

# 대용량 플래시 메모리 저장 장치를 위한 유닛 레벨 주소 변환 기법

김혁중, 신동근

성균관대학교 정보통신공학부

wangmir@skku.edu, dongkun@skku.edu

## Unit Level Address Mapping Technique for Large Capacity Flash Memory Storage Devices

Hyukjoong Kim, Dongkun Shin

School of Information and Communication Engineering, Sungkyunkwan University

### 요약

낸드 플래시 메모리는 하드 디스크와는 다른 여러 가지 특성 때문에 논리 주소를 물리 주소로 변환해 주는 주소 변환 계층(FTL)이 필요하다. 최근에 고성능의 저장 장치를 제공하기 위해서 페이지 수준의 주소 변환 기법이 많이 사용되고 있는데, 이 기법은 매핑 정보가 너무 커서 메모리에서 매핑 정보를 관리하기에는 힘들다는 문제와 데이터의 접근 지역성을 잘 활용하지 못하는 문제가 있다. 본 논문에서는 스토리지의 주소 공간을 유닛이라는 단위로 분리하여 페이지 수준의 주소 변환을 사용함으로써 매핑 정보를 크기를 줄이고 또한 접근 지역성을 활용하여 가비지 컬렉션 오버헤드를 줄이는 유닛 레벨 주소 변환 기법을 제시한다. 실험 결과, 제시한 기법은 기존의 페이지 매핑 기법보다 랜덤 접근 패턴에서 가비지 컬렉션 오버헤드를 40% 감소시켰으며, 매핑 데이터량도 38% 감소시켰다.

**키워드:** 낸드 플래시 메모리, 플래시 변환 계층, 페이지 매핑, 스토리지

### 1 서론

최근 모바일 디바이스의 사용이 확대됨에 따라 낸드 플래시 메모리의 사용이 급증하고 있다. 특히, 하드디스크를 대체할 차세대 저장장치로 낸드 플래시 메모리 기반의 Solid-State Disk가 각광을 받고 있다.

낸드 플래시 메모리는 덮어쓰기가 불가능하고, 단일 물리 블록 내의 페이지들에 대해서 순차적으로 써야 하며, 각 물리 블록마다 정해진 수명이 있는 등 하드디스크와는 다른 특성을 때문에 기존의 파일 시스템을 통해서 바로 사용할 수가 없으며 플래시 변환 계층(FTL: Flash Translation Layer)라는 시스템 소프트웨어를 필요로 한다. FTL은 호스트 시스템과 낸드 플래시 메모리 사이에서 논리 주소를 물리 주소로 변환하는 기능을 제공하여 기존의 파일 시스템이 플래시 메모리를 일반적인 블록 디바이스로 인식하는 것을 가능하게 해준다.

FTL의 주소 변환 기법 중에 페이지 매핑은 각 페이지 단위로 매핑 정보를 구성하기 때문에 좋은 성능을 제공하지만, 매핑 정보가 매우 커서 SRAM에서 전부 관리하는 것이 쉽지 않았다. 페이지 매핑에서 한 페이지당 필요한 매핑 데이터는 아래와 같이 표현할 수 있다.

$$\text{map}_{\text{page}} = \frac{\lceil \log_2(P_{\text{total}}) \rceil}{8}$$

$P_{\text{total}}$ 는 스토리지 전체의 크기를 페이지 개수로 나타낸 값을 의미한다. 그러므로, 스토리지의 크기가 커질수록 매핑 정보를 관리할 메모리의 크기가 같이 커져야 된다는 단점이 있다. 또한, 가비지 컬렉션(Garbage Collection) 수행 시 반응 시간이 급격히 저하되는 단점이 있었다. 페이지 매핑의 단점을 보완하기 위해서 하이브리드 매핑 기법이 제시되었는데, 이 기법에서는 저

장장치의 일부 공간을 로그버퍼로 할당하여 로그 버퍼에 대해서만 페이지 매핑을 사용하고, 나머지 영역에서는 블록 매핑을 사용하여 매핑 정보의 크기를 줄인 기법이다. 하지만, 로그버퍼에 빙 공간이 없을 때, 블록 병합이라는 과정이 필요하고 이 때문에 전체적으로 페이지 매핑에 비해서 성능이 떨어진다.

페이지 매핑의 문제점을 해결하기 위한 다른 기법으로 제시된 DFTL[1]은 기본적으로 페이지 매핑 기법을 사용하면서 전체 매핑 정보는 플래시 메모리에 저장하고 필요에 따라 사용할 일부 매핑 정보를 캐시에서 관리하여 매핑 정보를 저장할 SRAM 공간을 줄였다. 하지만, 랜덤 접근 시에는 캐시에서 해당 매핑 정보를 찾을 가능성이 줄어들기 때문에 매핑 정보를 읽기 위해서 플래시 메모리를 자주 접근해야 하는 문제가 있으며, 캐시의 크기에 비해 전체 매핑 정보가 커지면 캐시 미스(cache miss)가 자주 발생하게 된다. 또한, 페이지 매핑의 또 다른 단점인 고비용의 가비지 컬렉션 문제는 개선은 하지 못하였다.

본 논문에서는 DFTL의 문제점을 개선하기 위해서 유닛 레벨 매핑 기법을 제안한다. 본 기법에서는 전체 저장 공간을 유닛(unit)이라고 하는 크기로 분리하고, 유닛 내에서의 페이지 매핑을 함으로서 DFTL보다 매핑 정보의 크기를 줄여서 매핑 정보 관리 부하를 감소시켰으며, 유닛 내의 지역성(locality)을 이용하여 가비지 컬렉션 부하도 크게 감소시켰다.

### 2 관련 연구

제안하는 유닛 레벨 매핑 기법과 유사하게 스토리지 공간을 일정 크기로 분할하여 관리하는 기법들이 제시된 것이 있다. Superblock[2] 기법에서는 연속된 논리 블록을 슈퍼 블록이라는 단위로 묶고, 슈퍼블록 내에서

는 페이지 매팅을 사용한다. 하나의 논리 슈퍼블록에는 여러 개의 물리블록이 할당된다. Superblock 기법에서는 매팅 정보를 각 페이지의 스페어 영역에 기록하며, 매팅 정보를 3개의 계층을 통해 관리하기 때문에 매팅 정보를 읽기 위한 오버헤드가 크다. 특히, 스페어 영역의 제한으로 인해서 슈퍼블록의 크기가 한정될 수 밖에 없고, 심지어 ECC(Error Correction Code)의 길이가 상대적으로 긴 MLC 낸드 플래시에서는 아예 구현이 불가능할 수도 있다. 반면에, 제안하는 유닛 레벨 매팅 기법은 두 단계의 매팅 계층을 통해서 매팅 정보를 필요에 따라 캐시에 올리는 방식으로 관리하여 Superblock 기법의 문제점을 해결하였다.

Group mapping[3]이나 SAST[4] 기법도 논리 주소 영역을 동일한 크기의 그룹으로 나누어서 관리하는데, 이들 기법은 하이브리드 매팅을 사용하고 있어 하이브리드 매팅 구조의 한계를 극복하지 못하였다.

### 3 유닛 레벨 매팅 기법

#### 3.1 전체구조

유닛 레벨 매팅에서는 연속된 여러 논리 블록들을 유닛이란 단위로 관리하며, 각 유닛마다 논리 블록 개수보다 많은 물리 블록들을 할당 받을 수 있으며 그 최대값은 고정되어 있다. 유닛에 할당된 각 물리 블록은 유닛 블록 인덱스(UBI: Unit Block Index)를 통해 관리된다. 두 종류의 매팅 정보가 사용되는데, 하나는 UB(Unit Block) map으로서 각 논리 페이지가 기록된 유닛 내의 물리 블록의 UBI와 블록 내 오프셋을 가지고 있으며, 다른 하나는 LU(Logical Unit) map으로서 각 유닛에 할당된 물리 블록의 정보를 저장한다.

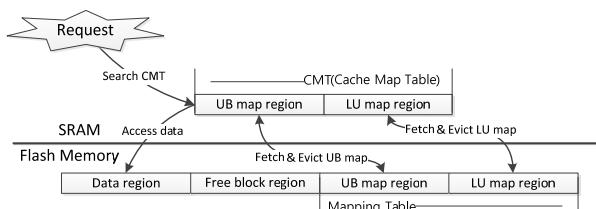


그림 1 유닛 레벨 매팅에서의 CMT 관리

그림 1은 본 논문에서 제시하는 FTL의 구조를 보여주고 있는데, 매팅 정보는 모두 플래시 메모리의 맵 영역에 저장되며 필요에 따라 SRAM의 CMT(Cache Map Table) 영역으로 읽혀진다. CMT는 두 영역으로 분할되며, 개개의 영역은 LRU(Least Recently Used) 정책에 따라 관리된다.

UB map과 LU map의 전체 크기인 UB map<sub>total</sub>과 LU map<sub>total</sub>은 각각 아래와 같이 표현될 수 있다.

$$\begin{aligned} \text{UB map}_{\text{total}} &= \text{UB entry} \times P_{\text{total}} \\ \text{LU map}_{\text{total}} &= \text{LU entry} \times PB_{\text{unit\_max}} \times U_{\text{total}} \\ \text{UB entry} &= \left\lceil \frac{\log_2(P_{\text{block}}) + \log_2(PB_{\text{unit\_max}})}{8} \right\rceil \\ \text{LU entry} &= \left\lceil \frac{\log_2 B_{\text{total}}}{8} \right\rceil, \quad PB_{\text{unit\_max}} = B_{\text{unit}} \times \alpha \end{aligned}$$

UB entry와 LU entry는 각각 UB map의 페이지당 매팅

정보의 크기와 LU map의 블록당 매팅 정보의 크기를 의미한다.  $PB_{\text{unit\_max}}$ 은 하나의 유닛에 할당 가능한 최대 물리 블록 개수를 나타내는데, 유닛의 논리 블록 개수인  $B_{\text{unit}}$ 의 고정배수로 설정한다.  $P_{\text{block}}$ 은 물리 블록 내의 페이지 수를 나타내며,  $B_{\text{total}}$ 은 스토리지 전체의 물리블록 개수를,  $U_{\text{total}}$ 은 스토리지 전체의 유닛 개수를 의미한다.

#### 3.2 논리 페이지 접근 방법

유닛 레벨 매팅 FTL은 논리 주소에 대한 읽기/쓰기 요청이 있으면, 접근이 요구된 논리 페이지 주소(LPN)에 대해서 UB map을 통해서 그 페이지의 UBI와 오프셋 값을 알아내고, LPN을 유닛의 논리적 페이지 개수 ( $B_{\text{unit}} \times P_{\text{block}}$ )로 나누어 그 페이지가 속한 논리 유닛 번호를 알아낸다. 그리고, LU map에서 논리 유닛 번호와 UBI를 이용해 해당 논리 페이지가 기록된 물리 블록 주소(PBN)를 알아낸 뒤에 오프셋을 이용해서 물리 페이지 주소(PPN)를 알아낸다. UB map과 LU map은 우선 CMT를 참조하며, CMT에 없을 시에 플래시 메모리의 map 영역에서 읽어온다. 그럼 2는 LPN이 42인 논리 페이지의 데이터를 접근하는 예시를 보여주고 있다.

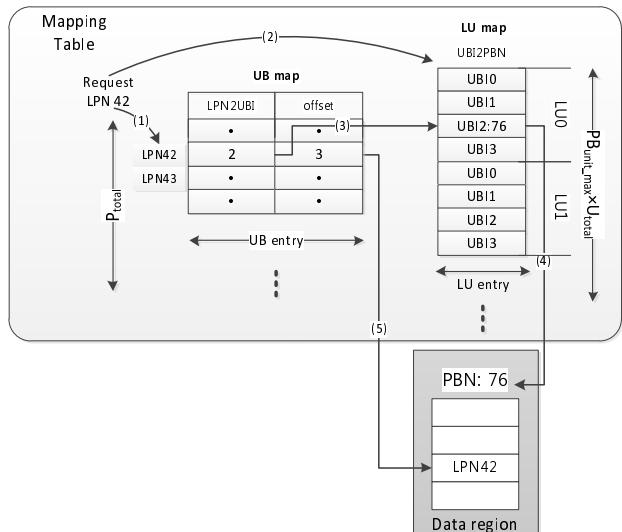


그림 2 유닛 레벨 매팅에서의 데이터 접근 구조

#### 3.3 가비지 컬렉션 연산

본 논문에서는 유닛 단위로 스토리지를 관리하는 FTL 구조에 부합하는 가비지 컬렉션을 수행하기 위해 두 종류의 가비지 컬렉션 기법을 제안한다. 첫째는 유닛 내 가비지 컬렉션(IntraGC)으로 유닛에  $PB_{\text{unit\_max}}$  개의 물리 블록이 할당되어 추가의 물리 블록을 할당할 수 없을 때 사용되며, 두 번째는 유닛 간 가비지 컬렉션(InterGC)으로 유닛에  $PB_{\text{unit\_max}}$  개 이하의 물리 블록이 할당되어 자유블록을 추가로 할당할 수 있지만 전체 스토리지에 자유블록이 여유가 없을 때 다른 유닛에 이미 할당된 블록을 가져오기 위해서 사용한다.

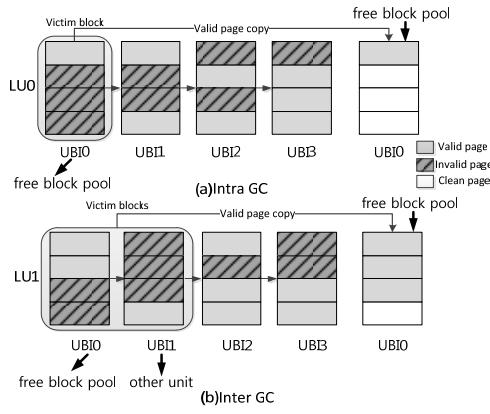


그림 3 유닛 레벨 매핑 FTL의 가비지 컬렉션

그림 3(a)는 유닛 내 가비지 컬렉션의 상황을 보여주고 있는데, 유닛 내 물리 블록 중 가장 무효 페이지(invalid page)가 많은 블록을 희생블록(victim block)으로 선정하고, 그 블록에 포함된 유효 페이지를 free block pool에서 가져온 자유 블록에 복사한 후, 그 블록을 해당 유닛에 귀속시켜서 쓰게 된다. 그림 3(b)는 유닛 간 가비지 컬렉션으로 자유 블록에 여유가 없어서 다른 유닛에서 가비지 컬렉션을 해서 자유 블록을 만들어서 가져오는 경우이다. 유닛 내 가비지 컬렉션이 유닛의 크기 제한에 따라서 이루어지는 가비지 컬렉션이라면, 유닛 간 가비지 컬렉션은 물리적 영역 고갈에 따라서 수행되는 가비지 컬렉션이다.

## 4 실험 및 평가

### 4.1 실험 환경

본 논문의 실험에서는 소프트웨어 시뮬레이터를 사용하였으며 대상으로 한 플래시 메모리 스토리지와 입력 트레이스는 다음과 같다.

- 4GB SLC NAND flash memory
  - 읽기 속도: 25us/page
  - 쓰기 속도: 200us/page
  - 지우기 속도: 1500us/block
- 트레이스: FAT32, pcNTFS, Iozone(random, sequential, all)

FAT32와 pcNTFS는 각각 FAT32, NTFS 파일시스템에서 여러 응용 프로그램을 실행하면서 수집한 트레이스이다.

### 4.2 매핑 정보량의 비교

그림 4는 DFTL과 유닛 레벨 매핑(ULM) 기법의 전체 매핑 정보의 크기를 전체 스토리지 크기를 변화시켜 가면서 비교한 그래프이다. DFTL의 경우 64GB를 기점으로 한 페이지에 대한 매핑 정보가 3 byte에서 4 byte로 증가하게 전체 매핑 정보의 크기가 64GB를 경계로 증가한다. 유닛 레벨 매핑의 경우 유닛의 크기인  $B_{unit}$ 이 512보다 작으면 DFTL의 매핑 정보 대비 보다 38% 감소한 크기의 매핑 정보를 요구한다. 유닛의 크기인  $B_{unit}$ 이 512보다 크면, UB map의 페이지당 매핑 정보 크기가 2 byte에서 3 byte로 증가하게 되는데, 그에 따라서 전체 매핑 정보의 크기가 64GB 이하의 용량에

서는 DFTL의 경우보다 크지만, 64GB 이상의 용량에서는 DFTL보다 작은 매핑 정보를 요구한다. 이 결과를 통해서, 유닛 레벨 매핑 기법이 DFTL보다 전체적으로 적은 매핑 정보를 요구하기 때문에 매핑 정보를 캐싱하는 메모리를 효율적으로 사용 할 수 있음을 알 수 있다.

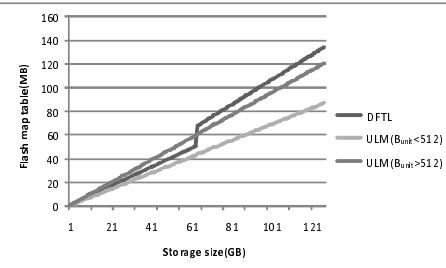


그림 4 ULM과 DFTL 기법의 매핑정보 크기 비교

### 4.3 성능 비교

DFTL과 성능을 비교하기 위해서 iozone 벤치마크를 사용하여 랜덤접근(iozone\_rand), 순차접근(iozone\_seq), 랜덤 및 순차접근(iozone\_all)의 3가지 패턴을 사용하여 성능을 관찰하였다. 랜덤접근과 순차접근 트레이스는 쓰기 요청만을 가지고 있으며, 랜덤 및 순차접근(iozone\_all) 트레이스는 읽기와 쓰기를 모두 가지고 있다. 실험은 가비지 컬렉션 오버헤드를 쉽게 관찰하기 위해서 스토리지의 전체 용량의 반을 데이터로 채운 후, 트레이스 별로 서로 다른 패턴으로 데이터에 대한 업데이트를 하면서 성능을 관찰하였다. 실험에서 사용한 유닛의 크기는 128이며, CMT의 크기는 64KB이다.

그림 5는 ULM기법의 성능을 DFTL의 성능에 대한 상대값으로 보여주고 있다. 랜덤 접근의 경우에, ULM이 DFTL보다 40% 적은 가비지 컬렉션 비용을 발생시키는 것을 알 수 있는데, 이것은 유닛 단위로 구분하여 페이지를 관리함으로서 hot/cold 데이터를 잘 분리하게 되어 가비지 컬렉션 시에 폐이지 복사횟수를 줄였기 때문이다. 순차접근 패턴에서는 두 기법의 가비지 컬렉션 비용이 큰 차이가 없었다. Iozone\_all 실험에서는 읽기 요청의 비중이 커 상대적으로 GC이 적게 발생하여 매핑 정보 관리 오버헤드가 크게 나타남을 알 수 있다.

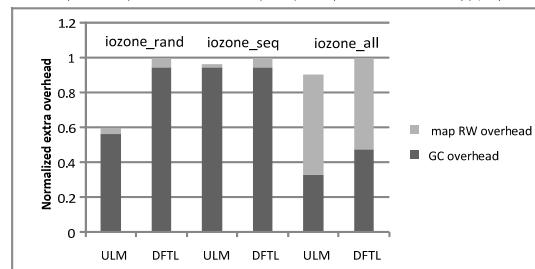


그림 5 ULM과 DFTL의 성능 비교

그림 6은 iozone 실험에서 CMT에서의 hit ratio를 보여 주고 있는데, 특히 랜덤 패턴에서 ULM이 DFTL보다 우수한 성능을 보임을 알 수 있다. 이것은 ULM에서 페이지당 매핑 데이터가 DFTL에서보다 작기 때문이다.

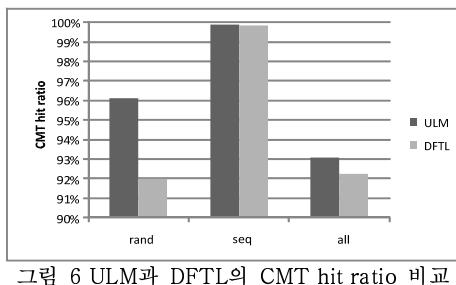


그림 6 ULM과 DFTL의 CMT hit ratio 비교

그림 7과 그림 8은 FAT32와 pcNTFS 트레이스에 대해 유닛의 크기를 변화시켜면서 I/O 처리의 오버헤드를 관찰한 결과이다. 유닛의 크기가 커짐에 따라서 LU map의 크기가 함께 증가하기 때문에 매핑 정보를 읽는 데 소모되는 부하는 증가했다. 하지만, 가비지 컬렉션에 의해 생기는 부하는 감소하는 것을 볼 수 있는데, 그 이유는 InterGC를 적게 유발하기 때문이다. 워크로드의 특성에 따라 차이가 있지만, 보통 하나의 유닛이 128개, 또는 256개의 블록을 가지게 될 때 최적의 성능을 낸다.

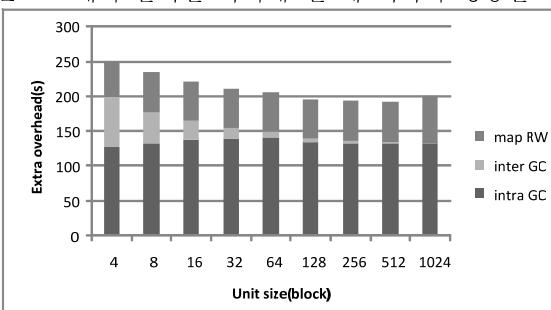


그림 7 유닛 크기 변경에 따른 오버헤드 변화(FAT32)

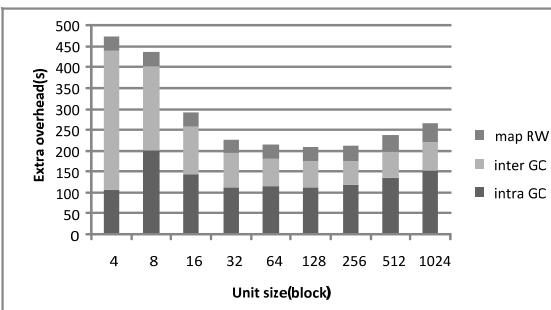


그림 8 유닛 크기 변경에 따른 오버헤드 변화(pcNTFS)

#### 4.5 CMT 분할에 따른 실험

CMT는 UB map과 LU map의 두 영역으로 분할되는데, 두 영역의 적절한 비율을 실험으로 관찰해 보았다. 그림 9는 UB map의 비율을 기준으로 매핑 정보를 관리하는 오버헤드를 측정한 그래프이다. 최적의 설정은 전체 CMT의 크기에 따라서 달라지는데, 예를 들어 32KB CMT에서는 70%의 영역을 UB map을 위해서 사용하면 된다. CMT 크기가 클수록 UB map 영역을 크게 잡는 것이 유리함을 알 수 있다.

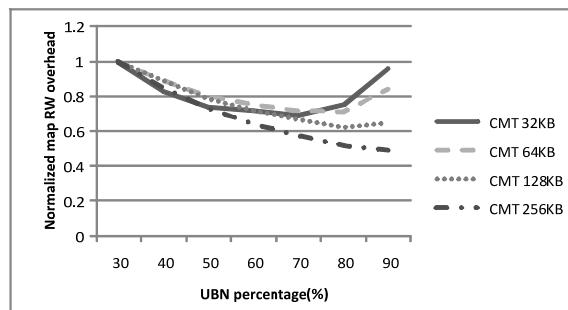


그림 9 UB map의 분할에 따른 매핑 관리 부하(FAT32)

### 5 결론

플래시 메모리가 대용량화됨에 따라 매핑 정보의 관리가 큰 문제로 부각되고 있다. 본 논문에서 제안한 유닛 레벨 매핑은 전체 스토리지를 유닛이란 단위로 나누어 페이지 매핑 기법을 사용함으로써 매핑 테이터의 크기를 감소시키며, 필요에 따라 일부의 매핑 정보만 캐시에 읽어들임으로서 실행시간에 필요한 메모리의 용량을 크게 줄이는 기법이다. 또한, 유닛 단위의 관리를 통해서 데이터 접근의 지역성을 활용하여 가비지 컬렉션의 오버헤드도 크게 줄일 수 있다. 향후에는 테이터의 접근 패턴에 따라서 유닛 별로 매핑 레벨을 다르게 하는 다른 매핑 기법에 대해서 연구하고자 한다.

#### 참고문헌

- [1] A. Gupta, Y. Kim, and B. Urgaonkar, "DFTL: A Flash Translation Layer Employing Demand-based Selective Caching of Page-level Address Mappings", ASPLOS'09, March 2009.
- [2] J. Kang, H. Jo, J. Kim, J. Lee, "A Superblock-based Flash Translation Layer for NAND Flash Memory", EMSOFT'06, October 2006.
- [3] H. Kwon, T. Chung, "An Efficient Mapping Technique for Flash Memory Using Grouped Blocks", ICEIE'10, August 2010.
- [4] C. Park, W. Cheon, J. Kang, K. Roh, W. Cho, and J.-S. Kim, "A reconfigurable FTL architecture for NAND flash-based applications", ACM Transactions on Embedded Computing Systems, Vol. 7, No. 4, Article 38, 2008.