모델체킹 도구를 사용한 플래시 메모리 디바이스 드라이버 검증

김문주, 김윤호
KAIST 전자전산학과
moonzoo@cs.kaist.ac.kr
kimyunho@kaist.ac.kr

김호태
삼성전자 기술총괄 소프트웨어연구소
hotae.kim@samsung.com

요약: 플래시 메모리는 휴대폰, 디지털 카메라, mp3 플레이어와 같은 모바일 기기의 핵심 부품으로 많이 사용되고 있다. 사용자에게 성공적으로 서비스를 제공하려면 디바이스 드라이버 소프트웨어가 모바일 기기 내의 플래시 메모리를 정확하게 제어해야 한다. 그러나 기존 테스트 방법으로는 복잡한 디바이스 드라이버 소프트웨어의 숨겨진 결함을 발견할 수 없는 경우가 종종 생긴다. 이런 약점 때문에 소프트웨어의 개발, 관리 비용이 크게 증가한다.

기존의 테스트 방법의 약점을 보완하고 플래시 디바이스 드라이버 소프트웨어의 오류를 검출하기 위해 본 연구는 다양한 모델체킹 도구를 사용하여 플래시 디바이스 드라이버 소프트웨어를 검증했다. 본 연구의 목표는 정형검증기법들 중 모델체킹이 소프트웨어 업계에서 디버깅 방법으로서 유효성을 알아보는 것이다. 그러므로 본 연구에서는 이미 개발된 모델체킹 도구를 검증 대상의 특성에 맞게 사용해서 소프트웨어를 검증했다. 본 연구의 결과로 기존에 발견되지 않은 여러 소프트웨어 오류를 검출하였다.

핵심어: 정형검증, 모델체킹 도구, 내장형 소프트웨어

1. 서론

플래시 메모리는 적은 전원 소모와 물리적 충격에 강한 특성 때문에 모바일 기기에 대중적으로 사용되고있다. 사용자에게 성공적으로 서비스를 제공하려면 디바이스 드라이버 소프트웨어가 모바일 기기 내의 플래시 메모리를 정확하게 제어해야 한다. 그러나 기존 테스트 방법으로는 복잡한 디바이스 드라이버 소프트웨어의 숨겨진 결함을 발견할 수 없는 경우가 종종 생긴다. 이런 약점 때문에 소프트웨어의 개발, 관리 비용이 크게 증가한다. 예를 들어 삼성전자는 플래시 디바이스 드라이버 소프트웨어를 개발하는 것보다 소프트웨어를 검사하는 데 더 많은 시간과 자원을 할애하고 있다. 특히 소프트웨어의 신뢰성과 성능을 보장하는 것은 높은 수준의 소프트웨어 품질을 보장하는 데 있어 가장 어려운 일이다.

모바일 기기 분야에서의 기존 테스트 방법은 몇 가지 제약 사항이 있다. 첫째, 테스트는 예상하지 못한 오류를 찾아내는 것에 있어서 효과적이지 않을 수 있다. 사람이 만드는 테스트 케이스는 제작자가 예상한 상황만 검사하고, 무작위 테스트 방법은 모든 가능한 경우를 검사할 수 없기 때문이다. 둘째, 기존 테스트 방법을 사용해서 멀티 쓰레드 내장형 소프트웨어를 검사하기 어렵다. 내장형 기기에서는 스케줄러를 사용자가 임의로 제어할 수 없기 때문에 멀티 쓰레드 프로그램을 제어하여 특정 테스트 케이스를 수행하도록 만들기 어렵고, 같은 이유로 오류가 발견되었을 때 오류를 재현하기 어렵다. 마지막으로 내장형 기기를 완벽히 제어할 수 없는 문제가 있다. 내장형 소프트웨어는 플래시 메모리의 블랑 블록을 사용하지 않게 하는 것과 같이 여러 가지 하드웨어 결함을 처리한다. 이러한 결함을 처리하는 함수를 테스트하려면 인위적으로 하드웨어 결함을 만들 수 있을 정도로 하드웨어를 미세하게 제어할 수 있어야 하나 현실적으로 어려운 일이다.

이 제약 사항은 삼성전자의 OneNAND™ 플래시 메모리를 개발하는 과정에서 명백히 드러난다[2]. 예를 들어, 플래시 메모리의 데이터 읽는 속도를 향상시키기 위해 여러 개의 섹터를 한꺼번에 읽는 함수가 플래시 디바이스 드라이버에 추가했다. 그러나 광범위한 테스트와 디버깅 노력에도 불구하고 이 함수에 많은 오류가 남아있었고 개발자들은 이 기능을 제거하는 것을 심각하게 고려했다. 따라서 OneNAND 디바이스 드라이버의 기존에 알려지지 않은 오류를 찾기 위해서 본 논문은 다양한 모델체킹 도구(Blast[6], Spin[9], NuSMV[4])를 사용하였다. 본 논문에서는 정형검증기법 중 모델 검증 기법이 소프트웨어 업계에서 실질적이고 효과적인 디버깅 방법으로 사용될 수 있는 가를 알아본다.

2. OneNAND FLASH 의 개요

이 장에서는 플래시와 OneNAND 플래시 메모리 드라이버의 특성에 대해서 설명한다.

2.1 플래시 메모리의 특성

플래시 메모리는 NAND 플래시 메모리와 NOR 플래시 메모리가 있다[1]. NAND 플래시는 집적도가 높기 때문에 일반적으로 저장 매체로 사용된다. NOR 플래시는 NAND 플래시와 달리 소프트웨어를 플래시 메모리 상에서 수행시킬 수 있기 때문에(XIP: execution in place) 일반적으로 소프트웨어의 바이너리 파일을 저장한다. NAND 플래시는 페이지의 집합으로 구성되며 페이지는 섹터의 집합으로 구성된다. 또한 페이지들을 더 큰 단위로 묶은 것은 블록이라고 한다. NAND 플래시의 기본 작업(operation) 은 한 페이지를 읽고 쓰는 것과 한 블록을 지우는 것이다. NAND 플래시 페이지를 읽는 데는 횟수 제한이 없으나 NAND 플래시 블록을 지우는 횟수에는 한계가 있다.

플래시 파일 시스템은 하나의 논리 페이지를 물리 페이지의 연결 리스트에 사상(mapping)시킨다. 플래시 메모리에 새로운 데이터를 저장하는 경우 기존 데이터를 갖고 있는 물리 섹터에 바로 겹쳐 쓰지 않고 빈 물리 섹터에 데이터를 저장한다. 기존 데이터를 갖고 있는 물리 섹터는 invalid 로 표시한다. 따라서 플래시 파일 시스템은 논리 섹터에서 물리 섹터로의 사상을 관리해야 하고, invalid 물리 섹터에 대한 쓰레기 수집(garbage collection)을 수행해야 한다.

2.2 OneNAND 플래시 메모리를 위한 디바이스 드라이버 소프트웨어의 개요

OneNAND 플래시는 NOR 플래시 인터페이스, NAND 플래시 제어 로직, NAND 플래시 배열 그리고 내부 RAM 으로 구성된 단일 칩이다. OneNAND 플래시는 내부 RAM 을 사용해서 NOR 인터페이스를 제공한다. OneNAND 에 저장된 응용 프로그램을 수행할 때 XIP 기능을 제공하기 위해 Demanding Page Manager(DPM) 가 프로그램이 저장된 페이지를 RAM 으로 불러온다.

Unified Storage Platform(USP)는 [그림 1]과 같이 OneNAND 를 기반으로 하는 모바일 내장형 시스템을 위한 소프트웨어 솔루션으로, 프로그램 저장소와 데이터 저장소를 관리한다. 응용 프로그램은 USP 의 파일 시스템을 통해 데이터를 저장하고 검색할 수 있다. USP 는 Flash Translation Layer(FTL) 을 통해 OneNAND 플래시에 저장된 프로그램과 데이터에 접근한다.

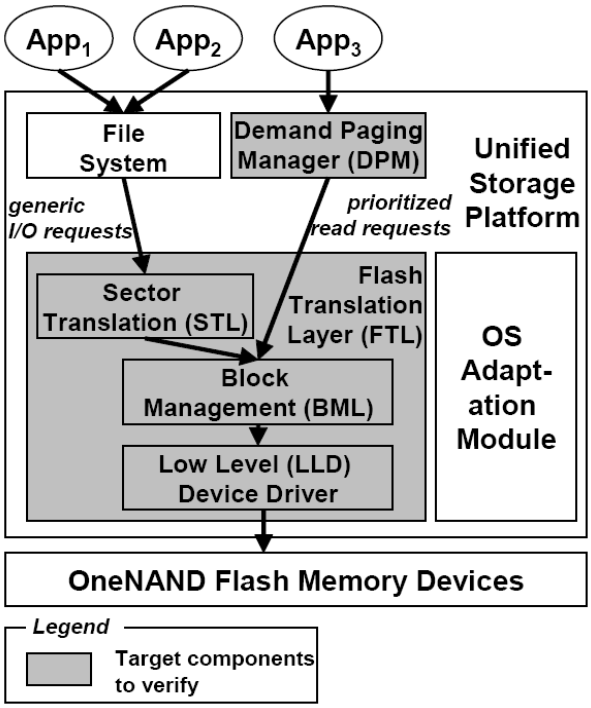

그림 1 OneNAND 플래시의 USP 개요

FTL 은 Sector Translation Layer(STL), Block Management Layer (BML)와 Low-level Device Driver (LLD)로 구성된다. 응용 프로그램의 범용 입출력 요청은 파일 시스템, STL, BML, LLD 의 순서를 거쳐 처리하며, 프로그램을 수행하면서 발생하는 우선 순위가 높은 프로그램 실행 요청의 경우 DPM 을 통해 직접 BML 을 거쳐 수행한다. 이 검증 프로젝트에서는 USP 의 FTL 과 DPM 의 구성요소를 분석하였다.

3. 프로젝트의 개요

3.1 프로젝트의 범위

본 프로젝트는 교수와 대학원생 그리고 삼성전자의 선임 연구원 각각 한 명이 팀을 이뤄 총 육 개월간 수행하였다. 처음 삼 개월은 USP 의 디자인과 코드를 검토하였다. USP 의 디자인은 총 259 페이지에 달하는 문서로 정리되어 있다. 대부분의 USP 코드는 C 언어로 작성되어 있고, 일부분은 ARM 어셈블리언어로 작성되어 있다. FTL 과 DPM 의 소스 코드는 대략 30,000 줄 정도다. 연구의 주안점이 소프트웨어 산업 환경에서 정형검증기법이 효과적인가를 알아보는 것이기 때문에 기존에 널리 사용되는 모델체킹 도구를 사용해서 새로운 디버깅 방법을 도입하는데 드는 부하를 최소화 했다.

3.2 검증 요구사항

USP 는 FTL 과 DPM 의 디자인 문서와 소프트웨어 요구 명세 문서를 가지고 있다. 이 문서는 13 개의 기능성 요구사항(functional requirement)을 명세하고 있다. 각 기능성 요구사항은 각각의 우선순위를 명시하고 있다. 다음은 가장 높은 우선순위의 기능성 요구사항이다.

- 동시성 처리
  USP 에는 두 가지 종류의 동시성 작업이 있다. 첫 번째는 범용 입출력 작업 사이에서의 동시성이고, 둘째는 범용 입출력 작업들과 우선순위가 높은 읽기 작업 사이에서의 동시성이다. USP 는 이 두 가지 종류의 동시성을 정확히 처리해야 한다. 즉, 세마포어(semaphore) 또는 락(lock)과 같은 동기화 기법을 사용하여 경쟁 조건(race condition) 또는 교착 상태(deadlock)를 피해야 한다.

- 우선 순위 높은 프로그램 실행 작업 제공
  요구 페이징(demand paging)을 통해서 프로그램을 수행하려면 페이지 폴트 예외가 발생할 때 DPM 이 코드 페이지를 내부 RA 으로 불러와야 한다. 이때 폴트 대기 시간을 최소화해야 하므로 FTL 은 파일 시스템의 범용 입출력 요청 보다 DPM 에서 발생하는 읽기 요청을 우선하여 처리해야 한다. 우선 순위가 높은 프로그램 실행 작업 요청은 범용 입출력 요청을 중단하고 먼저 실행될 수 있으며 중단된 범용 입출력 작업은 나중에 계속 진행된다.

- 섹터 관리
  파일 시스템은 플래시 메모리가 연속된 논리 섹터들로 구성되어있다고 가정한다. FTL 은 논리 섹터로부터 물리 섹터로의 사상을 제공한다. 즉, 다수의 논리 섹터는 분산된 물리 섹터에 사상되며, 분산된 물리 섹터를 정확한 순서로 읽을 수 있어야 한다. 그 외에 FTL 은 불량 플래시 블록을 관리할 수 있어야 하며 플래시 블록의 마모 정도를 제어할 수 있어야 한다.

본 논문은 하향식 접근 방법으로 FTL 과 DPM 코드가 앞서 제시한 세 가지 요구사항을 만족하는 지 분석하였다. FTL 과 DPM 코드를 분석하기 앞서 디자인 문서와 USP 코드를 검토하는 과정에서 디자인 문서와 C 코드 사이에서 다수의 불일치를 발견하였으며, 모든 불일치는 프로그램 코드에서의 문제가 아닌 디자인 문서에서의 오류로 밝혀졌다.

4. 검증 결과

본 장에서는 3.2 장에서 언급한 세가지 요구 사항 명세의 검증 결과를 설명한다.

4.1 동시성 처리

USP 는 STL 을 통해서 동시에 여러 입출력 요청을 처리하지만, BML 은 BML 범용 작업이 수행되는 동안 다른 BML 범용 작업을 수행하지 않는다. 즉, BML 작업은 항상 순차적으로 처리된다. BML 작업을 순차적으로 처리하기 위해서 BML 은 이진 세마포어(semaphore)를 사용하여 STL 의 동시 입출력 요청을 조정한다. 이진 세마포어의 대표적인 요구사항은 세마포어를 얻은(acquire) 모든 작업은 세마포어를 해제(release)해야 한다는 것이다. 본 연구에서는 Blast[6]를 사용하여 이 요구사항을 분석하였다. Blast 는 명시적인 모델을 만들지 않고 C 코드를 직접 분석할 수 있다. 본 연구에서는 정수형 변수 smp 를 추가하여 세마포어의 상태를 나타내고, smp 값을 증가시키고 감소시키는 코드를 USP 의 C 코드 내에 세마포어 작업에 대응되는 삽입하여 다음 두 가지 특성을 검사하였다.

- 모든 세마포어 작업에서 $0 \le \mathrm{smp} \le 1$ 을 만족해야 한다.
- 세마포어를 사용하는 함수가 반환되는 시점에서 $\mathrm{smp} = 1$ 이 되어야 한다.

Blast 를 사용하여 두 가지 특성을 USP 가 만족하는 지 검사한 결과 첫 번째 특성은 만족하였지만 두 번째 특성은 만족하지 않았다. 세마포어를 사용한 14 개의 함수 중에서 13 개의 함수가 두 번째 특성을 위반하였다. 세마포어 해제 작업이 실패할 경우, 즉 에러 코드를 반환하는 경우 세마포어를 사용한 함수는 즉시 STL 로 BML_RELEASE_SM_ERROR 에러를 반환한다. 대부분의 STL 함수들은 이 에러 플래그를 받으면 즉시 STL_CRITICAL_ERROR 을 반환하여 사용자 어플리케이션이 이 문제를 처리할 수 있도록 한다. 그러나 63 개의 STL 함수 중 5 개의 함수는 BML 에러를 받았을 때 STL_CRITICAL_ERROR 를 보내지 않았으며, 이는 잠재적으로 심각한 문제들을 만들어낼 수 있다.

4.2 우선 순위가 높은 프로그램 실행 작업

USP 는 현재 수행되고 있는 범용 입출력 작업을 중단하고 우선 순위가 높은 프로그램 실행 작업을 수행할 수 있다. 따라서 USP 는 중단된 범용 입출력 작업의 상태를 저장하고, 우선 순위가 높은 작업이 수행되고 난 다음에 다시 작업이 수행될 수 있도록 중단된 작업의 상태를 복구할 수 있어야 한다. 이러한 상태 저장 및 복구 기능은 우선 순위가 높은 작업을 처리하기 위한 함수에 구현되어 있다.

본 연구에서는 중단된 범용 작업의 상태가 저장되고 그것이 정확하게 복구되는 지 검증하였다. 이 검증을 위해서 위의 기능을 수행하는 두 개의 함수를 Spin[9] 모델체킹 도구의 입력 언어인 Promela 모델링 하였다. C 함수에서 나타나는 모든 제어 구조를 대응되는 Promela 제어 구조로 모델링 하고 범용 입출력 요청을 하는 프로세스들에 대해서는 추상화하여 모델링 하였다[1]. 생성된 모델은 총 250 줄로 범용 입출력 요청을 무작위로 호출하는 프로세스와 우선순위가 높은 프로그램 실행 작업을 요청하는 DPM 프로세스로 구성되어 있다.

Spin 모델체킹 도구를 사용하여 모델을 검증한 결과 교착상태가 발생하지 않았다. 하지만 범용 삭제 작업이 수행되고 있는 중에 우선 순위가 높은 작업을 수행할 때 삭제 작업의 현재 상태를 저장하지 않았다. USP 의 원 개발자들은 OneNAND 하드웨어의 특성상 이 오류가 실제로 문제가 되지 않지만 논리적 결함이라는 것을 확인했다.

4.3 멀티 섹터 읽기 작업

USP 는 가능한 한 많은 섹터를 한번에 읽을 수 있는 메커니즘을 제공한다. 이 메커니즘은 하나의 함수(MSR())로 작성되어 있다. 복잡한 자료구조를 사용해서 논리 섹터를 물리 섹터로 사상하기 때문에 MSR()는 7 개의 중첩된 반복문을 갖는 복잡한 함수로 작성되었다. 본 연구는 이 함수가 멀티 섹터를 정확하게 읽는 지 검사하였다. 즉, 이 함수가 데이터를 다 읽었을 때 읽기 버퍼의 내용이 플래시 메모리의 내용과 같아야 한다.

MSR() 함수를 Spin 과 NuSMV[4] 두개의 모델체킹 도구를 사용해서 각각 모델을 만들고 검증했다. MSR() 모델은 무작위로 물리 섹터에 데이터를 쓰고 논리 섹터와 물리 섹터의 사상에 따라 데이터를 읽어 읽기 버퍼에 기록한다. MSR()을 모델링 할 때 다음 두 모델링 전략을 사용했다.

- 모든 제어 구조를 모델링 하지만 플래시 자료구조는 최소화 해서 모델링 한다. 최소화 한 플래시 자료구조는 여섯 개의 물리 페이지를 가지고, 하나의 페이지가 네 개의 섹터를 가지고 하나의 섹터는 1byte 크기를 갖는다. 또한 데이터는 총 6 개의 섹터에 걸쳐 쓰여진다고 가정한다. 이런 설정에서 그림 2 와 같이 6 개의 논리적 섹터에서 물리적인 섹터로의 분배 방법이 대략 1 억 개 존재한다.

- Spin 과 NuSMV 에는 포인터 형이 존재하지 않기 때문에 포인터는 배열의 인덱스로 교체한다.

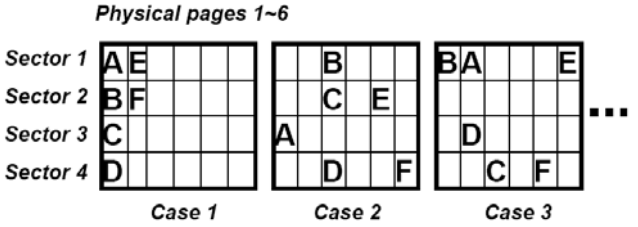

그림 2 논리적 섹터로부터 물리적 섹터로의 가능한 분배 방법

Spin 의 입력 언어인 Promela 는 제어 지향적 특성을 갖기 때문에 조건문(if), 반복문(do) 복잡한 자료형 재정의(typedef) 배열 등을 사용해서 C 프로그램을 직접적으로 모델링할 수 있다. 그러나 NuSMV 는 데이터 지향적이므로 각 클럭마다 데이터가 동시에 갱신되는 NuSMV 모델로 C 프로그램을 변환해야 한다. 또한 NuSMV 는 배열의 인덱스 변수를 제공하지 않기 때문에 Spin 의 모델은 약 200 줄이지만 NuSMV 의 모델은 약 1300 줄이다.

검증 성능을 살펴보면 Spin 은 45 초동안 347MB 의 메모리를 사용하여 모델이 주어진 요구사항을 만족함을 보였다.[2] 하지만 NuSMV 는 24 시간을 넘어도 검증 결과가 나오지 않았다. 따라서 5 개의 물리 섹터와 5 개의 논리 섹터로 모델을 줄여서 검증을 수행하였다. 이 작은 모델에 대해서 cone of influence 와 dynamic reordering of variables 의 옵션을 사용했을 때 NuSMV 는 11 시간동안 274MB 의 메모리를 사용하여 요구사항을 만족함을 보였다. 같은 모델에 대해서 Spin 은 3 초 동안 23MB 메모리를 사용하여 검증 결과를 보여줬다.

비록 작은 모델에 대해서만 멀티 섹터 읽기 작업이 제대로 동작함을 검증하였지만, 모델체킹 도구가 모델에서 가능한 모든 상태를 검사해서 멀티 섹터 읽기 작업이 요구사항을 만족함을 증명해 줬기 때문에 멀티 섹터 읽기 작업이 올바르다고 추정할 수 있다. 대부분의 OneNAND 소프트웨어 테스트는 파일 시스템 단계에서 수행되기 때문에 테스트를 통해 논리 섹터와 물리 섹터의 사상을 제어하거나 불량 블록 관리 기능을 제어할 수 없다. 따라서 기존 테스트 방법은 FTL 에서 발생하는 오류를 찾지 못할 수 있다. 그러므로 작은 모델에서의 모델체킹을 통한 검증은 파일 시스템에서의 수 많은 테스트보다 MSR()의 정확성에 대한 신뢰성을 제공할 수 있다.

5. 토의

[1] Blast 에서는 멀티 쓰레드 프로그램을 분석할 수 없으므로 Blast 를 사용할 수 없었다.

[2] Fedora core 5 가 설치된 Intel Core2Duo E6400 (2.13GHz)와 4GB 메모리를 가진 PC 에서 Spin 4.2.7 과 NuSMV 2.4.1 을 사용하여 검증을 수행하였다.

5.1 디버깅 방법으로서의 모델 검증 기법의 효과

정형검증기법 중 특히 모델체킹은 포착하기 어려운 결함을 탐지해서 컴퓨터 시스템의 신뢰성을 높이는 도구로 사용되었다. 하지만 상태 폭발 문제와, 검증 대상 프로그램과 대응되는 정형 모델 사이의 차이 때문에 모델 검증 기법은 소프트웨어 개발자들로부터 디버깅 방법으로 사용되기 적합하지 않다고 인식됐다. 그러나 C 프로그램을 모델체킹 하기 위한 활발한 연구가 있어왔고, 연구 활동에 대한 결과가 다양한 연구 집단에서 주목 받고 있다[5,10]. 이 프로젝트를 통하여 모델체킹 도구가 프로그램의 작지만 매우 중요한 부분, 특히 전통적인 테스트 방법을 사용하였을 때 발견하기 어려운 오류들에 대해서 디버깅 도구로 사용될 수 있음을 보였다.

5.2 직접 코드 검증의 이익

이 프로젝트에서 대부분의 시간이 할애된 일은 멀티 섹터 읽기를 위한 NuSMV 모델을 생성하는 것이었다. 이와 반대로 Blast 를 사용하여 동시성의 정확성을 검증하는 일이 가장 쉬웠다. 이는 NuSMV 모델을 생성하는 과정에서 사람이 직접 모델을 작성해야 하며, 이 과정에서 에러를 발생시킬 가능성이 있었기 때문이며, Blast 의 경우 C 프로그램을 거의 변경하지 않고 직접 검증할 수 있었기 때문이다. 소프트웨어 산업체에서 모델체킹을 활용하기 위해 C 코드를 직접 분석할 수 있는 능력이 필요하며, 그에 따라 소프트웨어 모델 검증 기법이 심도 있게 연구되어야 한다.

5.3 적합한 검증 프레임워크 선정의 중요성

모델링 난이도, 검증 성능, 생성한 모델의 관리비용은 검증 도구에 따라 크게 다르기 때문에 검증 도구의 선택은 검증 프로젝트에 많은 영향을 미친다. Spin 은 C 코드의 작은 부분을 검증할 때 매우 유용하였다. 일반적으로 명시적 모델체킹 도구가 심볼릭 모델체킹 도구 보다 성능이 떨어진다고 알려진 바와 달리 큰 자료구조와 결정적(deterministic) 행동을 보이는 대부분의 C 프로그램을 검증하는데 매우 뛰어난 성능을 보인다.

6. 결론

본 연구에서는 널리 활용되고 있는 모델체킹 도구를 사용하여 성공적으로 OneNAND 플래시 메모리 디바이스 드라이버의 버그를 찾았다. 이 오류들은 기존에 탐지되지 않았던, 세마포어를 제대로 처리하지 않은 문제와 삭제 작업이 우선 순위가 높은 작업을 하기 위해 중단될 때 현재 삭제 작업 상태를 저장하지 않는 논리적 결함들이다. 그리고 모든 크기의 플래시 자료구조를 검증한 것은 아니지만 매우 복잡한 멀티 섹터 읽기 함수에 대하여 함수가 올바르다고 확신할 수 있다.

현재의 모델 검증 기술은 safety-critical 시스템의 C 코드 전체를 검증할 정도의 확장성은 없으나[3], 작은 핵심 부분을 디버깅 하기 위한 보완책으로 활용할 수 있다. 삼성전자는 이 검증 결과를 높이 평가했고 다음 프로젝트로 플래시 파일 시스템의 전원이 무작위로 끊어졌을 때의 데이터 일관성을 검사할 것을 계획하고 있다. 플래시 메모리가 모바일 기기에 광범위하게 사용되고 있고 모바일 기기의 전원이 예상치 않게 꺼지는 것은 자주 일어나는 일이기 때문에 이 때 데이터 일관성이 유지되는지 검사하는 것이 중요하다.

[1] A collection of NAND flash application notes, whitepapers. Available at http://www.datio.com/NAND/NANDApplicationNotes.asp

[2] Samsung OneNAND fusion memory. http://www.samsung.com/global/business/semiconductor/products/fusionmemory/Products_OneNAND.html

[3] A. Groce, G. Holzmann, and R. Joshi, “Randomized differential testing as a prelude to formal verification,” The International Conference on Software Engineering, May 2007

[4] A. Cimatti, E.M. Clarke, E. Giunchiglia, F.Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, A. Tacchella, “NuSMV 2: An opensource tool for symbolic model checking,” International Conference on Computer-Aided Verification, 2002

[5] C. Kilian, J.W. Anderson, R. Jhala, A. Vahdat, “Life, death, and the critical transition: Finding liveness bugs in systems code,” USENIX symposium on Networked Systems Design and Implementation, 2007

[6] D. Beyer, T.A. Henzinger, R. Jhala, R. Majumdar, “The software model checker blast: Applications to software engineering,” International Journal on Software Tools for Technology Transfer, 2007

[7] E.M. Clarke, O. Grumberg, D.A. Peled, *Model Checking. MIT Press*, January 2000.

[8] E.M. Clarke, O. Grumberg, S.Jha, Y. Lu, H. Veith, “Progress on the state explosion problem in model checking,” Lecture Notes in Computer Science 2000, 2001

[9] G.J. Holzmann. *The Spin Model Checker. Wiley*, New York, 2003

[10] J. Yang, P. Twohey, D. Engler, M. Musuvathi, “Using model checking to find serious file system errors,” Operating System Design and Implementation, 2004