NAND 플래시 메모리의 프로그램 속도 개선을 위한 데이터 코드 변환 기법의 성능 평가

정관일, 유수원, 현철승, 이동희
서울시립대학교 컴퓨터과학부
gijeong@uos.ac.kr, ysw1508@uos.ac.kr, cshyun@uos.ac.kr, dhl_express@uos.ac.kr

Evaluation of Data Encoding Method Enhancing Program Performance of NAND Flash Memory

Gwanil Jeong, Soowon You, Choulseung Hyun, Donghee Lee
Dept. of Computer Science, University of Seoul

요 약

다양한 응용에서 저장 매체로 사용되는 NAND 플래시 메모리는 저비용과 대용량을 위해 셀 당 비트 수 증가, 제조 공정의 미세화, 그리고 적층 기술 등 다양한 기술을 사용한다. 그렇지만 이러한 기술들은 플래시 메모리 셀의 안정성과 성능에 악영향을 준다. 특히 QLC 3D 플래시 메모리인 경우, 셀 상태가 많고 상태 간 임계 전압 간격이 좁기 때문에 프로그램과 읽기에 필요한 시간이 길다. 본 논문에서는 프로그램 수행 시간을 줄이고 셀 안정성에 긍정적인 영향을 줄 수 있도록 데이터 코드를 변환하는 비균일 스크램블 기법을 소개하고, 실제 시스템 데이터를 이용하여 스크램블 기법의 성능을 평가한다. 시뮬레이션을 통해 얻은 결과에 따르면 데이터 코드를 변환하여 저장하는 스크램블 기법은 최대 204%의 프로그램 성능 개선 효과를 보인다.

1. 서론

NAND 플래시 메모리는 모바일 기기부터 서버 컴퓨터에 이르기까지 다양한 응용에서 저장 매체로 사용된다. 비트당 제조 비용을 낮추고 대용량 메모리를 제조하기 위해 공정을 미세화하고 하나의 메모리 셀에 여러 비트의 정보를 저장하도록 진화하여, 현재 셀 당 2, 3, 그리고 4 비트를 저장하는 MLC, TLC 그리고 QLC 플래시 메모리가 출현하였다. 뿐만 아니라 적층이 가능하도록 Planar 기술을 도입하여 3D 플래시 메모리를 생산하고 있다[1].

QLC 3D NAND 플래시 메모리인 경우, 기존의 플래시 메모리에 비해 셀의 안정성이 더 낮다. 이는 QLC 플래시 메모리가 하나의 셀에 4 비트 정보를 저장하기 위해 셀 상태를 16 개로 구분해야 하기 때문이며, 이를 위해 MLC 나 TLC 에 비해 각 상태 간 임계(threshold) 전압의 간격이 더 좁다. 그리고 원하는 셀 상태로 프로그램하기 위해서 많은 단계의 ISPP(Incremental Step Pulse Programming)를 거쳐야 한다. 그 결과 ISPP 프로그래밍에 더 오랜 시간이 걸릴 뿐만 아니라 인접 셀에 주는 영향도 커져 리텐션 에러도 증가한다[2]. 아울러 더 많은 셀 상태를 구분해야 하기 때문에 읽기 성능도 떨어진다.

본 논문에서는 NAND 플래시 메모리 셀의 손상을 줄이고 프로그램 성능을 개선하기 위한 데이터 코드 변환 기법의 성능평가 결과를 소개한다. 다양한 목적으로 데이터를 다른 코드로 변환하여 플래시 메모리 셀에 저장하는 기법들이 소개되었다. 플래시 메모리의 셀 사이의 전압 간격을 일정하게 유지하기 위해 데이터 스크램블 기법[3, 4]이 제안되었고, 낮은 임계 전압 상태를 갖는 셀들을 많이 만들기 위해 데이터 코드를 변환하는 ACS[5], NRC[6], ELSE[7], CeSR[8] 등과 같은 기법들이 제안되었다. 최근에는 읽기 성능을 향상시키기 위한 IDA 기법[9]도 제안되었다.

본 논문의 선행 논문[10]에서 제안한 기법은 기존의 기법과 달리 추가 비트를 사용하지 않고, 데이터 코드 변환과 함께 스크램블도 수행한다. 특히, 스크램블을 수행할 때 사용되는 확률 밀도 함수에 따라 변환된 데이터 코드의 분포를 원하는 형태로 만들 수 있다. 본 논문에서는 제안된 코드 변환 기법을 사용하여, 가능하면 셀이 낮은 전압 상태를 가지도록 프로그래밍하고, 선행논문에서 제시되지 않았던 성능 비교 결과를 제시한다. 실험 결과를 보면 순수 랜덤 스크램블 기법에 비해 셀 상태를 조절하는 스크램블 기법의 성능이 최대 204% 정도 좋음을 알 수 있다.

2. 데이터 코드 변환 기법

본 장에서는 선행 논문에서 소개되었던 데이터 코드 변환 기법인 VDSC(Variable Density Scrambling Scheme)에 대해 설명한다. VDSC 는 여러 방법으로 구현될 수 있으며, 아래에 여러 방법들 중 회전판 기법[10]에 대해 간략히 소개한다.

생성되는 데이터 코드의 분포가 균일한 기존 랜덤 스크램블 기법과 다르게 VDSC 기법은 스크램블을 수행하면서 변환된 코드의 분포를 균일하지 않게 만들 수 있다. 변환된 코드가 비균일 분포를 가지려면 변환 이전의 코드가 비균일 분포를 가지고 있어야 한다. VDSC 실험 결과에 따르면 대부분의 시스템에 저장된 데이터들은 비균일 분포를 가진다[10]. VDSC 기법은 비균일 속성을 반영하여 코드 변환 테이블을 구성할 뿐만 아니라, 비균일 속성을 유지하도록 비균일 랜덤 함수로 스크램블을 수행한다.

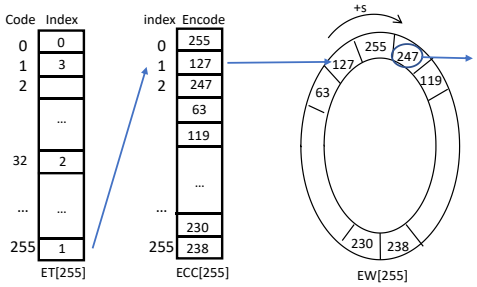
비균일 속성을 반영하기 위하여 VDSC 기법은 데이터 코드를 출현빈도 순으로 정렬하고, 이를 목표로 하는 플래시 메모리 셀 상태와 대응시켜 코드 변환 테이블을 구성한다. QLC 플래시 메모리라면 S0~S15 까지 16 가지 상태에 특정 값의 그레이 코드가 할당된다. 목표 셀 상태가 S0 상태라면 출현 빈도가 가장 높은 데이터가 셀 상태 S0 를 가진 코드로 변환되도록 변환 테이블을 구성한다. 결국 출현 빈도가 높은 데이터 코드일수록 S0 상태와 가까운 코드로 변환된다.

(그림 1)과 (그림 2)는 8 비트 데이터 코드를 변환하는 인코딩과 디코딩 테이블이다. (그림 1)에서 ET(Encoding Table)는 0~255 까지 8 비트 데이터 코드의 출현 빈도 순서인 인덱스를 가지는 테이블이며, 인덱스가 작을수록 데이터 코드의 출현 빈도가 높다. ECC(Encoding Code)는 코드를 S0 에 가까운 순으로 정렬한 테이블이다. 이 예에서 8 비트 코드 255 는 QLC 플래시 메모리에서 두 개의 셀에 저장되는데, 셀 당 4 비트를 저장하는 두 개의 셀 모두 S0 상태, 즉 이진수로 “1111” 값을 가지는 상태라고 가정하였다. 그리고 코드 127 은 두 셀 중 하나는 S0, 나머지 하나는 S1 인 상태이다. ECC 테이블은 스크램블을 위해 원형 형태의 EW(Encoding Wheel) 테이블로 재구성된다. 또한 ET 와 ECC 로부터 디코딩 테이블을 구성한다. (그림 2)의 DT(Decoding Table)와 DCC(Decoding Code) 테이블은 (그림 1)의 ET 와 ECC 로부터 생성된 디코딩 테이블이다.

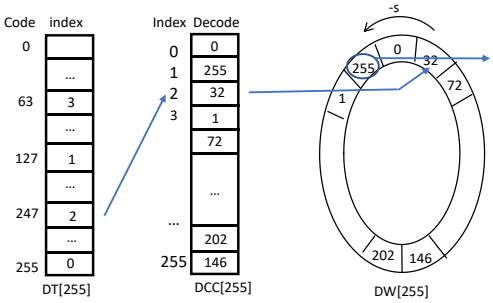
VDSC 기법은 변환 테이블을 사용하여 코드를 변환한 후, 변환된 코드를 대상으로 랜덤 함수로 스크램블을 수행한다. 이 랜덤 함수의 이름은 shift 이며, 데이터 저장 주소를 입력으로 받아 -128 에서 127 사이의 정수 값을 반환한다. Shift 함수는 다양한 확률 밀도 함수로 구현될 수 있다. 만약 shift 함수가 균일한 확률로 -128~127 사이의 정수를 반환하면 순수한 랜덤 스크램블 기법이 된다. 그리고 0 을 반환할 확률이 가장 높고, 0 으로부터 멀어질수록 반환할 확률이 낮아지는 확률 밀도 함수를 이용하여 shift 함수를 구현하면, 코드 변환 테이블의 순위에 비례하여 생성되는 코드의 밀도가 달라진다. 극단적인 경우 shift 함수가 항상 0 만 반환하면, 데이터 코드를 항상 동일한 코드로 변환하는 1:1 변환 기법이 된다.

데이터를 기록할 때 저장장치는 저장할 주소와 데이터를 받게 되며, VDSC 기법으로 데이터를 인코딩한 후 플래시 메모리에 기록한다. (그림 1)은 데이터 코드 255 의 인코딩 과정을 보여준다. ET 에서 코드 225 의 인덱스는 1 이며, 인덱스 1 에 해당하는 ECC 의 엔트리로부터 변환 코드 127 을 얻을 수 있다. 다음으로 주소를 입력으로 주고 shift 함수를 호출하면 -128~127 사이의 정수를 반환한다. 만약 반환 값이 2 라고 가정하면, EW 테이블에서 127 이 위치한 곳에서 시계 방향으로 2 칸 이동하여 코드 247 를 얻게 되며, 이 코드 247 이 최종적으로 변환된 코드이다. 즉 데이터 코드 255 는 247 로 변환되어 지정된 주소에 저장된다.

데이터를 읽을 때 저장장치는 주소를 받게 되며, 해당 주소로부터 읽은 데이터가 247 이라고 가정하자. 저장장치는 247 을 대상으로 디코딩을 수행하며, (그림 2)는 디코딩 과정을 보여준다. DT 에서 코드 247 의 인



(그림 1) 인코딩 테이블



(그림 2) 디코딩 테이블

덱스는 2 이며, 인덱스 2 에 해당하는 DCC 의 엔트리로부터 코드 32 를 얻는다. 그리고 주소를 입력으로 주고 shift 함수를 호출하여 반환 값 2 를 받는다. 이제 DW 에서 32 가 존재하는 곳에서 반 시계 방향으로 2 칸 이동하여 디코딩된 코드 255 를 얻게 된다.

3. 실험 결과

1 장과 2 장에서 언급한 대로 비균일 스크램블을 위해서는 변환 전의 데이터에 비균일성이 존재해야 하며, 이를 확인하기 위해 <표 1>에 나열된 실제 시스템의 저장장치에 있는 데이터를 분석하였다. 분석 결과에 따르면 실험에 사용한 시스템의 저장장치에 존재하는 데이터 코드들은 비균일한 밀도를 가지고 있다.

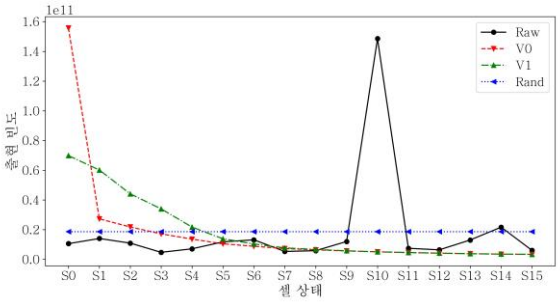
<표 1> 실험에 사용된 시스템들

종류	운영체제	저장장치 사용률
노트북	Win10 pro(v.2004)	32/128 GB
데스크탑	Win10 edu.(v.20H2)	139/512 GB
서버	Ubuntu 18.04 (ker. 5.8.17)	167/256 GB

실험에서는 데이터를 스크램블하여 저장하고, 읽어올 때 원래 데이터로 복원하며, QLC 플래시 메모리를 저장장치로 사용한다고 가정하였다. 실험은 스크램블 기법을 다르게 하면서 <표 1>에 제시된 시스템의 데이터를 저장하는 모의실험 형태로 진행되었으며, 각 기법마다 데이터가 저장된 QLC 플래시 메모리의 상태를 분석하여 성능을 평가하였다.

QLC 플래시 메모리의 경우 각 셀은 S0~S15 까지 16 개의 상태를 가진다. 소거(erase) 직후의 상태가 S0 이고 임계 전압 순으로 S1~S15 의 상태를 가진다고 가정하자. 셀을 원하는 상태로 프로그래밍하는 ISPP 는 번호가 높은 상태일수록 더 많은 단계를 거쳐야 하므로 프로그래밍 시간이 오래 걸리며, 읽기 시간 역시 커진다. 따라서 셀 상태 번호가 작아지도록 데이터를 변환하여 저장하면 프로그래밍과 읽기 속도를 개선할 수 있다. 개선 정도를 평가하기 위하여 VDSC 기법으로 가능하면 S0 에 가까운 셀 상태가 많이 생성되도록 스크램블하고, 기존의 순수 랜덤 스크램블 기법과 성능을 비교하였다.

(그림 3)은 <표 1 >에서 제시된 시스템에 저장된 데이터를 사용하여 실험한 결과를 보여준다. (그림 3)에서 “Raw”는 데이터를 변환하지 않고 그대로 저장할 때 셀 상태의 분포이다. 시스템에 있는 데이터에서 가장 많이 나오는 바이트 코드는 0 이며, 이 코드를 두 개의 QLC 플래시 메모리 셀에 저장하면 두 셀은 모두 S10 상태를 가진다. 따라서 “Raw” 기법의 경우 S10 셀 상태가 가장 많다. “Rand”는 기존의 순수 랜덤 스크램블 기법이며, “Rand” 기법을 사용하면 모든 셀 상태가 균일하게 나타난다. “V0”와 “V1”은 VDSC 기법을 사용하여 스크램블 하며, 셀 상태를 S0 에 가깝게 만드는 코드를 더 많이 생성한다. 특히 “V0”에서 shift 함수는 항상 0 을 반환하며, “V1”의 경우 정규 분포를 사용하여 shift 함수가 0 을 반환하는 확률이 가장 높고 0 으로부터 멀어질수록 반환 확률이 낮아진다.



(그림 3) QLC 에서의 시뮬레이션 결과

(그림 3)의 결과를 보면 “Raw”의 경우 셀 상태는 S10 에 집중되어 있으며, 그 이유는 앞에서 설명하였다. 순수 랜덤 스크램블 기법인 “Rand”는 균일한 셀 상태를 가진다. “V0”는 출현하는 빈도가 높을수록 코드를 셀 상태 S0 에 가까운 코드로 변환하므로, 셀 상태 S0 가 가장 많이 나오고, S1~S15 까지 순서에 따라 셀 상태 사용량이 적어진다. “V1”의 경우에도 S0 상태가 가장 많고 S0~S15 까지 순서에 따라 사용량이 적어진다. 그렇지만 “V1”의 경우 정규 분포 함수로 스크램블 하기 때문에 S0~S15 상태가 정규 분포 형태를 가진다. “V0”와 “V1”을 비교하면, S0 상태는 “V0”에서 더 많으며, S1~S5 상태는 “V1”에서 더 많다. 결국 “V0”는 S0 상태가 월등하게 많고 다른 상태는 적은 반면, “V1”은 S0~S15 상태가 정규 분포 형태로 사용되도록 한다. 앞에서 설명하였듯이, 번호가 큰 상태일수록 ISPP 단계가 많고 프로그래밍과 읽기 시간이 길다. 따라서 프로그래밍에 필요한 ISPP 단계를 기준으로 성능을 평가하면 “Rand” 기법과 비교하여 “V0”는 204%, “V1”은 141%만큼 성능을 향상시킨다.

4. 결론

3D NAND 플래시 메모리의 경우 데이터 코드에 따라 프로그램과 읽기 속도가 다르다. 따라서 빈번히 출현하는 코드를 프로그램과 읽기 속도가 빠른 코드로 변환하여 저장하며 성능을 개선할 수 있다. 본 논문은 변환되는 코드의 밀도를 조절하는 스크램블 기법인 VDSC 기법을 사용하여, 소거 직후의 셀 상태로부터 가장 적은 ISPP 단계를 거쳐 기록할 수 있도록 데이터를 변환하였다. 실제 시스템에 있는 데이터로 실험한 결과는 VDSC 기법이 상당한 수준으로 프로그

래밍과 읽기 단계를 줄일 수 있음을 보여준다. 향후 더 정교한 실험을 통해 VDSC 기법의 성능 향상 정도를 측정할 계획이다.

Acknowledgements

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2016009066).

참고문헌

[1] R. Micheloni, “3D Flash Memories”, Springer, 2016.

[2] K. Suh, B. Suh, Y. Um, J. Kim, Y. Choi, Y. Koh, S. Lee, S. Kwon, B. Choi, J. Yum, J. Choi, J. Kim, and H. Lim, “A 3.3 V 32 Mb NAND Flash Memory with Incremental Step Pulse Programming Scheme,” in IEEE ISSCC Dig. Tech. Papers., San Francisco, CA, USA, Feb. 1995, pp. 128-129.

[3] J. Cha and S. Kang, “Data Randomization Scheme for Endurance Enhancement and Interference Mitigation of Multilevel Flash Memory Devices,” ETRI Journal, vol. 35, pp. 166-169, 2013.

[4] N. Sommer, M. Anholt, O. Golov, U. Perlmutter, S. Winter, and G. Semo, “Data scrambling schemes for memory devices,” US Patent (US-8261159), 2012.

[5] C. Lee, S. Lee, S. Ahn, J. Lee, W. Park, Y. Cho, C. Jang, C. Yang, S. Chung, I. Yun, B. Joo, B. Jeong, J. Kim, J. Kwon, H. Jin, Y. Noh, J. Ha, M. Sung, D. Choi, S. Kim, J. Choi, T. Jeon, H. Park, J. Yang, and Y. Koh, “A 32-Gb MLC NAND Flash Memory With Vth Endurance Enhancing Schemes in 32 nm CMOS,” IEEE Journal of Solid-State Circuits, vol. 46, no. 1, pp. 97-106, 2011.

[6] “NRC: A Nibble Remapping Coding Strategy for NAND Flash Reliability Extension,” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 35, no. 11, pp. 1942-1946, 2016

[7] W. Lee, M. Kang, S. Hong, and S. Kim, “Interpage-Based Endurance-Enhancing Lower State Encoding for MLC and TLC Flash Memory Storages,” IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 27, no. 9, pp. 2033-2045, 2019.

[8] Y. Zhao, W. Tong, J. Liu, D. Feng and H. Qin, "CeSR: A Cell State Remapping Strategy to Reduce Raw Bit Error Rate of MLC NAND Flash," 2019 35th Symposium on Mass Storage Systems and Technologies (MSST), 2019, pp. 161-171.

[9] W. Choi, M. Jung and M. Kandemir, "Invalid Data-Aware Coding to Enhance the Read Performance of High-Density Flash Memories," 2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), 2018, pp. 482-493.

[10] C. Hyun, G. Jeong, S. Yoo, and D. Lee, "Data Scrambling Scheme that Controls Code Density with Data Occurrence Frequency," KIPS Transactions on Computer and Communication Systems, vol. 10, no. 9, pp. 235-242, 2021.