들을 사용되는 용도에 맞도록 수정하는 작업과 추가할 필요가 없는 변수들에 관한 부분 또한 수정할 것이다. 그리고 구현 방법에 대한 간단한 주석들을 추가할 것이며, 함수들의 동 작또한 최적화 작업을 진행할 것이다.

이러한 과정들이 끝나면 최종적으로 OpenSSD wiki page에 Sector Level Mapping버전들을 등재하는 작업이 이루어질 것이다.

인턴십 소감

**홍경환**

이번 인턴십 기회는 갑자기 내게 들이닥친 행운이었다. 저번 여름방학에 한국에 있는 인디링스에서 잠시 인턴십을 하였지만, 정말 겨울방학에 미국에 있는 같은 회사로 오게 될 줄은 몰랐다. 그만큼 인턴십을 하기 전에는 모든 것이 막막했고 걱정이 많았다. 미국에 가서 어학연수도 아니고, 일을 한다니! 대학 들어와서 영어시험에는 손도 대지 않았던 내게 영어는 근심덩어리였다. 머릿속에서 햄버거조차도 주문하지 못해 굶고 있을 것 같은 끔찍한 상상이 펼쳐졌다. 그러다가 미국에 막상 오니, 의외로 괜찮았다. 나도 모르게 말을 하고 있었다. 유창하게 토론할 수 있는 정도는 아니지만, 짧게 이야기하는 정도는 괜찮았다. 정말 어려울 때는, 이 회사에 한국어를 할 줄 아는 분들이 계셔서 도움을 많이 받았다.

같은 회사에서 인턴 근무를 했지만, 여름방학 때 느꼈던 한국의 사무실 분위기와 이번에 느낀 미국의 사무실 분위기는 많이 달랐다. 이 소감을 쓰고 있는 지금, 옆칸에는 영어, 한국어 뿐만 아니라 스페인어, 중국어로도 말하고 있다. 여기는 언어도 다양한 만큼 다른 사람들을 많이 이해하고 배려하였다. 그만큼 나도 많이 이해하고 배려하여야 하였다. 이런 이해를 통해 다른 지역에서 자란 사람들이 화합하여 최상의 가치를 만들어내는 모습을 지켜보았다. 한국에서는 볼 수 없었던 ‘세계의 용광로’의 가치다. 놀라웠다.

**한규화**

**이번 인턴은 작년 여름방학에 2개월간 인턴을 하면서 다루었던 OpenSSD Project를 이용하여 진행되었다. 작년에는 Garbage Collection에 초점을 맞추었는데 이번 인턴십은 Sector Level Mapping에 대하여 진행되었다. 이번 인턴을 통하여 OpenSSD Project에 있는 소스코드의 자세한 부분들까지 공부하는 기회가 되었으면 SATA와의 상호 데이터전달, Mapping Table 관리등 여러가지 면에서 공부할 수 있는 좋은 기회가 되었다.**

**한국에서는 미국으로의 인턴십이었기 때문에 두려운 감정도 있었지만, 막상 이 곳에서 인턴십을 하면서는 불편함을 느끼지 않을수 있었으며, 편한 환경에서 인턴을 할 수 있었다. 미국에서의 인턴을 통해 미국의 문화에 대해서도 많은 것을 배울수 있었으며 학업적인 면에서도 많은 것을 얻을 수 있는 기회가 되었다. 매우 보람있고 많은 것을 배울수 있는 인턴십이었다.**

**오기환**

**3학년 1학기에 청강 수업으로 들었던 임베디드 시스템 설계 과목을 통해 처음으로 OpenSSD Project를 알게 되었고, 같은 연구실에 계시던 석사 연구생 선배를 통해서 FTL에 관련한 지식을 공부한 것을 바탕으로 좋은 기회를 얻어 이번 인턴십을 진행하게 되었다. 평소에 전혀 접할 수 없었던 실제 하드웨어 상에서의 임베디드 소프트웨어의 설계는 강의 시간에 배웠던 것보다 더 많은 지식을 필요로 했다. 직접 Jasmine OpenSSD Board에 FTL을 구현하면서 단순히 소프트웨어 적인 측면 많이 아닌 SATA Host와의 관계, Nand Flash Memory와의 관계, 실제 하드웨어의 작동 구조와 각 모듈 별 Bandwidth 등 많은 하드웨어적 측면도 공부를 하게 되고 알게 되어서 정말 값진 지식을 얻게 된 것 같다. 앞으로도 이런 기회가 생긴다면 반드시 다시 한번 참여 하고 싶고, 개인적인 바람으로는 학교에서도 실습 강의를 늘려 실제 임베디드 환경에서의 실습을 진행 할 수 있으면 좋겠다.**

참고 문헌

1. **DFTL: a flash translation layer employing demand-based selective caching of page-leve address mappings** / 저자: Aayush GuptaKim, Bhuvan UrgaonkarYoungjae // ACM SIGPLAN. - ACM.
2. **OpenSSD Project** [온라인] / 저자: Indilinx // OpenSSD Project. - www.openssd-project.org.
3. **The Jasmine OpenSSD Platform: FTL Developer's Guide** [온라인] / 저자: 임상필 // OpenSSD Project. - http://www.openssd-project.org/mediawiki/images/Jasmine\_FTL\_Dev\_Guide\_v.1.2.pdf.
4. **The Jasmine OpenSSD Platform: Technical Reference Manual** [온라인] / 저자: 임상필 // OpenSSD Project.   
   - http://www.openssd-project.org/mediawiki/images/Jasmine\_Tech\_Ref\_Manual\_v.1.4.pdf.