공학석사학위논문

프로그래밍 가능한

플래시 인터페이스 및 이의 응용

Programmable flash interface

and its application

2017 년 8 월

서울대학교 대학원

컴퓨터공학부

이용건

초록

NAND 플래시 메모리는 데이터가 유지되는 비휘발성(Non-volatile) 메모리로 기존 하드 디스크를 대체하는 저장매체로 각광을 받으며 빠르게 발전하여 저장 매체 시장을 점유해 나가고 있다. NAND 플래시 제품이 각광을 받을수록 기술 발전도 빨라지고 새로운 제품 출시도 빨라지고 있지만 통일된 표준 인터페이스가 없기 때문에 NAND 플래시 메모리는 제품마다 조금씩 다른 인터페이스를 가지고 있다. 이러한 환경에서는 NAND 플래시 메모리 제어기의 유연성이 관건이 될 것이며 자칫 오랜 기간 많은 돈을 들여서 개발한 제어기를 짧은 시간밖에 쓸 수 없는 경우가 발생 할 수 있다.

본 논문에서는 서로 다른 인터페이스를 사용하는 상황에서 일관성을 제공하기 위한 방법으로 프로그래밍 가능한 플래시 메모리 인터페이스를 정의하고 이를 FPGA환경을 이용한 NAND 플래시 메모리 제어기로 구현했다. 프로그래밍 가능한 인터페이스는 서로 다른 인터페이스를 갖는 NAND 플래시 메모리 제품에 유연하게 대응할 수 있으며 호스트에 일관성을 제공할 수 있을 것이다. 또한 이러한 유연하고 일관성 있는 인터페이스를 활용하기 위한 응용환경으로 QoS(Quality of Service)기반 공정 대기열 스케줄링을 구현하여 성능과 효과를 검증했다.

주요어 : 플래시 메모리 기반 저장 장치, 플래시 메모리 컨트롤러, 플래시 메모리 동작 스케줄링, 공정 대기열

학 번 : 2015-22910

목차

제 2장 배경 지식 및 관련 연구

제 3장 프로그래밍 가능한 플래시 인터페이스

제 4장 실험 환경 및 평가

그림 목차

표 목차

제 1 장 서론

1.1 연구 동기

NAND 플래시 메모리는 전자적 방식을 통하여 집적 소자에 데이터를 저장하는 방식을 사용함으로 비교적 빠른 속도로 데이터를 저장하고 지울 수 있으면서, 전력 소비가 적고, 외부 자극에 대한 강한 내구성을 갖고 있는 등의 장점이 있다. 이러한 장점들로 인해 NAND 플래시 메모리는 모바일 기기의 저장 장치 시장을 석권하고 기술 발전을 선도하고 있으며 기존 PC 시장의 하드 디스크를 대체하고 더 나아가 서버 시장마저도 점유해 나가고 있다.

NAND 플래시 메모리 기술은 가격 경쟁력을 증대하기 위하여 같은 면적에 싼 가격으로 더 많은 데이터를 저장하는 쪽으로 발전해 왔다. 소자 기술은 하나의 소자에 다수 비트(Bit)를 저장할 수 있는 기술로 발전하였고, 집적을 위한 설계 공정 기술은 내부 선 폭을 줄여서 단위 집적도를 높이는 방향으로 발전하였다. 또한 최근에는 3D NAND의 개발로 또 한번 집적도를 크게 개선할 수 있는 계기를 마련했다. NAND 플래시를 기반으로 한 기술 발전은 점점 가속화 되고 새로운 제품이 나오는 주기가 빨라지는 추세이다. 이러한 기술 발전 동향은 더 빠르고

효율적인 동작을 위해 One Shot 프로그램 [1] 등의 기존에 없던 새로운 플래시 메모리 연산의 개발로 이어졌으며 기존에 없던 새로운 연산을 지원 하기 위한 인터페이스의 변화로 이어지고 있다. NAND 플래시 메모리는 통일된 하나의 인터페이스가 없기 때문에 제품별로

인터페이스가 다른 결과로 이어진다. NAND 플래시 메모리를 사용한 저장 장치에서는 호스트가 각 NAND 플래시 제품 인터페이스의 디테일을 모르더라도 간단하게 동작을 컨트롤 할 수 있도록 NAND 플래시와 호스트 사이에는 플래시 제어기를 두는 것이 일반적이다. 제어기는 호스트에서 오는 읽기, 쓰기, 지우기의 각 연산을 NAND 플래시 제품 인터페이스에 맞게 해석하여 신호를 만들어주는 역할을 하며 제품에 따라서 그 제품의 인터페이스에 알맞게 대응하는 서로 다른 제어기가 필요하게 된다. 결과적으로는 제품에 따라 다른 제어기가 필요하게 된다. 이러한 상황에서는 서로 다른 인터페이스에 적응 할 수 있는 유연성을 갖춘 동시에 하나의 공통적인 인터페이스를 제공 할 수 있는 제어기가 필요하다.

플래시 메모리는 장점만 가지고 있는 것이 아니라 단점도 동시에 가지고 있다. 제자리 갱신 동작이 불가능하여 갱신 전 먼저 지우기 연산을 수행해야 하고, 읽기와 쓰기 연산간에 속도 불균형, 쓰기 영역과 지우기 영역 간의 불균형 등의 비대칭 부분이 있으며, 지우기 수명에 제한이 있어서 어느 부분을 지우고 다시 쓸지에 대한 관리가 필요하다 [2]. 이러한 단점 중 비대칭 부분을 극복하고 기존 하드디스크가 제공하는 것과 동일한 인터페이스를 호스트에 제공하기 위하여 다수의 NAND 플래시 칩을 하나의 버스에 연결하여 병렬 처리를 하는 방법을 사용하고 있다. 이렇게 공유된 버스는 NAND 칩 간의 공유 자원이 되며 공유 자원을 효율적으로 쓰면서 동시 동작성을 높이고 병렬 처리를 잘 하기 위한 스케줄링이 필요하게 된다. 스케줄링의 방법에는 소프트웨어로 스케줄링을 하는 방법과 컨트롤러에 연결된 하드웨어로 스케줄링을 하는 방법이 있다. 하드웨어로 스케줄링을 하는 방법을 사용하면 특정 상황에 최적화된 결과를 만들어 낼 수 있고 호스트가 스케줄링을 신경 쓰지

않고 읽기 쓰기 지우기 요청을 보낼 수 있는 장점이 있는 대신 여러 가지 스케줄링 기법을 사용할 수 없고 소프트웨어로 스케줄링을 하는 방법을 사용하면 다양한 스케줄링 방법을 유연하게 사용할 수 있다.

또한 스케줄링의 기준을 무엇으로 할 것인가도 중요한 문제이다. NAND 플래시 메모리 제어기에서의 연산 처리율만을 기준으로 하면 경우에 따라서 호스트에 원하는 성능을 제공하지 못 하는 상황이 발생할 수 있다. 이러한 상황에서는 각각의 스트림(Stream)에 성능을 보장해 줄 수 있는 방법이 필요하다.

1.2 연구 내용

본 논문에서는 NAND 플래시와 관련 된 기술이 빠르게 변화 발전하는 상황에서 제품과 인터페이스에 상관없이 호스트에 일관성을 제공하기 위하여 프로그램 가능한 NAND 플래시 인터페이스를 갖는 컨트롤러를 구현하고 이를 실제로 적용하는 응용 예시로 공정 대기열을 사용한 소프트웨어 스케줄링을 구현 했으며 잘 정의된 프로그래밍 가능한 NAND 플래시 제어기를 사용하면 프로그램이 가능한 인터페이스로 서로 다른 제품과 요구사항에 대응할 수 있는 유연성을 확보하는 동시에 호스트에는 일관성과 함께 간단한 인터페이스를 제공하여 호스트에서의 작업 편의성을 높아지는 것을 확인했다.

1.3 논문의 구성

본 논문의 전체적인 구성은 다음과 같다. 2장에서는 배경지식으로 플래시 메모리 기반 저장 장치의 구성요소와 각각의 특성 및 역할에 대해 알아보면서 주요 요소들과 함께 NAND 플래시 스케줄러와 제어기를 살펴 볼 것이며, 동시에 NAND 플래시 메모리를 효율적으로 사용하기 위한 스케줄링 기법들을 살펴 볼 것이다. 또한 NAND 플래시 인터페이스의 역사와 현황에 대해서도 알아 볼 것이다. 3장 에서는 프로그래밍 가능한 플래시 인터페이스의 정의와 이를 제어기로 구현하기 위한 디자인 그리고 이러한 인터페이스를 응용하여 공정 대기열 스케줄링을 구현하는 것을 설명한다. 4장에서는 구현한 컨트롤러와 스케줄링 방법의 성능을 평가하며 마지막으로 5장에서 본 연구에 대한 결론으로 논문의 끝을 맺는다.

제 2 장 배경 지식 및 관련 연구

2.1 NAND 플래시 메모리

플래시 메모리란 비휘발성 컴퓨터 기억 장치의 일종으로 비트 정보를 저장하는 셀이라 부르는 플로팅 게이트 트랜지스터(Floating Gate Transistors)를 기본 소자로 하여 다수의 셀로 구성된 2차원 배열 안에 정보를 전자적인 방식으로 정보를 저장하는 방식을 사용한다. 최근 개발된 3D NAND에 이르러서는 플로팅 게이트 트랜지스터 대신 차지 트랩 플래시(Charge Trap Flash) 를 기본 소자로 사용하고 있으며 기존 플로팅 게이트 트랜지스터는 게이트와 서브 사이에 존재하는 도체에 데이터를 저장하는 방식이었지만 차지 트랩 플래시는 게이트와 서브 사이에 존재하는 절연체에 데이터를 저장하는 방식을 사용한다.

플래시 메모리는 NOR 플래시 메모리와 NAND 플래시 메모리로 나눌 수 있다. NOR 플래시 메모리는 저장단위인 셀을 병렬로 배열하는 구조로 바이트(Byte)단위로 데이터 접근이 가능하고 원하는 데이터를 빨리 찾을 수 있어 NAND 플래시 메모리보다 상대적으로 읽기 속도가 빠르고 데이터의 안전성 측면에서도 우위를 갖는다. 반면 쓰기 동작 시에는 셀의 주소를 찾는 과정이 필요하여 NAND 플래시 메모리 보다 쓰기 속도는 느리고 각 셀의 주소를 기억하고 접근해야 하기 때문에 회로가 복잡하여 NAND 플래시 메모리에 대비하여 상대적으로 데이터를 저장할 수 있는 공간이 좁아져서 집적이 어렵다는 단점이 있다.

NAND 플래시 메모리의 경우 저장 단위인 셀을 직렬로 배열하는 구조이기 때문에 상대적으로 좁은 면적에 많은 셀을 집적할 수 있는 장점이 있고, 직렬로 연결된 다수의 셀로 이루어진 페이지 단위로 쓰기 동작을 하기 때문에 NOR 플래시 메모리에 비해 쓰기 속도 또한 빠르다. 집적이 용이한 NAND 플래시 메모리의 소자 기술은 빠르게 발달하고 있으며 과거 한 셀에 한 비트만을 저장했던 SLC(Single Level Cell) 방식의 한계에서 벗어나 2비트를 저장 할 수 있는 MLC(Multi Level Cell)를 거쳐서 3비트를 저장할 수 있는 TLC(Triple Level Cell) 방식으로의 발전해 나가고 있다. 또 설계 공정 기술도 발전하고 있어서 사용하는 선폭을 10nm 대까지 줄여서 같은 공간에 더 많은 소자를 집적할 수 있게 되었다. 최근에는 소자를 3차원으로 배열하는 3D NAND가 개발되어 집적화에 있어서 또 한번의 도약을 이루었다. 이를 통해 같은 용량 대비의 소형화 및 경량화 그리고 같은 크기 대비의 대용량화가 가능해지게 되고 있다 [3]. 이처럼 쓰기 연산이 빠르고 소형화, 대용량화가 용이한 특성을 가지는 NAND 플래시 메모리는 다양한 모바일 기기부터 기업 단위의 서버시장까지 넓고 다양한 분야의 저장 장치로 사용되고 있으며 기존 하드 디스크 기반의 저장 장치가 점유하고 있던 시장을 빠른 속도로 대체해 가고 있다.

NAND플래시 메모리에는 여러 단점이 존재한다. NAND 플래시 메모리 칩 하나는 크게 두 개 정도의 플레인(Plane)이라고 부르는 병렬 처리 단위로 나누어지고 이러한 플레인은 지우기 연산의 단위가 되는 다수의 블록(Block)으로 이루어져 있다. 블록은 쓰기와 읽기의 단위가 되는 다 수의 페이지(Page)로 이루어져 있고 하나의 페이지는 NAND 플래시 메모리 기본 소자인 다수개의 셀로 구성되어 있다. 이렇듯이 NAND 플래시 메모리에는 지우기와 쓰기/읽기의 기본 단위에 비대칭성이

존재하며, 가장 긴 시간이 소요되는 지우기 연산과 지우기 보다는 짧지만 읽기 보다는 긴 쓰기 연산 그리고 가장 짧은 시간이 소요 되는 읽기 연산의 연산 시간의 비대칭성이 존재한다.

NAND 플래시 메모리는 한 번 쓴 데이터에 대하여 제자리 덮어쓰기는 불가능하며 반드시 지운 후 써야 하는 저장 장치의 역할을 하기에는 치명적으로 보이는 단점이 존재한다. 지우기 연산은 한 번 썼던 데이터의 유효 여부를 확인하여 다른 곳에 다시 써 줘야 하는 추가 작업이 필요할 수도 있다. 한 블록에 지우기 연산을 할 수 있는 회수에는 제한이 있어서 전체 칩의 블록을 고르게 지우지 않으면 점점 가용한 블록의 개수가 줄어드는 효과가 발생한다. 지우기 회수는 SLC>MLC>TLC 방식 순으로 적어지며 집적도가 큰 기술을 사용한 제품일수록 – 다시 말해 최신 기술을 적용한 제품일수록 – 지우기 횟수가 적다.

쓰기 연산은 작은 페이지 번호부터 써야 하는 쓰기 규칙이 존재하며 실제 쓰는데 걸리는 시간은 페이지마다 조금씩 다르다. 사실 NAND 플래시 메모리는 한 플레인 안에 존재하는 블록 간에도 지우기 성능에 차이가 있고 더 나아가 하나의 페이지를 구성하는 셀 간에도 성능 편차가 존재한다. 또한 이러한 편차는 SLC<MLC<TLC순으로 커진다 [4].

또한 불량 셀의 개수와 불량 블록의 개수를 관리하여야만 한다. NAND 플래시 메모리는 블록당 일정 개수의 불량 발생을 허용하며 일정 개수의 셀 불량이 발생한 블록은 불량 블록으로 관리를 해야 한다. 현재 정상적인 블록이라도 지우기 횟수가 증가하면 불량이 증가하게 되어 불량 블록이 될 수도 있다. 이러한 불량 발생 빈도 역시 집적 집적도가 큰 기술을 사용한 제품에서 더 높게 나타난다.

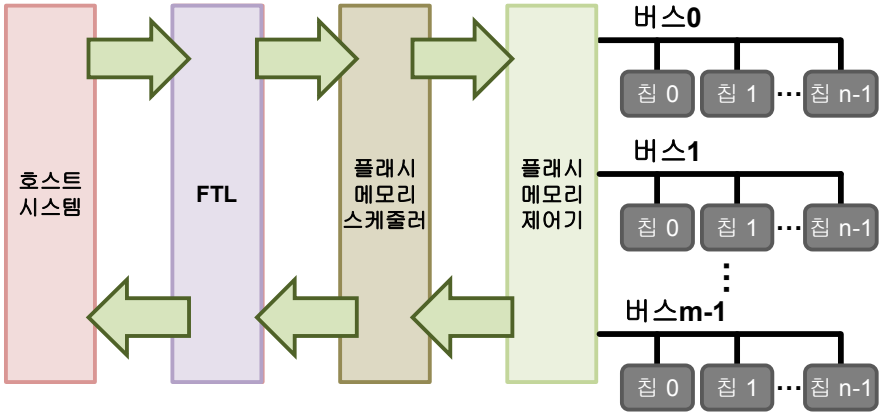

그림 1. 플래시 메모리 기반 저장 장치 구조

2.2 NAND 플래시 메모리 기반 저장 장치

그림 1은 플래시 저장 장치의 구조도 이다. 플래시 저장 장치는 플래시 변환 계층(Flash Translation Layer 줄여서 FTL), 플래시 메모리 제어기(Controller) 와 여러 개의 버스 그리고 버스를 공유하는 여러 개의 NAND플래시 메모리 칩들로 이루어져 있다. 여러 개의 칩들이 버스를 공유하는 것은 병렬 처리를 위한 구조이며 서로 영향을 주지 않는 동작 요구들을 서로 다른 버스에 병렬로 인가하고 이를 다시 서로 다른 칩에서 병렬로 처리하여 병렬 처리율을 높이는 방식을 사용함으로 NAND 플래시 메모리의 단점을 보완할 수 있다. 이제 각 계층의 역할을 좀 더 자세 히 설명해 보고자 한다.

2.2.1 플래시 변환 계층

NAND 플래시 메모리는 장점과 단점을 동시에 갖고 있기 때문에 저장 장치로서의 장점을 활용하고 단점을 보완하면서 기존 저장 매체인 하드디스크와 동일한 인터페이스를 호스트에 제공하기 위하여 플래시 변환 계층이라 불리는 별도의 계층을 내부적으로 사용하는 것이 일반적이다. 플래시 변환 계층은 제자리 갱신이 되지 않아서 발생하는 주소변환과 쓰레기 수집, 마모 한계를 가진 NAND 플래시 셀들의 고른 수명 관리를 위한 마모 평준화, 성능 향상을 위한 병렬 처리 그리고 과도한 발열로 인한 에너지 소모를 방지하기 위한 열 조절 등 저장 장치로서 NAND 플래시에 필요한 동작들을 제어하는 역할을 한다.

2.2.1.1 주소 변환

컴퓨터에서 사용되는 기본 연산 단위는 섹터이다. NAND 플래시 메모리의 경우 읽기/쓰기 연산의 경우엔 페이지, 지우기 연산의 경우엔 블록을 기본연산 단위로 사용하기 때문에, 호스트에 하드 디스크를 사용하는 것과 동일한 인터페이스를 보장하기 위해서는 주소 변환을 사용하여야 한다. 주소 변환을 통하여 논리적인 섹터 주소를 물리적인 블록과 페이지로 변환하여 기존 섹터 단위를 사용하여 기존 하드 디스크 위에서 동작하던 파일 시스템, 운영체제, 프로그램들이 아무런 문제없이 NAND 플래시 메모리 기반 저장 장치를 사용할 수 있도록 만들어 준다.

이러한 동작은 파일 시스템이나 운영체제의 개입 없이 즉 별도의 수정 없이, 호스트가 사용하는 논리적인 섹터 주소와 NAND 플래시 저장 장치의 물리적인 블록 및 페이지 주소 사이에 사상(Mapping)을 유지함으로 이루어지며 이들 사이에 사상 테이블을 유지해야 한다.

2.2.1.2 쓰레기 수집

제자리 덮어쓰기가 불가능한 NAND 플래시 메모리의 특성 때문에 쓰기 요청은 모두 새로운 블록에 적으며 기존에 적었던 논리 주소를 덮어쓰기 하는 경우에는 새로운 페이지에 적고 논리 주소와 새로운 물리 주소를 연결하고 기존 정보를 담고 있는 물리 주소를 무효화 처리하는 방식을 사용한다. 이 때 무효화 처리는 사상을 유지하는 사상 테이블 상에서만 이루어 진다.

이렇게 새로운 페이지에 적는 방식을 반복하다 보면 결국에는 더 쓸 새로운 블록이 없어 질 것이며 이 때에는 기존 블록 중 가장 무효화 된 페이지가 많이 들어 있는 블록을 골라 아직 유효한 페이지는 새로운 블록에 적고 사상 테이블을 수정한 후 해당 블록에 지우기 동작을 실행하여 쓰기가 가능한 새로운 블록을 만들어 내는 쓰레기 수집 작업을 진행하여야 한다.

이러한 쓰레기 수집 작업은 경우에 따라서 호스트의 요청과 경쟁을 하게 될 수 있다. 최악의 경우는 지우기 작업을 진행한 새로운 블록이 없어서 쓰레기 수집 작업을 진행해야만 하는데 호스트의 요청도 많은 경우이다. 이러한 쓰레기 수집 작업이 호스트 요청의 관계는 이후에 기술할 공정 대기열 스케줄링 [5]이 필요한 근거가 된다.

2.2.1.3 마모 평준화

NAND 플래시 메모리의 셀은 지우기 동작 가능 횟수의 상한이 정해져 있다. 지우기 동작은 블록 단위로 이루어지기 때문에 실제로는 NAND 플래시 메모리의 각 블록 단위로 지우기 횟수를 관리하는 것이 필요하다. 제자리 덮어쓰기가 불가능한 NAND 플래시 메모리의 특성 때문에 특정 페이지에 집중된 쓰기 작업이 일어나는 것은 집중된 지우기 작업이 동반되는 것을 의미하며, 지우기가 자주 일어나는 블록 전체의 수명이 줄어드는 결과를 초래하고 결국 해당 블록은 수명을 일찍 소모하여 다른 블록 보다 빨리 사용할 수 없는 상태에 도달하여 마치 용량이 줄어드는 듯한 결과가 발생할 것이다.

이러한 NAND 플래시 메모리의 특성 때문에 각각의 페이지들이 골고루 쓰여져야 하지만 하드 디스크와 동일한 인터페이스를 제공해야 하기 때문에 상위 계층에서는 플래시 메모리의 특성을 고려하지 않고 페이지를 사용하게 된다. 결국 페이지 간 쓰기 횟수의 불균형 문제를 플래시 변환 계층 단계에서 해결하게 되는데 이를 마모 평준화라고 한다. 현재 다양한 방식의 플래시 변환 계층이 존재 하며 각각의 플래시 변환 계층은 서로 다른 마모 평준화 방법을 사용할 수 있지만 기본적으로 각 페이지의 쓰기 횟수를 고려하여 새로운 페이지를 할당하여 각 블록의 지우기 연산의 횟수를 조절하는 것은 동일하다.

2.2.2 NAND 플래시 메모리 스케줄러

플래시 메모리 저장 장치를 효율적으로 사용하기 위해 각각의 버스와 칩을 동시에 동작시키는 병렬 처리를 최대한 이용하게 된다. 만일 각 연산들이 서로 독립적이고 서로 영향을 주지 않는 다면 여러 칩에서 동시에 진행하여 병렬 처리하는 것이 가능하게 될 것이며 이 때 공유 자원은 여러 개의 칩들이 공유하는 공유 버스가 된다.

공유 버스가 점유되는 시기는 각 칩으로 명령을 보낼 때와 해당 칩으로 데이터를 전송하거나 해당 칩에서 데이터를 받을 때가 되기 때문에 지우기 연산이 다른 칩에서 진행되는 도중 데이터를 주고 받는 방식으로 병렬 처리율을 높일 수 있다. 공유 버스를 사용하는 방식을 연산의 동시 동작성 여부와 요청된 순서에 대한 보존 여부를 기준으로 분류하여 순차 실행 방식, 순서 중첩 실행 방식, 무순서 중첩 실행 방식으로 나눌 수 있다.

순차적 실행 방식은 동시 동작성을 배제하여 요청을 순서를 유지하며 실행하는 방식으로 모든 연산은 원자적으로 처리되며 동시에 연산이 처리되지 않고 모든 연산을 도착 순서대로 실행한다. 즉 버스를 공유하는 여러 개의 칩 중에서 한 번에 한 칩에서만 하나의 연산이 수행되는 방식이다.

두 번째 방법은 동시 동작성을 인정하여 지연 단계에서의 다른 연산의 수행이 진행되지만 순서는 도착 순서로 보존하여 시작 및 종료 시간의 순서는 지켜지는 연산 방식이며 이를 순서 중첩 실행 방식이라고 한다.

마지막은 무순서 중첩 실행 방식으로 연산들의 도착 순서에 상관 없이 동시 동작성을 최대로 활용하여 병렬 처리를 최대한 효율적으로 진행하는 방식이다. 무순서 중첩 실행 방식을 사용하더라도 나오는 결과를 다시 순서대로 배열하는 것이 필요하며 각 연산 요청이 들어온 순서를 타임 스탬프를 사용하여 기록하고 리오더 버퍼(Re-order Buffer)를 사용하여 연산 처리 결과를 다시 시간 순서대로 배열하여 호스트로 전달하는 방식을 사용한다.

Ozone 플래시 제어기는 이러한 세가지 NAND 플래시 스케줄링 모델을 하드웨어로 구현하여 성능평가를 진행하였으며, 성능 측면에서는 동시 동작성을 최대한 활용하여 병렬 처리율을 높인 방식인 무순서 중첩 실행 방식이 가장 좋고 순차 실행 방식이 가장 떨어지는 것을 확인하였다 [6]. 그러나 이러한 스케줄링 방식들은 연산들의 관계를 생각하지 않은 채 시스템 전체의 처리량을 늘리는 것에만 초점을 맞춘 것이라는 한계가 있다.

2.2.2.1 QoS 기반 스케줄링

NAND 플래시 스케줄링 방법에 대하여 간단하게 살펴 보면서 컨트롤러 레벨의 스케줄링 방법의 필요성과 의의 그리고 처리율을 최대로 높이기 위한 스케줄링 방법인 무순서 중첩 실행 방식을 살펴 보았다. 그러나 실제 NAND 플래시 저장 장치의 동작 상황에서는 두 개 이상의 스트림이 존재할 때 특정 스트림의 성능을 보장해야 하는 상황이 발생할 수 있다. 예를 들어 호스트의 요청에 의한 연산과 플래시 변환 계층의 쓰레기 수집이 동시에 일어나고 있다고 가정해 본다. 호스트의

요청에 의한 연산이 많이 발생하고 있고 쓰레기 수집으로 미리 지워둔 블록이 충분한 수가 아니라면 쓰레기 수집작업에서 수행하는 플래시 연산들을 빨리 처리해 주는 것이 호스트의 요청 수행도 빨라지게 할 수 있는 상황이 될 수 있다. 하지만 이미 쓰레기 수집으로 지워둔 블록을 충분히 확보한 상황이라면 호스트의 요청을 우선적으로 처리하는 것이 더 좋은 선택이 될 것이다. 이러한 상황에서 현재 먼저 처리할 것이 무엇인지를 판단하지 않고 단순하게 전체 처리율을 우선 순위 판단의 근거로 한다면 각각의 상황에서 적절한 성능을 보장하는 것이 쉽지 않을 것이다.

QoS기반 스케줄링의 가장 간단한 구현은 우선 순위 스케줄링이다. 다수의 스트림 중 가장 긴급하게 처리를 하여야 할 스트림에 우선 순위를 주어 다른 스트림보다 빨리 처리 될 수 있도록 하는 방법이다. 이러한 방법은 자칫하면 우선 순위가 낮은 스트림이 처리가 전혀 되지 않는 극단적인 상황을 발생시킬 수 있다. 따라서 우선 순위를 동적으로 변하게 하여 이런 상황을 극복하는 것이 필요하다. 우선 순위 스케줄링의 단점을 보완한 것이 가중치를 준 우선 순위 스케줄링 방법이다. 실행 횟수를 가중치로 나누어서 더하여 가장 낮은 숫자의 가중치를 가진 스트림을 실행하는 방법으로 모든 스트림이 실행 될 수 있도록 보장하는 방법이다. 그러나 이러한 방법을 사용하더라도 실행 대기열의 상황을 반영하지 못한다는 단점이 있다. 우선 순위를 가진 요청들이 실행 대기열을 가득 점유하고 있을 때 우선 순위가 낮은 요청들의 응답 시간은 보장 할 수 없는 상황이 된다.

마지막으로 대기열의 혼잡까지 고려한 가상시간을 사용한 공정 대기열 스케줄링 방법이 있다 [7]. 이 방식은 혼잡으로 인해 대기열에서 기다리는 시간을 고려해서 가중치로 주는 방식이기 때문에 대기열에서

오래 기다린 스트림의 요청들이 우선 순위가 높아지는 방식이다. 가상시간을 사용한 공정 대기열 방식은 그러나 각 요청들이 시작한 시간과 끝난 시간을 기록해야 하는 부담이 있다.

하드웨어 스케줄링으로 구현된 QoSFC는 가상 시간을 기준으로 하여 다수의 플래시 변환 계층 작업이 동시에 발생하였을 때 시스템의 상황에 따라 우선 순위를 변하게 하여 평균 응답률을 높였다 [8].

2.2.2.2 하드웨어/소프트웨어 스케줄링

하드웨어로 스케줄링을 구현하는 경우 특정 상황에 최적화된 결과를 얻어내는 것이 가능하지만 동시에 여러 가지 스케줄링 기법을 사용하는 것이 가능하도록 설계하는 것은 어렵다. 소프트웨어로 스케줄링을 구현하는 경우 하드웨어에 비해 최적화된 결과를 내기는 어렵지만 여러 가지 방법을 유연하게 적용할 수 있다는 장점이 있으며 기준점이나 작동방법을 동적으로 구현하여 상황에 맞는 스케줄링 방법을 사용할 수 있는 등의 유연함을 얻을 수 있다.

앞서 기술한 Ozone은 하드웨어로 스케줄링을 구현한 예이고 SOS는 Ozone에서 사용한 스케줄링 방법을 소프트웨어로 구현한 예이다. SOS [9]는 Ozone에서는 다루지 못했던 내용인 대기열 간 부하 재분배와 동일 논리 주소로 가는 쓰기 연산의 반복을 제거하여 Ozone보다 개선된 성능을 보여 주었다. 그러나 SOS의 경우 소프트웨어 스케줄링에서 얻는 유연성을 이용하기 보다는 보다는 응답률을 높이는 방법론에 집중한 경우이다.

2.2.3 NAND 플래시 메모리 제어기

NAND 플래시 메모리 컨트롤러는 호스트에서 보내고 플래시 변환 계층을 거쳐서 오는 읽기 쓰기 지우기 동작을 받아서 이를 수행할 칩에 실제 동작을 수행할 수 있는 신호를 만들어주는 역할을 한다. 즉 호스트의 요청을 NAND 플래시 칩이 이해할 수 있는 신호로 변환해 주는 역할을 하는 것이며 이 때 어떤 인터페이스를 가진 NAND 플래시 메모리 칩이냐에 따라서 그 칩과 약속된 인터페이스의 신호 체계로 통신을 해야만 한다. NAND 플래시 메모리의 쓰기 지우기 동작 속도는 페이지 별, 블록 별로 속도가 조금씩 다르며 하나의 페이지나 블록의 경우도 지우기 횟수에 따라서 속도가 조금씩 변하게 된다. 쓰기나 지우기 동작이 일정시간 이상 걸리면 불량으로 간주하지만 불량이 되지 않는 한계 내에서는 조금씩 달라 질 수 있기 때문에 해당 NAND 플래시 메모리 칩이 다음 신호를 받을 수 있는 준비가 되어 있는 가를 확인하는 인터페이스가 포함이 되어 있다.

NAND 플래시 메모리의 제어기에서 연산 스케줄링 하는 경우도 있지만 본 논문에서는 소프트웨어로 구현된 메모리 스케줄러를 후술하기 위하여 제어기와 스케줄러를 구분하여 설명한다.

2.3 NAND 플래시 메모리 인터페이스

먼저 대표적인 플래시 메모리 인터페이스인 ONFI와 Toggle Mode

인터페이스를 간략하게 살펴 보면서 플래시 메모리 인터페이스의 발전과정을 알아 본 후 세부 인터페이스에 대하여 알아볼 것이다.

2.3.1 ONFI 와 Toggle Mode 인터페이스

ONFI(Open NAND Flash Interface)는 NAND 플래시 메모리끼리 인터페이스를 표준화 시켜서 상호 호환성을 높이기 위하여 SK하이닉스 마이크론 인텔 등의 메모리 제조 회사가 참여하여 만든 공개 표준 인터페이스이다 [10]. ONFI의 특징은 다음과 같다.

- 다양한 장치 기능 및 새로운 변화 지원
- 기존 NAND 플래시 인터페이스와 호환성 제공
- 기능 및 성능에 대한 명시를 칩에서 자체 제공
- 새 플래시 디바이스에 대하여 호환성 제공
- 기존 인터페이스와 호환되는 하이 스피드 인터페이스 정의
- 코어 전압과 (Vcc) 레일과 I/O(VccQ)레일의 분리 허용

ONFI1.0에서 비동기식 표준으로 시작하여 ONFI2.0에서 동기식 표준을 잠시 따랐고 이후는 비동기식 표준으로 따르는 것으로 다시 변경되었다. 현재 ONFI4.0이 마지막으로 정의된 표준이다. ONFI계열에서는 기능 및 성능에 대한 명시를 칩에서 자체 제공해야 하기 때문에 기본 쓰기 읽기 지우기 연산 외에 현재 칩이 ONFI의 어떤 Version을 따르고 있는지 알 수 있도록 해주는 연산을 제공해야 한다.

ONFI를 사용하는 칩들은 동일한 인터페이스를 공유하게 되지만 이러한 표준 인터페이스에는 두 가지 한계가 있다. 기존에 만들어진 인터페이스와 호환되는 것을 목표로 하였기 때문에 ONFI3.0부터는

비동기식과 동기식을 모두 지원해야 하는 등의 호환성을 유지하게 만들어져야 하는 부담이 있다. 또한 가장 큰 점유율을 가진 NAND 플래시 메모리 제조 회사인 삼성과 도시바가 참여하지 않은 인터페이스라는 한계를 가지고 있다. ONFI4.0이 마지막 개편된 2014년이후로는 개편이 더 이상 없다.

Toggle Mode는 ONFI에서 빠진 삼성과 도시바가 만든 표준으로 ONFI처럼 공개되지 않은 인터페이스다. 삼성과 도시바는 오랜 기간 동안 NAND 플래시 제조 회사 중 점유율 1,2위를 차지하고 있기 때문에 Toggle Mode의 영향력을 무시할 수는 없다. 비동기식에서 동기식을 거쳐 다시 비동기식으로 발전한 ONFI와는 다르게 비동기식으로 꾸준히 발전해 왔다. 공개된 표준이 아니기 때문에 해당 제조사의 칩을 사야만 인터페이스에 대한 명세를 받아 볼 수 있다.

ONFI이후로는 통일된 표준을 만들고자 하는 노력은 없으며, NAND 플래시 인터페이스를 알기 위해서는 각 제조회사가 제조한 제품을 구입하여 제조사가 제공하는 명세를 확인해야만 한다.

2.3.2 NAND 플래시 세부 인터페이스

플래시 메모리 인터페이스는 핀과 신호 구성 등의 물리 인터페이스와 칩 내부 구조와 주소 체계가 정의 되는 메모리 구조 인터페이스, SDR DDR등의 데이터를 주고 받는 방식과 이에 따른 요구 사항이 정의된 데이터 인터페이스, 각 칩이 지원하는 연산을 정의한 연산 인터페이스, 연산을 인가하기 위한 명령을 정의한 명령 인터페이스, 연산과 명령의 시간 요구 사항을 정의한 타이밍 인터페이스 등으로

나누어 볼 수 있다. 이러한 각각의 인터페이스는 서로 영향을 주고 받게 되는데, 예를 들어 물리 인터페이스의 변하게 되면 타이밍 인터페이스와 명령 인터페이스도 변해야만 하는 경우가 생길 수 있다.

2.3.2.1 물리 인터페이스

NAND 플래시 칩의 패키지 구성과 각 패키지 구조에서 필요한 핀 구성, 그리고 지원하는 데이터 인터페이스에 따라 정의되는 신호 요구사항 및 각 핀에 인가해야 하는 신호의 AC/DC특성 요구 사항 등이 포함된다.

플래시 칩의 패키지는 칩의 AC/DC특성이 좋아지는 방향 즉 빠른 동작 속도와 신호인가를 지원하고 잡음을 잘 견딜 수 있으면서 얇고 가벼우면서도 온도 변화에 잘 견디는 구조로 발전하고 있다. 과거 Lead를 사용하는 TSOP 구조에서 볼을 사용하는 LGA, BGA를 거쳐서 작은 면적에도 적용 가능한 fBGA로 발전해 왔다.

핀 구성은 어떤 데이터 인터페이스를 지원하느냐에 따라 요구사항이 달라지는데 보통은 가장 많은 핀이 필요한 경우의 데이터 인터페이스를 지원할 수 있게끔 만들고 해당 핀이 필요 없는 데이터 인터페이스를 사용할 때는 핀에 신호를 인가하지 않아도 동작할 수 있는 호환성을 제공하는 방식을 사용한다.

Signal Name	Input / Output	Description
R/B_x_n	O	**Ready/Busy** The Ready/Busy signal indicates the target status. When low, the signal indicates that one or more LUN operations are in progress. This signal is an open drain output and requires an external pull-up.
RE_x_n (RE_x_t)	I	**Read Enable (True)** The Read Enable (True) signal enables serial data output. This signal shares the same pin as W/R_x_n in the NV-DDR data interface.
RE_x_c	I	**Read Enable Complement** The Read Enable Complement signal is the complementary signal to Read Enable True, optionally used in the NV-DDR2 or NV-DDR3 data interface. Specifically, Read Enable Complement has the opposite value of Read Enable True when CE_n is low, i.e., if RE_x_t is high then RE_x_c is low; if RE_x_t is low then RE_x_c is high.

표 1. 데이터 인터페이스에 따라서 달라지는 핀 구성의 예. Read Enable핀의 경우 DDR2 Mode에서만 RE_c를 사용한다.

각 핀에 인가해야 하는 신호의 AC/DC 특성도 지원하는 데이터 인터페이스에 따라 다르게 정의가 된다.

Parameter	Maximum Overshoot area above VccQ and Maximum Undershoot area below VssQ
	SDR
All timing modes	3 V-ns
	NV-DDR
Timing Modes 0-2	3 V-ns
Timing Mode 3	2.25 V-ns
Timing Mode 4	1.8 V-ns
Timing Mode 5	1.5 V-ns

표 2. 동작 모드에 따라서 DC특성이 달라지는 예. VccQ의 전압 요구는 SDR과 NV DDR이 다르다.

그 외에 기본적인 동작에 대한 DC 요구사항도 물리 인터페이스로 정의 된다. 이러한 물리 인터페이스에서 변화가 생기는 것은 다른 인터페이스에까지 변화가 생기는 상황이 될 가능성이 크다. 예를 들어

지금까지 사용하지 않았던 새로운 핀을 사용하게 된다면 각각의 연산에서 그 핀에 어떤 신호를 어떻게 주어야 하는지에 대한 정의가 필요하게 될 것이다. 또한 현재 사용하고 있는 핀을 더 이상 사용하지 않는다면 이를 이용한 인터페이스가 모두 개정되어야 할 것 이다.

2.3.2.2 메모리 구조 인터페이스

명령어를 인가하여 연산을 실행하는 과정은 메모리 구조에 영향을 받는다. LUN(Logical Unit Number)은 독립적으로 명령어를 인가하여 연산을 실행할 수 있는 최소 개체로 정의 되며 개개 칩을 의미한다. LUN의 개수에 따라서 CE핀의 구성이 달라지게 되며 최대로 활용할 수 있는 병렬성도 칩의 개수로 한정이 된다. 어떤 제품을 사용하는가에 따라서 사용할 수 있는 LUN의 개수는 달라지며 CE0, CE1과 같이 CE핀에 번호를 붙인 후 이를 비트처럼 사용하여 LUN을 표현한다. 하나의 칩은 1~2개의 플레인으로 구성되어 있다. 플레인 역시 병렬 처리 단위이지만 LUN과 달리 병렬 동작에 제한이 있으며 다중 플레인 연산과 같이 처리율을 높이는 방식으로 동작할 수 있다. 각각의 플레인은 지우기 연산의 기본단위가 되는 블록으로 이루어져 있다. 블록의 개수는 칩의 종류에 따라 다르며 각각의 블록에는 고유의 번호가 부여되어 있고 블록 번호로 해당 블록이 어떤 플레인에 속해 있는지 알 수 있도록 규칙을 정한다. 예를 들어 짝수 블록은 플레인 #0 홀수 블록은 플레인 #1에 속하게 하는 방식 등을 사용한다. 각각의 블록은 쓰기 연산의 단위인 페이지로 구성되어 있다. MLC나 TLC의 경우 물리적으로는 하나인 메모리 소자 셀에 얼마만큼의 전자를 주입하는 가에 따라서 최대

3비트까지 표현할 수 있게 하는 방식을 사용하기 때문에 하나의 소자에서 사용할 수 있는 페이지를 정의해야 한다. 각각의 소자가 어떤 페이지를 사용할 수 있는지는 설계 방법에 따라 달라지며 One Shot 쓰기 연산 등의 소자에 속한 모든 페이지를 사용하는 연산을 사용하는 경우 접근해야 하는 페이지가 달라 진다. 읽기 쓰기 연산을 하기 위하여 원하는 페이지에 접근하거나 지우기 연산을 위해 원하는 블록에 접근하려면 주소가 필요하다. 실제로는 접근하고자 하는 LUN번호, 플레인 번호, 블록 번호와 함께 페이지 번호에 해당하는 열 주소 해당 페이지 안에서의 위치에 해당하는 행 주소가 필요하며 이러한 주소를 NAND 플래시 메모리에 인가하기 위한 규칙이 필요하다. NAND 플래시 메모리는 사용하는 핀의 개수를 줄이기 위하여 별도의 주소 핀을 사용하지 않고 ALE(Address Latch Enable)핀에 신호가 인가 되었을 때 I/O핀에 인가되는 신호를 주소로 변환하는 방식을 사용한다. 현재 NAND 플래시 메모리 제품들은 8개의 I/O핀을 사용하는 경우가 대부분이며 8개의 비트로는 필요한 모든 주소를 한번에 인가할 수 없어서 몇 번에 걸쳐서 정보를 나누어서 인가하게 된다.

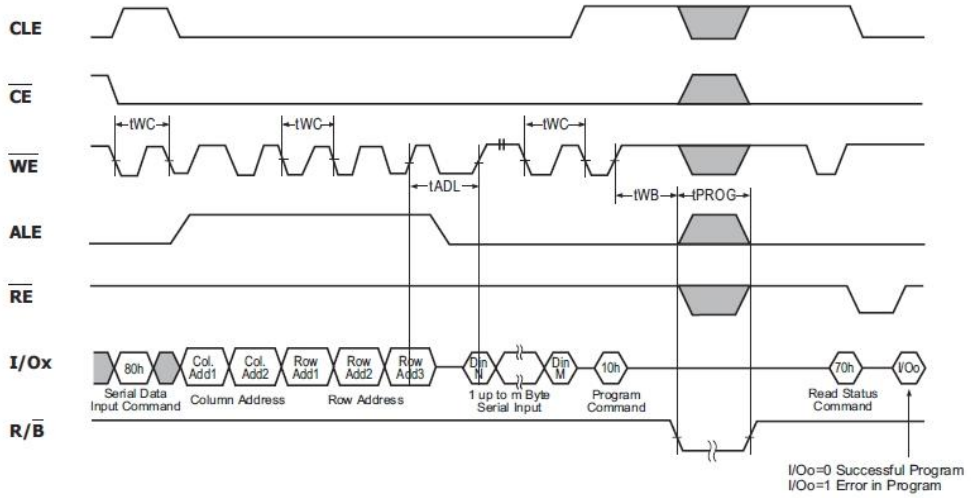

그림 2. ONFI를 따르는 쓰기 연산의 예. 행 주소를 2Cycle 열 주소는 3Cycle에 걸쳐서 인가한다 [11].

NAND 플래시 메모리는 블록당 일정 개수의 불량을 허용하지만 불량 한도가 넘어가는 경우에는 사용할 수 없는 블록으로 표시를 해야 하며 전체 사용할 수 없는 블록의 개수가 일정 개수 이하여야 하는 제한이 있다. 이를 지원하기 위해서는 불량 블록을 표시하기 위한 규칙도 필요하다.

메모리 구조는 2D를 지나서 3D로 발전하고 있고 제품의 용량과 한 페이지 당 메모리 셀의 개수가 비약적으로 커질 전망이다. 이러한 기술의 발전에 의한 구조적인 변화가 인터페이스의 변화로 이어 질 수 있다.

2.3.2.3 데이터 인터페이스

제품에 따라서 지원하는 데이터 송신 모드와 이를 지원하기 위한 규칙을 정의한다. 현재 상용제품에서 사용하고 있는 데이터 송신 모드는 SDR과 DDR 모드가 있다. SDR 모드에서는 DQS핀을 사용하지 않으며 DDR모드에서는 I/O와 동기 하는 데이터 신호로 이용한다. 모드에 따라서는 Differential Signal을사용하기도 한다. 세부 시간 요구 사항이나 세부 전압 요구 사항도 모드에 따라서 달라 질 수 있다.

Feature	Data Interface			
	SDR	NV-DDR	NV-DDR2	NV-DDR3
Protocol	Single data rate (SDR)	Double data rate (DDR)	Double data rate (DDR)	Double data rate (DDR)
Maximum Speed		200 MT/s	800 MT/s	800 MT/s
CE_n Pin Reduction support	Yes	Yes	Yes	Yes
Volume Addressing support	Yes	Yes	Yes	Yes
On-die termination support	No	No	Yes	Yes
Differential signaling	No	No	Yes, optional for DQS and/or RE_n	Yes, optional for DQS and/or RE_n
VccQ support	3.3 V or 1.8 V	3.3 V or 1.8 V	1.8 V	1.2 V
External Vpp support	Yes	Yes	Yes	Yes
External VREFQ support	No	No	Yes	Yes
ZQ Calibration	No	No	Yes, optional	Yes, optional
Previous name in older ONFI specifications	Asynchronous	Source S ynchronous	n/a	n/a

표 3. ONFI 데이터 인터페이스의 명세

또한 MLC TLC 제품의 경우 SLC 동작 모드를 지원하는 경우도 있다. 이 경우는 정상 동작과 SLC 동작의 경우 사용하는 핀과 시간 요구 사항 등이 전혀 달라게 된다.

데이터 인터페이스를 지원하기 위해서는 별도의 하드웨어 지원이

필요 할 수 있다. SDR과 달리 DDR 모드로 데이터를 주고 받기 위해서는 DQS에 동기 하여 데이터를 인식할 수 있는 하드웨어가 필요하며 SDR과 DDR모드 두 가지를 모두 지원하기 위해서는 이를 고려하여 제어기가 설계되어야 할 것이다. 만일 새로운 데이터 전송 모드를 지원하는 제품이 개발된다면 기존에 사용하고 있는 제어기로는 지원이 불가능할 수 있다.

2.3.2.4 연산 인터페이스

NAND 플래시 메모리 제품은 기본적으로 페이지 단위 쓰기 읽기와 블록 단위 지우기 연산을 제공한다. 이러한 기본 연산 이외에도 제품에 따라서 추가적인 연산을 제공하게 되는데 처리량을 증가시키기 위하여 두 개의 플레인을 동시에 접근하는 멀티 플레인 연산과 데이터 파이프라이닝을 지원하는 캐시(Cache)연산 등이 있으며, MLC와 TLC에서 동일 물리 소자의 모든 페이지를 접근하기 위한 One Shot 쓰기 연산과 더 나아가서는 지연시간이 긴 연산인 지우기 연산의 동작을 여러 단계로 나누어서 동시 동작성과 병렬 처리성을 높이기 위한 쓰기/지우기 연산의 동작 중 잠시 정지 동작을 지원하게 할 수 도 있다 [12]. One Shot 쓰기 연산은 SLC에서는 지원이 되지 않는 연산이며, 쓰기/지우기 연산의 잠시 정지 동작을 지원하려면 해당 연산을 지원할 수 있도록 제품을 설계하여야 한다.

현재 NAND 플래시 메모리는 표준 인터페이스를 따라서 발전하고 있지 않으며, 설계/소자/생산 기술이 빠르게 발전하고 있기 때문에 기존 제품에 없던 연산이 새로운 제품에 추가 되는 경우가 발생하고 있다. 이러한 연산은 기존의 데이터 인터페이스를 이용하는 방식으로 개발되는

것이 보통이고 이 경우 새로운 연산에 필요한 새로운 명령을 입력할 수 있도록 핀에 인가하는 신호와 시간 요구를 가변 할 수 있는 유연한 설계로 대응해야 할 것이다.

2.3.2.5 명령 및 타이밍 인터페이스

NAND 플래시 메모리가 원하는 연산 동작을 하게 만들기 위해서는 명령어를 인가해야 한다. CLE (Command Latch Enable)에 신호가 인가된 상태에서 I/O핀에 인가되는 신호는 NAND 플래시 메모리에 인가하는 명령으로 규정이 된다. 그러나 실제 쓰기 연산 등을 수행하기 위한 명령을 인가하기 위해서는 일련의 명령어 조합을 정해진 순서대로 인가한 후 주소를 인가하는 과정이 필요하며 그 이후 데이터를 송신하게 된다. 또한 이러한 일련의 과정에서는 어떤 핀에 신호를 인가해야 하는지에 대한 규칙과 다양한 시간적 요구를 맞추어야 하는 규칙이 규정되어 있다. 명령과 타이밍 요구 사항들은 제품에 따라서 조금씩 다르다.

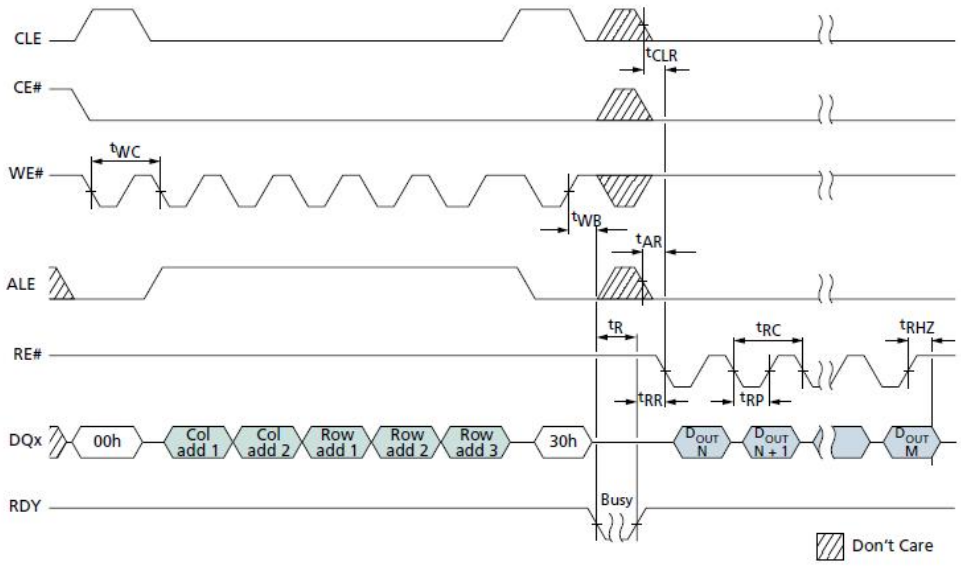

그림 3. ONFI를 따르는 읽기 연산의 명세 CLE가 인가 된 상태에서 00h를 I/O에 인가하는 것을 시작으로 열 주소, 행 주소 추가 명령 코드 등 일련의 신호들을 순서대로 인가해야 하며 tR, tRC 등은 과정에서 지켜야 하는 시간 요구이다.

제 3 장 프로그래밍 가능한 플래시 인터페이스

앞서 살펴 본 것처럼 NAND 플래시 인터페이스는 공통된 표준 인터페이스를 따르지 않고 있으며 기술 발전에 따른 필요에 의해 새로운 제품에는 기존과 조금씩 다른 인터페이스가 적용되는 형태로 발전하고 있기 때문에 일관성이 있는 인터페이스를 유지 하는 것이 어려워졌다. 이러한 상황에서는 보다 유연하게 변화 발전에 대응할 수 있는 방법이 필요하다.

MIPS 명령어체계처럼 간단한 명령어로 이루어진 마이크로 코드의 조합으로 프로그래밍 가능한 인터페이스를 만들면 현재의 다양한 인터페이스와 앞으로 나올 새로운 인터페이스에도 대응할 수 있을 것이다 [13]. 이러한 마이크로 코드 구조는 제조사나 칩의 종류에 관계없이 대응 가능한 범용성이 있어야 하고 제품들의 다양한 시간 요구 사항을 충족시킬 수 있을 만큼 유연하면서도 효율적으로 NAND 플래시 연산을 수행할 있어야 한다.

3.1 프로그램 가능한 플래시 인터페이스 정의

호스트는 NAND 플래시 제품이 어떤 제품인지 관계 없이 일관성 있는 인터페이스를 제공 받아야 한다. NAND 플래시 제품에 따라서 다른 연산을 제공한다거나 필요한 연산을 제공하지 않는 다면 이러한 일관성은 지켜지기 어렵다. NAND 플래시 메모리의 물리적 인터페이스

나 데이터 인터페이스가 바뀌지 않았다는 가정하에 새로운 연산을 지원하는 제품이 생산되었을 때 유연하게 설계된 제어기라면 기존의 연산의 연산코드나 시간 요구를 적절히 활용해서 대응할 수 있을 것이며 유연성이 없다면 기존 제어기로는 대응이 불가능할 것이다. 호스트 관점에서 보는 일관성과 유연함을 유지하기 위해서 해결해야 할 문제를 좀 더 일반화해서 정의해 보면 다음과 같다:

- 기술의 발달에 의하여 기존에 없던 새로운 연산이나 생기고 있다.
- 제품에 따라서 지원하는 연산의 종류가 다르다.
- 같은 연산 이라도 제품이 다르면 명령어와 시간요구 사항이 다를 수 있다.

이러한 문제들을 해결하기 위한 방법론으로 프로그래밍 가능한 인터페이스를 생각해 볼 수 있다. 프로그래밍 가능한 인터페이스는 잘 정의된 하부 요소들을 가지고 이를 조합하여 복잡한 신호 체계 구조를 만들어 낼 수 있는 인터페이스이며 이를 통해 다양하고 동적으로 변화하는 상황에서도 일관성을 유지하고 동시에 유연하게 대응할 수 있다. 본 논문에서는 프로그래밍 가능한 인터페이스를 NAND 플래시 메모리 제어기에서 구현했고 제어기의 동작을 정의하기 위한 마이크로 코드를 사용했다. 이 때 호스트에서 플래시 변환 계층을 거쳐서 제어기에 전달되는 인터페이스는 데이터 패킷을 사용하는 것을 가정했다. 프로그래밍 가능한 인터페이스를 구성하기 위해 필요한 마이크로 코드 연산의 요구를 생각해 보면 다음과 같다.

- 호스트에서 플래시 변환 계층을 거쳐서 제어기에 전달되는 패킷에서 주소와 명령어를 추출하기 위한 산술 작업
- 동일한 작업을 효과적으로 반복수행 할 수 있게 하기 위한 분기

원하는 시간에 신호를 주기 위한 동작 시간 제어

- 정해진 신호 생성

필요한 요구들을 수행하기 위해서는 명령어 체계 및 주소와 명령어를 계산하고 저장할 레지스터 구조에 대한 정의가 필요하다. NAND 플래시 메모리의 연산 중 Common Case에 해당하는 것은 데이터 전송이다. 워크로드에 따라서 차이가 있겠지만 NAND 플래시 메모리는 Idle하지 않은 시간 중 대략 95%의 시간을 데이터 송/수신에 사용한다 [14]. 마이크로 코드는 산술/논리 연산과 분기 연산 그리고 특수 연산으로 정의하였는데 특수 연산에 데이터 전송을 담당하는 연산이 속하며 다른 산술/논리 연산과 여타 특수 연산은 이러한 데이터 전송 연산을 효과적으로 하기 위한 보조 연산으로 정의할 수 있다.

마이크로 코드 연산을 프로그래밍해서 사용하기 위해서는 이를 저장할 메모리가 필요하며 현재 주소를 유지하거나 연산결과를 판단하여 새로운 주소로 이동하게 할 때 사용하기 위한 메모리 주소 레지스터가 필요하다. 또한 데이터를 저장하고 저장한 데이터를 읽기 위한 레지스터들이 필요하다. 이를 위하여 256행을 저장할 수 있는 메모리를 구현하여 메모리 주소를 통하여 접근할 수 있도록 했고 코드 수행에 필요한 32비트 레지스터를 32개 두었다. 모든 구현은 FPGA 환경에서 진행하였다.

앞서 살펴본 인터페이스의 여러 요소들 중 물리적 인터페이스의 변화나 데이터 인터페이스의 변화를 모두 프로그래밍 가능한 인터페이스로 대응하는 것에는 한계가 있다. 핀 구성이 바뀌거나 기존에 없던 전혀 다른 데이터 전송 방법을 적용하려면 하드웨어를 바꾸는 방법 외에는 방법이 없을 것이기 때문이다. 하지만 기존에 알려진 방식을

사용하는 경우라면 미리 준비된 하드웨어의 조합으로 이러한 물리적 인터페이스나 데이터 인터페이스도 프로그래밍 가능하게 구성할 수 있을 것이다. 물리적 인터페이스나 데이터 인터페이스는 즉 이미 알려진 방식을 조합하여 사용하는 것은 가능하지만 새로운 방식을 지원하기 어렵다. 그 외 메모리 구조 인터페이스나 연산 인터페이스 명령어 인터페이스 그리고 타이밍 인터페이스는 하부 요소들을 잘 정의한다면 프로그래밍 방식으로 이를 조합하여 새로운 인터페이스를 표현하는 것이 가능할 것이다.

3.1.1 마이크로 코드 포맷

마이크로 코드는 32비트의 데이터 형태로 프로그래밍 가능한 메모리에 저장 하며 이를 해석하고 실행하는 주체는 마이크로 코드 실행기이다. 기본적으로 산술/논리, 분기, 특수 연산과 신호 엔진에 명령을 주기 위한 신호 엔진 명령 연산의 4가지 형태로 정의 하였다.

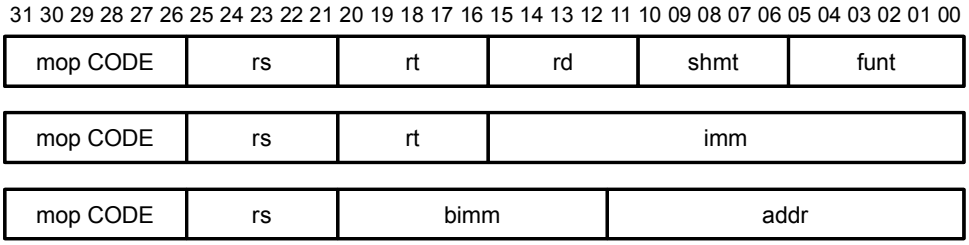

그림 4. 기본 마이크로 기본 코드 포맷. 위부터 순서대로 레지스터를 사용하는 연산과 레지스터와 직접 입력 값을 사용하는 연산, 그리고 주소를 이용하는 연산의 포맷이다.

이러한 기본 포맷 외에 8번째부터 11번째까지의 비트를 사용하는 엔진 명령어가 추가로 필요하다.

3.1.2 마이크로 코드 산술 논리 연산

산술/논리 연산은 패킷에서 명령어와 주소를 추출하거나 분기 연산의 분기가 이루어질 시기를 계산하기 위한 연산으로 3개의 레지스터에 저장된 값을 사용하는 연산과 두 개의 레지스터와 직접 입력된 숫자를 인식하여 계산하는 연산으로 구성했다.

연산 종류	연산자	사용 예	의미
산술/논리 연산	mop_add	mop_add r1,r2,r3	r3= r1+r2
	mop_addi	mop_addi r1,r2,10h	r2= r1+10h
	mop_sub	mop_sub r1,r2,r3	r3= r1-r3
	mop_subi	mop_subi r1,r2,10h	r2= r1-10h
	mop_and	mop_and r1,r2,r3	r3= r1 & r3
	mop_addi	mop_addi r1,r2,10h	r2= r1 & 10h
	mop_or	mop_or r1,r2,r3	r3= r1 | r3
	mop_ori	mop_ori r1,r2,10h	r2= r1 |10h
	mop_sll	mop_sll r1,r2,r3	r3= r1 << r3
	mop_slli	mop_slli r1,r2,10h	r2= r1 << 10h
	mop_slr	mop_slr r1,r2,r3	r3= r1 >> r3
	mop_slri	mop_slri r1,r2,10h	r2= r1 >> 10h

표 4. 프로그래밍 가능한 인터페이스의 명령어 체계 산술/논리 연산

3.1.3 마이크로 코드 분기 연산

분기 연산은 반복되는 작업을 효율적으로 표현하기 위한 연산으로 두 개의 레지스터에 저장된 값을 비교하여 직접 입력된 주소로 이동하거나 직접 입력한 주소로 이동할 수 있도록 구성하였다.

연산 종류	연산자	사용 예	의미
분기 연산	mop_beq	mop_beq r1,r2,10h	if(r1==r2) go to 10h
	mop_bne	mop_bne r1,r2,10h	if(r1 !=r2) go to 10h
	mop_blt	mop_blt r1,r2,10h	if(r1 < r2) go to 10h
	mop_bnlt	mop_bnlt r1,r2,10h	if(r1>=r2) go to 10h
	mop_beqi	mop_beqi r1,1h,10h	if(r1==1h) go to 10h
	mop_bnei	mop_bnei r1,1h,10h	if(r1==1h) go to 10h
	mop_blti	mop_blti r1,1h,10h	if(r1 < 1h) go to 10h
	mop_bnlti	mop_bnlti r1,1h,10h	if(r1>=1h) go to 10h
	mop_j	mop_j 10h	go to 10h

표 5. 프로그래밍 가능한 인터페이스의 명령어 체계 분기 연산

3.1.4 마이크로 코드 특수 연산

특수 연산은 NAND 플래시 인터페이스에서만 필요한 연산으로 데이터 Path를 직접 제어하는 연산과 레지스터를 거치지 않고 데이터나 명령어를 직접 NAND 플래시 메모리에 전달하는 연산의 두 가지로 나누어진다. 프로그래밍 가능한 제어기는 데이터 전송이 주된 Common Case이기 때문에 플래시 변환 계층과 패킷을 주고 받는 연산이 주

연산이 될 것이다. NAND 플래시 메모리에 필요한 명령어를 순차적으로 실행할 때에 시간 제약이 있어서 레지스터를 거쳐서 값을 내보낼 수 없는 경우가 발생할 수 있는데 이 경우에는 레지스터를 거치지 않고 바로 명령어나 주소를 전달하는 방법이 필요하여 직접 명령어/주소 인가 연산을 정의하였다. 또한 NAND플래시에 인가한 연산에 대한 응답 신호를 NAND에서 보내는 경우를 위하여 해당 신호를 응답으로 인식하고 저장할 수 있는 연산도 필요하다.

연산 종류	연산자	사용 예	의미
특수 연산	mop_ctrlsnd	mop_ctrlsnd	패킷을 받아서 레지스터에 저장
	mop_ctrlget	mop_ctrlget	레지스터의 값을 패킷으로 전송
	mop_cmdsnd	mop_cmdsnd r1	r1을 명령으로 보냄
	mop_cmdsndi	mop_cmdsndi 1h	1h를 명령으로 보냄
	mop_rdfwds	mop_rdfwds	Read Forward Enable
	mop_rdfwde	mop_rdfwde	Read Forward Disable
	mop_rdget	mop_rdget	NAND에서 받은 응답을 저장
	mop_wrfwds	mop_wrfwds	Write Forward Enable
	mop_wrfwde	mop_wrfwde	Write Forward Disable
	mop_wrsnd	mop_wrsnd r1	Write r1

표 6. 프로그래밍 가능한 인터페이스의 명령어 체계 특수 연산

3.1.5 마이크로 코드 신호 엔진 명령

실제 디자인 구성은 마이크로 코드를 해석하는 제어기인 마이크로 코드 실행기와와 NAND 플래시 메모리 인가할 실제 신호를 생성하는 신호 엔진으로 나뉘었기 때문에 마이크로 프로그램 변환기에서 신호 엔진에 인가하는 엔진 명령어가 또한 필요하다. 엔진 명령어는 마이크로

코드의 8번째부터 11번째까지의 비트를 사용하여 표현하게 하였다.

연산자	의미
engine_cmd_set_ce	CE 핀을 Set
engine_cmd_clr_ce	CE 핀을 Clear
engine_cmd_write_cmd	CLE핀에 신호 인가
engine_cmd_write_addr	ALE핀에 신호 인가
engine_cmd_sdr_wr	SDR 모드로 데이터 쓰기
engine_cmd_sdr_rd	SDR 모드로 데이터 읽기
engine_cmd_wait	단위 시간 동안 아무 동작 안 함
engine_cmd_ddr_wr	DDR 모드로 데이터 쓰기
engine_cmd_ddr_rd	DDR 모드로 데이터 읽기

표 7. 프로그래밍 가능한 인터페이스의 명령어 체계 신호 엔진 명령

3.2 제어기 디자인

프로그래밍 가능한 인터페이스를 구현할 제어기는 변환계층으로부터 스케줄러를 거쳐서 온 패킷을 받아서 주소와 명령어를 추출하는 NAND플래시 메모리 제어기인 마이크로 코드 실행기와 해석된 연산을 수행하기 위해 순차적으로 진행해야 하는 동작이 정의 된 명령어 메모리, 그리고 그것을 실제로 신호로 만들어 주는 신호 엔진의 세 부분으로 이루어 진다. 마이크로 코드 실행기가 동작하는 방식은 메모리를 참조하여 동작하는 CPU의 동작과 비슷한 부분이 있지만 CPU는 메모리에 쓰기 읽기를 모두 진행하는 것과 달리 마이크로 코드 실행기는 정의된 내용을 읽기만 한다는 점에서 다르다.

그림 5는 플래시 변환 계층에서 NAND 플래시 메모리 칩 사이에 있는 메모리 스케줄러와 제어기가 존재하는 형태를 간략하게 나타낸 그림이다. 실제 사용에서는 플래시 메모리 변환 계층에서 바로 패킷을 받는 것이 아니라 플래시 메모리 스케줄러가 존재하는 것이 보통이며 이러한 스케줄러와 제어기는 하나의 하드웨어로서 존재할 수도 있고 서로 다른 하드웨어로 구성 될 수도 있다. 본 논문에서는 기존 Finite State Machine으로 구현된 제어기를 프로그래밍 가능한 제어기로 대체하는 구조이기 때문에 메모리 스케줄러가 별도의 구조로서 구성된 경우를 상정하였다.

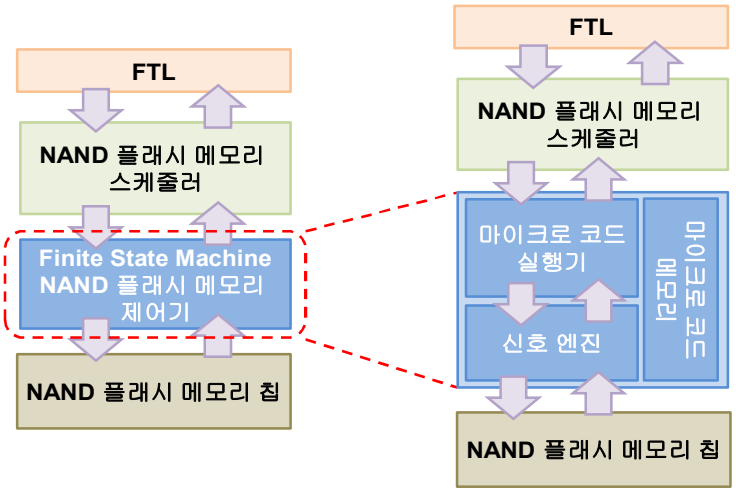

그림 5. 프로그래밍 가능한 제어기 구조. 기존 Finite State Machine으로 구현된 제어기 대신 마이크로 코드 실행기와 마이크로 코드 메모리 그리고 신호 엔진으로 구성된 프로그래밍 가능한 제어기로 대체 한다.

3.2.1 마이크로 코드 실행기

플래시 변환 플래시 변환 계층에서 보낸 패킷은 플래시 메모리 스케줄러를 거쳐서 마이크로 코드 실행기에 전달되고 마이크로 코드 실행기는 마이크로 코드 메모리를 순차적으로 읽으며 메모리에 저장된 코드를 따라 필요한 정보를 계산하거나 레지스터에 저장하고 엔진 명령어를 생성한다. 또한 필요한 경우 NAND 플래시 메모리에서 보낸 응답을 신호 엔진을 거쳐서 받아 이를 패킷으로 생성하여 플래시 메모리 스케줄러를 거쳐서 플래시 변환 계층에 전달한다.

3.2.2 마이크로 코드 메모리

사용자가 프로그램 가능한 마이크로 코드 메모리에는 플래시 변환 계층에서 스케줄러를 거쳐서 보낸 패킷에 명시된 요구를 수행하기 위해 어떤 코드 연산을 해야 하는지에 대한 명세가 정의되어 있다. 마이크로 코드 실행기가 접근하는 주소의 내용을 실행기에 전달하며 사용자가 미리 연산에 필요한 코드 명세를 적어 놓아야 한다.

3.2.3 신호 엔진

마이크로 프로그램 실행기에서 받은 명령을 통해 필요한 실제신호를

만들어 내고 필요한 경우 NAND 플래시 메모리의 응답을 마이크로 프로그램 실행기로 전달 한다. 마이크로 코드 실행기에서 직접 신호를 만들어내지 않고 엔진을 거치는 이유는 효율성 때문이며 엔진에 필요한 신호들을 만드는 명령어 체계가 잘 정의되어 있다면 마이크로 코드에서 생성하는 값과 엔진의 명령어의 조합으로 효율적인 제어가 가능하다.

3.2.4 제어기의 동작

이제 디자인 된 제어기의 동작을 실제 지우기 동작이 실행 되는 과정을 통하여 설명해 보겠다. 이러한 동작을 위해서는 프로그래밍 가능한 메모리에 지우기 동작의 명세가 순차적으로 정의되어 있어야만 한다. ONFI 인터페이스 기준으로 실제 지우기 연산이 실행되는 순서는 그림 6에 나와 있다.

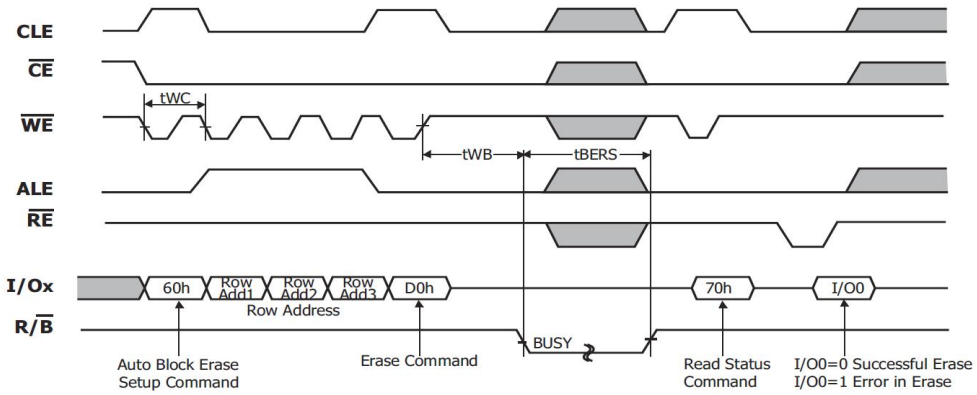

그림 6. ONFI를 따르는 지우기 연산의 예

이러한 동작에 필요한 명세는 마이크로 코드 메모리 50부터 저장되어 있다고 가정한다. 실제 메모리에 저장 되어 있는 내용은 실행되는 순서는 아래와 같다.

1) 레지스터를 모두 비우고 패킷을 받을 준비를 한다.

2) 플래시 변환 계층에서 보낸 패킷이 스케줄러를 거쳐서 전달 된다.

3) 마이크로 코드 실행기가 받아서 연산 정보를 추출하고 패킷은 레지스터에 저장을 한다.

4) 레지스터에 저장 된 연산자를 해독하여 지우기 연산임을 확인하고 분기 연산을 통해 지우기 연산이 정의된 프로그램 가능한 메모리의 첫 번째 주소(50)로 주소 프로그래밍 가능한 메모리에 접근하기 위해 마이크로 코드 메모리에 주소를 보낸다.

5) 해당 주소의 마이크로 코드를 받아서 실행한다.

: 주요 실행 내용은 주소 연산, 신호 인가, 분기 이다. 신호 인가 시에는 신호를 받을 해당 칩이 신호를 받을 준비가 되었는지 확인을 하며 준비 되지 않았다면 준비가 될 때까지 다음 주소의 명령을 실행하지 않고 기다린다.

6) 해당 메모리 주소의 코드의 수행이 끝나면 주소에 1을 더한 주소를 메모리에 인가한다. 단 분기 연산의 경우는 연산 결과가 되는 메모리의 주소를 인가한다.

이러한 방식으로 메모리에서 코드를 가져오고 준비 될 때까지 기다렸다가 코드를 실행하는 과정을 되풀이 하다 보면 지우기 연산에 필요한 마이크로 코드 연산과 엔진 동작이 끝날 것이다. 하나의 연산이 완료 된 다음에는 다음 패킷을 받을 준비를 하고 대기를 하게 된다.

3.3 소프트웨어로 구현된 공정 대기열 스케줄링

이제 프로그래밍 가능한 인터페이스를 활용하여 응용하는 예로서 소프트웨어로 공정 대기열 스케줄링을 구현해 보고자 한다. 이를 통하여 마이크로 코드 사용 제어기가 호스트에 어떤 편의성을 제공할 수 있는지도 함께 확인해 볼 것이다.

앞서 언급했듯이 스케줄러는 하드웨어로 구현할 수도 있고 소프트웨어로 구현할 수도 있다. 소프트웨어로 구현하는 경우 다양한 스케줄링 방법을 구현할 수 있는 유연성이 제공되는 대신 성능에서 다소 손해를 보게 되는데 잘 설계된 소프트웨어 스케줄러라면 하드웨어 대비 성능저하가 크지 않아야 한다. 먼저 소프트웨어로 무순서 중첩 스케줄링을 구현하고 이를 하드웨어로 구현한 스케줄러와 성능비교를 실시 하였다. 그리고 유연한 소프트웨어 스케줄링의 예로서 공정 대기열 스케줄링을 구현하였다.

공정 대기열의 스케줄링의 의의는 앞서 기술하였듯이 NAND 플래시 메모리 내부에 서로 다른 성격의 요청들간의 성능을 보장한다는 점에 있다. 이러한 서로 다른 요청들 중 대표적으로 예를 들 수 있는 것이 호스트의 요청과 쓰레기 수집의 관계이다. 앞서 언급하였듯이 NAND 플래시 메모리 동작의 Common Case는 데이터 전송이다. NAND 플래시 메모리 동작에서 읽기 보다는 쓰기가 쓰기 보다는 지우기가 지연시간이 길며 병렬 처리와 동시 동작 스케줄링을 잘 하여 지우기 연산과 쓰기 연산에서 발생하는 지연시간을 최대한 가려서 데이터 전송의 효율을

높이는 것이 스케줄링의 목표가 된다.

이때 데이터 전송을 얼마만큼의 크기로 해야 하는 가에 대한 결정이 성능에 영향을 주게 된다. 데이터 전송 요청은 크기가 큰 데이터를 순차적으로 전송하는 요청과 크기가 작은 요청을 임의로 전송하는 요청으로 나누어진다. 크기가 큰 요청을 중간에 중지 하는 일이 없이 진행하는 방식을 사용할 경우 크기가 작은 요청이 기다려야 하는 시간이 늘어나서 지연시간이 증가할 것이지만 작은 크기의 데이터 전송으로 나누어 실행한다면 도중에 작은 크기의 데이터 전송을 동시 처리를 이용하여 처리할 수 있기 때문에 지연시간이 작아지는 대신 전송률이 작아 질 것이다. 이러한 두 가지 종류의 요청을 적절히 처리하기 위하여 데이터 전송은 1KB 단위로 진행하게 하였다.

먼저 해야 할 일은 공정 대기열 구현을 위해 필요한 연산을 정의하고 이에 필요한 마이크로 코드를 프로그래밍 하는 일이다. 공정 대기열과 앞서 기술한 가장 좋은 처리율을 낼 수 있는 스케줄링 방법인 무순서 중첩 스케줄링 방법을 적용하기 위해서 NAND 플래시의 지우기 읽기 쓰기 연산에 대하여 추가적인 추상화 연산을 정의 하였는데 각각의 연산을 스케줄링을 효율적으로 할 수 있는 단위 연산으로 나누었다 [15].

지우기 연산은 지우기 연산과 상태 확인 연산의 두 부분으로 나누었다. 지우기 활성화 연산은 연산에 필요한 연산 코드와 칩과 블록 주소 등의 모든 정보를 담고 있고 상태 확인 연산은 상태 레지스터를 읽어서 지우기 연산의 완료 여부를 확인한다.

페이지 쓰기 연산은 4종류 연산들의 집합으로 나누었다. 쓰기 시작연산은 칩과 블록과 페이지 등의 정보를 담고 있고 쓰기 요청을 활성화 한다. 데이터 들여오기 연산은 실제 데이터를 전송하는 연산이고 쓰기 활성화 연산으로 마무리가 되는데 이때 비로서 페이지

레지스터에서 실제 셀에 데이터가 써지게 되며 지우기와 마찬가지로 쓰기 연산의 성공 여부를 상태 확인 요청 연산을 통해 확인한다.

페이지 읽기 연산은 3종류 연산의 집합으로 나누었다. 읽기 연산은 셀에서 페이지 레지스터 데이터를 이송하며 준비가 되면 데이터 내보내기 연산을 실행한다. 데이터 내보내기 연산의 완료를 상태 확인 연산을 통하여 확인한다.

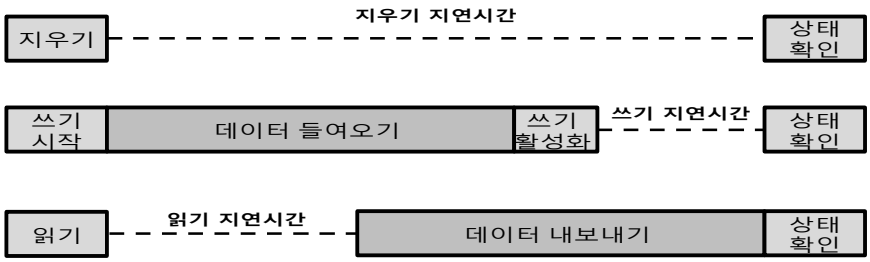

그림 7. 정의된 추상 연산 명세 위부터 지우기, 쓰기 읽기 연산이다.

무순서 중첩과 공정 대기열 스케줄링을 적용하기 위해 정의한 각 추상 연산들의 순차적 실행을 마이크로 코드로 프로그래밍했다. 즉 유저가 필요한 동작 단위를 정의하여 마이크로 코드의 조합으로 이를 나타낼 수 있는 것으로 마이크로 코드는 일관성과 유연성 외에 이러한 편의성을 제공할 수 있다.

기본적으로 무순서 중첩 스케줄링 방식을 사용하고 공정 대기열은 가상 시간을 기준으로 하는 방식을 채택하여 우선 순위를 계산하여 우선 순위가 높은 스트림의 연산이 먼저 실행 될 수 있도록 하였다.

제 4 장 실험 및 평가

프로그래밍 가능한 인터페이스를 적용하여 구현한 NAND 플래시 메모리 제어기의 비교 평가는 세 가지 측면에서 이루어졌다. 먼저 프로그래밍 가능한 인터페이스를 적용한 인터페이스와 기존의 Finite State Machine환경을 사용한 인터페이스로 구현한 제어기의 성능평가를 진행했고 소프트웨어로 구현한 스케줄러와 기존의 하드웨어 스케줄러와의 성능 비교 했으며, 마지막으로 스트림에 대해 QoS를 적용하지 않은 스케줄링과 공정 대기열 스케줄링의 결과를 비교했다.

마이크로 코드로 정의된 명령어체계는 지금까지의 NAND 플래시 메모리 연산을 모두 커버할 수 있는 완전성과 새로운 연산도 수행 가능한 호환성이 있어야 한다. 그러면서도 호스트에는 새로운 연산을 조합할 수 있는 편의성을 제공하여야 하는데 이러한 완전성과 호환성 그리고 편의성을 실험을 통하여 검증하였다.

4.1 실험 환경

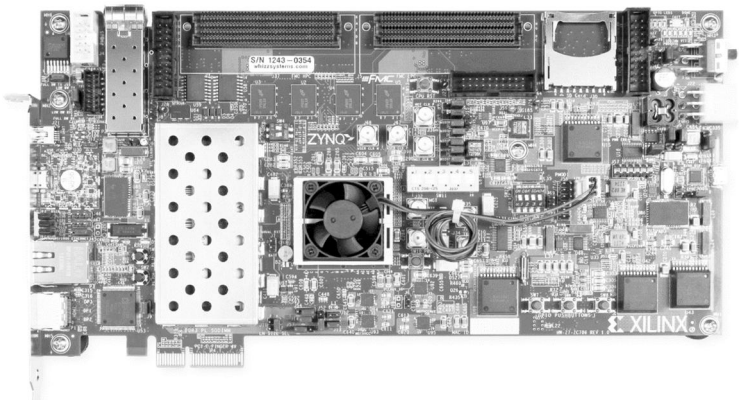

그림 8. Xilinx ZC706 보드

Xilinx사의 ZC706보드를 사용하여 NAND 플래시 메모리 제어기와 스케줄러를 구현하였다 [16]. ZC706보드는 ARM cortex A9 프로세서 2개와 함께 제공하는 FPGA 환경을 활용하여 워크로드 생성기와 하드웨어 스케줄러 그리고 소프트웨어 스케줄러를 구현했다. ZC706 보드에서 제공하는 커넥터를 이용하여 실제 NAND 플래시 메모리 제품에 연결하여 성능을 평가하였다.

실험에 사용한 NAND 플래시 메모리의 명세는 표8에 나와 있다.

항목	값
채널 #	1
채널당 칩#	4
칩당 블록#	4096
블록 당 페이지#	256
페이지 크기	16KB

항목	값
읽기 지연 시간	75us (typ) 110us (max)
쓰기 지연 시간	1.5ms (typ) 5ms (max)
지우기 지연 시간	5ms (typ) 10ms (max)
채널 bandwidth	100MB/s
총 용량	64GB

표 8. 실험에 사용한 NAND 플래시 메모리 명세

ZC706보드에서 제공하는 두 개의 코어 중 하나는 워크로드 생성을 전담하게 하고 다른 하나는 소프트웨어 스케줄을 전담하게 구현하였다. 워크로드 생성기는 합성 워크로드를 생성하며 미리 정해진 쓰기 읽기 지우기 비율로 안에서 하여 추상화된 요청을 생성하게 했다. 이 때 도착시간간격 시간을 최소부터 최대까지 시간을 지정하면 그 안에서 균등 난수로 생성하게 되고 생성된 요청은 동작 모드에 따라서 하드웨어로 구현된 스케줄러에 의해 스케줄링 되거나 소프트웨어로 구현한 공정 대기열 적용 스케줄러를 거쳐서 NAND 플래시 메모리에 인가된다. 모든 요청은 하드웨어로 구현된 기록기를 거치며 time stamp를 기록하게 되고 후에 DRAM에 기록된다. 역시 하드웨어로 구현된 데이터 생성기에서 랜덤 데이터를 생성하게 하였다. 이러한 구조에 Finite State Machine으로 구현된 플래시 메모리 컨트롤러와 프로그래밍 가능한 마이크로 코드를 사용한 플래시 메모리 컨트롤러를 따로 구현하여서 성능을 비교하였다.

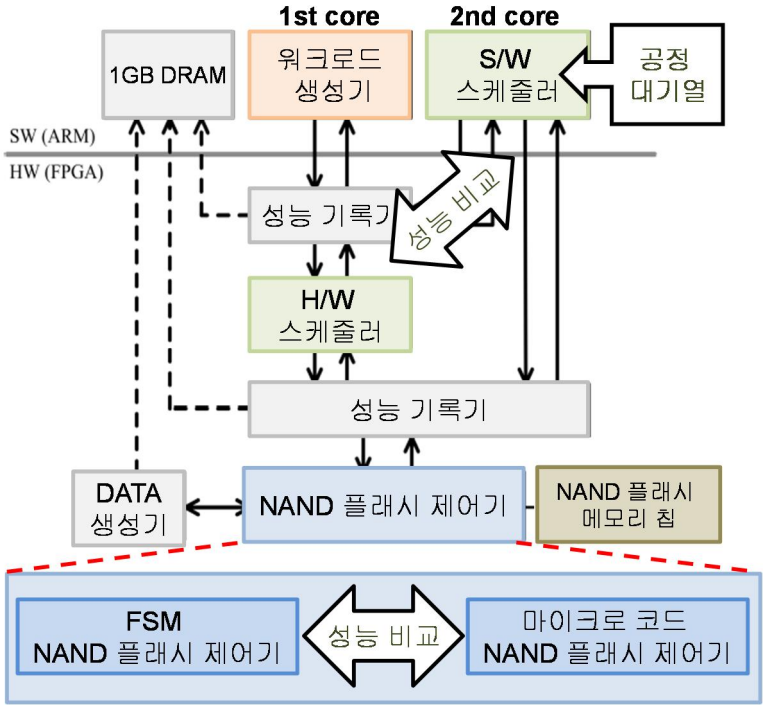

그림 9. 실험에 사용한 구조

4.2 실험 평가

실험을 통하여 확인해야 할 내용은 먼저 마이크로 코드 제어기의 동작 정합성과 성능과 마이크로 코드 프로그램을 통해 편의를 얻은 S/W 스케줄러의 동작 정합성과 성능 그리고 공정 대기열 스케줄링의 정합성 세 가지이다.

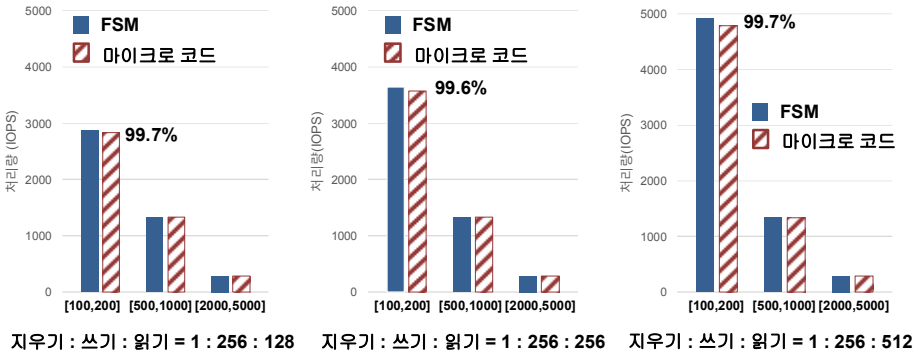

그림 10. 마이크로 코드 제어기와 FSM 성능 비교. 대괄호 안의 숫자는 균등 난수로 생성되는 워크로드의 생성 속도의 상한과 하한이며 지우기 쓰기 읽기의 비율을 다르게 하여 세가지 조건에서 비교를 진행하였다.

마이크로 코드를 적용한 제어기와 FSM을 사용한 제어기의 성능을 비교한 결과 최악의 경우 99.6%로 거의 차이가 없다. FSM을 최적화 구현 진행한 상태가 아니라서 단순 비교만 할 수 있지만 위의 결과로 보아 마이크로 코드로 프로그래밍 방식의 성능이 크게 떨어지지 않는 것으로 판단할 수 있다.

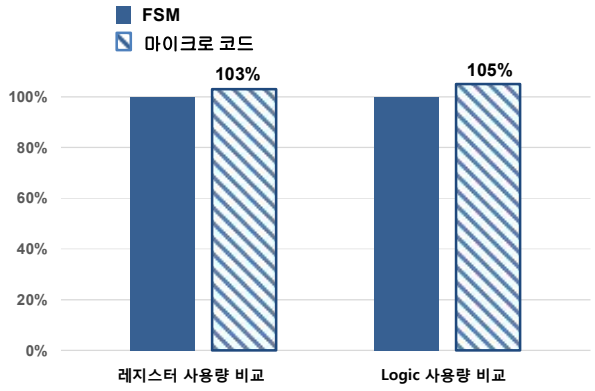

그림 11. 마이크로 코드 제어기와 FSM 성능 자원 소모 비교

FPGA환경에서의 자원 사용량 비교한 결과 Finite State Machine 대비 레지스터 사용량은 103% Logic 사용량은 105%로 마이크로 코드를 사용한 경우가 자원소모량이 많다. Finite State Machine으로 설계한 제어기는 설계 최적화로 자원 소모량을 최소화한 것은 아니기 때문에 최적화를 진행할 경우 더 소모량이 적어질 수 있다.

일반적인 경우 마이크로 코드를 실행하기 위한 메모리와 추가 로직들이 필요한 마이크로 코드 실행 환경이 자원 소모가 많으며 최적화가 가능한 Finite State Machine 환경이 자원소모가 적다.

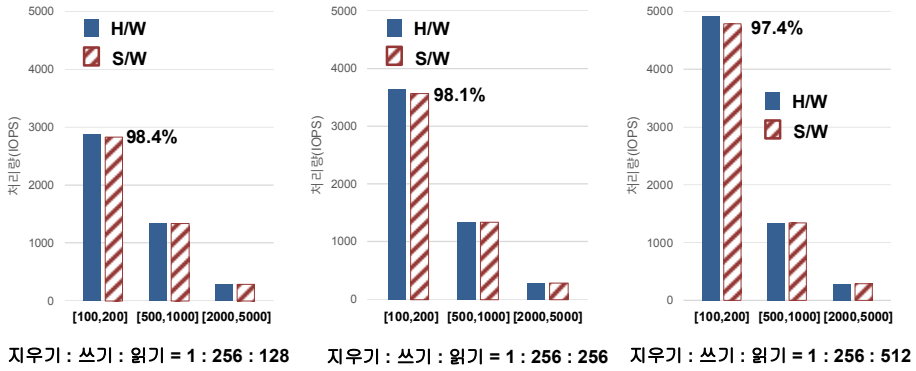

그림 12. 하드웨어 스케줄러와 소프트웨어 스케줄러 성능 비교

소프트웨어 스케줄러와 하드웨어 스케줄러의 성능을 처리율로 비교한 결과 최악의 경우 97.4% 차이가 발생하였다. 이러한 성능 차이는 하드웨어 스케줄러는 소프트웨어 스케줄러 대비하여 NAND 플래시 칩에 빨리 접근 할 수 있고 소프트웨어 스케줄러는 NAND 플래시 칩에서 온 신호를 바로 처리 할 수 없기 때문에 발생한다.

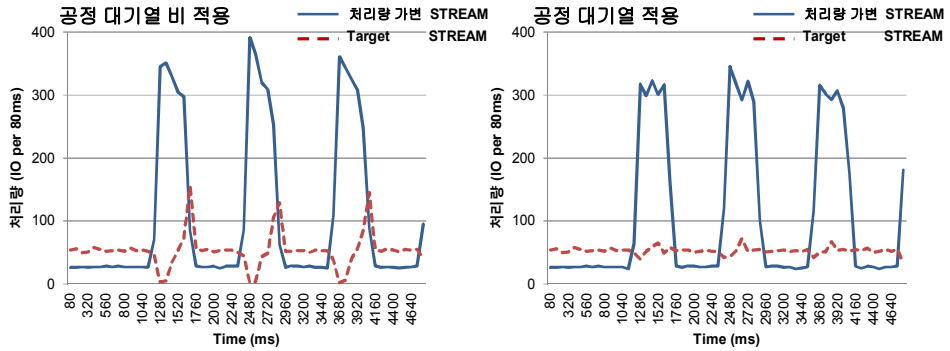

그림 13. 공정 대기열 성능 비교

공정 대기열은 처리량이 가변되는 스트림의 처리가 대상 스트림의 처리에 영향을 미치지 않도록 하기 위하여 사용한다. 공정 대기열을 사용하지 않은 경우 처리량이 변하는 스트림에 의하여 대상 스트림의 처리량이 확연하게 영향을 받는 것을 볼 수 있다. 그러나 공정 대기열을 사용한 경우에는 처리량이 변하여도 대상 스트림의 처리량이 일정하게 유지되는 것을 확인할 수 있다.

제 5 장 결론

본 논문에서는 통일 된 인터페이스가 없이 빨리 기술이 발전한 결과 제품마다 인터페이스가 제 각각인 NAND 플래시 메모리에 프로그래밍 가능한 마이크로 코드를 사용하는 제어기를 적용하였고 그 결과 유연하고 일관성이 있는 인터페이스를 제공하면서도 동시에 작업에 편의성을 더 할 수 있다는 것을 실험을 통하여 확인하였다.

마이크로 코드 프로그래밍은 아직은 효율적이지 못한 부분이 많이 있으며 이를 개선하기 위한 연구가 더 진행 될 필요가 있다. 파이프라인 구조를 이용하여 효율을 높이거나 현재 구현된 Vertical구조에서 하나의 명령어가 많은 연산을 병렬로 수행할 수 있는 Horizontal구조로 더 발전시키는 방법 등을 생각해 볼 수 있다.

참고 문헌

[1]skhynix.com, ‘Products: Raw NAND,’ 2017, [Online], Available:https://www.skhynix.com/eng/product/nandRaw.jsp

[2]Jeong, Jaeyong, et al. "Lifetime improvement of NAND flash-based storage systems using dynamic program and erase scaling." FAST. 2014.

[3]C. Trinh et al., “A 5.6MB/s 64Gb 4b/Cell NAND flash memory in 43nmCMOS,” in IEEE Solid-State Circuits Conference, 2009,

[4]L. M. Grupp et al., “Characterizing flash memory: Anomalies,observations, and applications,” in IEEE/ACM Microarchitecture, 2009,

[5]A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair queueing algorithm," in SIGCOMM, 1989,

[6] E.H. Nam, B.S. Kim, H.Eom, and S.L. Min, "Ozone (O3): An Out-of-Order Flash Memory Controller Architecture," in IEEE Trans. On Computers, vol. 60, no. 5, pp. 653-666, May 2011

[7] S. Jamaloddin Golestani, "A Self-Clocked Fair Queueing Scheme for Broadband Applications," in INFOCOM'94, 1994

[8] B.S. Kim and S.L. Min, “QoS-aware flash memory controller,” in IEEE RTAS, 2017

[9] S.S Hahn, S. Lee, and J. Kim, “SOS: software-based out-of-order scheduling for high-performance NAND flash-based SSDs,” in IEEE MSST, 2013,

[10] ONFI [Online], Available:http://www.onfi.org/specifications

[11]micron.com, ‘Products: NAND’ [Online], Available: https://prod.micron.com/~/media/documents/products/data-sheet/nand-flash/die/l95b_die_128gb_nand.pdf

[12] Guanying Wu and Xubin He, “Reducing SSD Read Latency via NAND Flash Program and Erase Suspension” in FAST 2012

[13]R. Razdan and M.D. Smith, “A high-performance microarchitecture

[14] Cagdas Dirik, “Performance Analysis of NAND Flash Memory Solid-State Disks”, 2009,

[15] B.S. Kim and S.L. Min, “Framework for Efficient and Flexible Scheduling of Flash Memory Operations,” in NVSMA 2017

[16] Xilinx.com, “Xilinx Zynq-7000 All programmable SoC ZC706 Evaluation Kit”, 2017, [Online], Available: https://www.xilinx.com/products/boards-and-kits/ek-z7-zc706-g.html with hardware-programmable functional

Abstract

NAND flash memory is a non-volatile memory in which data is retained even when power is not applied. It is rapidly emerging as a storage medium replacing existing hard disks and occupies the storage media market. As NAND flash products get their spotlight, technology advances faster and new product launches faster, but each NAND flash memory product has slightly different interface because there is no uniform standard interface. In such an environment, the flexibility of the NAND flash memory controller will become a problem, and the controller developed over a long period of time can only be used for a short time.

In this thesis, we define a programmable flash memory interface as a method to provide consistency in a situation where different interfaces are used and implement it as a NAND flash memory controller using FPGA environment. The programmable interface can flexibly respond to NAND flash memory products with different interfaces and provide consistency to the host. We also implemented quality-of-service (QoS) -based process queue scheduling as an application environment to take advantage of this flexible and consistent interface to verify its performance and effectiveness.

Keywords: NAND flash memory based storage system, NAND flash memory controller, NAND flash memory operation scheduling,

Fair queueing

Student Number: 2015-22910