소프트웨어 Instrumentation 기법을 활용한 빠른 임베디드 시스템 검증 기법

김창무°, 송지석, 김지홍
서울대학교 컴퓨터공학부
{kyuirs, danielsong, jihong}@davinci.snu.ac.kr

A Fast System-Level Verification Technique Using Software Instrumentation

Changmoo Kim°, Jiseok Song, Jihong Kim
School of Computer Science and Engineering, Seoul National University

요 약

다양한 분야에 임베디드 시스템이 사용되면서 여러 가지 요구 조건에 부응하는 맞춤형 임베디드 시스템을 주어진 시간 내에 개발하는 것이 중요해졌다. 개발 기간을 맞추기 위해 하드웨어와 소프트웨어의 개발이 동시에 이루어지면서 실제 하드웨어가 완전히 갖춰지기 전에 소프트웨어를 개발하는 경우도 많아지고 있다. 하지만 이를 위한 시스템 시뮬레이터의 편의성과 성능이 좋지 않아 다른 대안이 절실하게 필요한 실정이다. 이 문제를 해결하기 위해서, 소프트웨어 Instrumentation 도구를 이용하여 소프트웨어의 수행 중에 추가적인 코드를 삽입함으로써, 주변 장치의 역할을 해주는 장치 시뮬레이터를 개발하였다. 별다른 장비 없이도 사용이 가능하며 장치 드라이버와 같은 API를 제공함으로써 편의성을 증대시켰고 동작 속도도 기존의 시스템 시뮬레이터보다 20배 빠르다. 간단한 임베디드 시스템을 구성하여 실험해본 결과, 시뮬레이터의 편의성과 빠른 성능을 확인할 수 있었으며 소프트웨어의 동작 검증을 편리하게 할 수 있었다. 앞으로 임베디드 소프트웨어의 개발에 유용하게 쓰여 개발 기간을 단축시킬 수 있을 것이다.

1. 서 론

반도체 집적도가 증가하면서 소형화를 거듭해온 임베디드 시스템은 가정에서 사용하는 간단한 전자 제품에서부터 자동차, 비행기 등 대형 시스템까지 널리 사용되고 있으며 앞으로 더욱 다양한 용도로 사용될 것이다. 이렇게 임베디드 시스템의 역할은 다양하지만 대부분의 경우에 입력 장치에서 받은 데이터를 분석해 판단을 하고 그에 따라 다른 장치에 명령을 내리는 일이 많으므로 주변 장치를 제어하는 것은 필수적인 역할이다. 그래서 임베디드 시스템을 구동하는 소프트웨어가 주변 장치들과 제대로 동작하는지 검증하는 것이 매우 중요하다. 간단한 역할을 수행하는 경우에는 사용자의 불편을 초래하는 데에 그치겠지만 중요한 시스템의 경우 오동작은 치명적인 사고를 유발할 수도 있기 때문이다.

그런데 임베디드 시스템의 소프트웨어를 개발하는 데에 가장 어렵고 시간이 많이 소요되는 부분이 이 검증 단계이다. 소프트웨어의 검증이 원래 어려운 일이기도 하지만 특히 임베디드 소프트웨어의 경우 개발 환경이 다양하여 검증이 더욱 어렵고 심지어 개발 단계에서 하드웨어가 완전히 갖춰지지 않은 경우도 있다. 소프트웨어와 하드웨어의 개발이 병렬로 진행되면 전체 개발기간을 줄일 수 있는 장점이 있기 때문이다. 그러나 이렇게 검증하기 어렵다는 점이 소프트웨어 개발자들을 힘들게 하고 있으며 임베디드 시스템이 널리 사용되는 데에 장애가 되고 있다.

이 문제를 해결하기 위해 다양한 도구가 사용되고 있지만 아직 만족스러운 해결책은 마련되지 않은 실정이다. 각종 디버거들이 사용되고 있지만 개발 환경에 영향을 많이 받고 동작도 느린 편이다. 또 주변 장치가 갖춰지지 않은 상황에서 소프트웨어 개발을 시작해야 하는 경우에 주변 장치 역할을 대신할 수도 없다. 그래서 입출력 에뮬레이션 기능을 갖춘 시스템 수준의 하드웨어 시뮬레이터가 사용되기도 한다. 하지만 동작 속도가 매우 느리고 사용 방법이 간단하지 않다는 단점이 있다.

그래서 본 연구에서는 이를 해결하기 위해 주변 장치

역할을 대신하는 시뮬레이터를 개발하였다. 검증할 임베디드 소프트웨어의 실행 가능한 이진 코드를 가상 머신 위에서 수행하면서 코드 중간에 추가적인 코드를 삽입하는 Instrumentation 도구를 이용한다. 이를 이용한 검증 기법의 장점은 아래와 같다.

- 동작 속도: 시스템 수준의 하드웨어 시뮬레이터에 비해 월등하게 빠른 속도로 동작한다. 이는 멀티미디어 응용과 같이 계산이 많은 프로그램도 납득할만한 시간 내에 검증이 가능하다는 것이다.
- 편의성: 일반적인 장치 드라이버와 같은 API를 제공하므로 실제 장치를 사용하는 소프트웨어에서 장치 드라이버를 교체하는 것만으로 바로 검증할 수 있다.
- 정확성: 최종적인 실행 가능한 이진 코드에 추가적인 코드를 삽입하는 것이기 때문에 기존 코드의 동작과 Instruction 단위 수준에서 동일하다.
- 비용 절감: 별다른 장비나 소프트웨어의 설치가 필요하지 않고 사용한 소프트웨어 Instrumentation 도구가 무료로 제공되고 있어 개발 비용이 절감된다.

다음 장에서는 사용한 Instrumentation 도구에 대해 소개한다. 시뮬레이터의 구현에 대해 자세히 설명한 부분이 3장이고 구현된 시뮬레이터를 사용한 결과가 4장에서 소개된다. 마지막으로 5장에서 결론을 내린다.

2. 소프트웨어 Instrumentation 도구

소프트웨어 Instrumentation은 소프트웨어의 동작을 관찰하기 위해서 추가적인 코드를 삽입하는 것을 말한다. 추가적인 코드의 삽입 시기에 따라 분류가 가능한데 소스 코드에 삽입할 수도 있고 컴파일할 때나 링크한 후, 또는 수행 중에 삽입하기도 한다. 이 중에서 수행 중에 동적으로 Instrumentation이 가능하도록 인텔에서 개발한 PIN을 사용하였다. PIN은 Xscale, IA-32, Intel64, Itanium 프로세서용 이진 코드를 Instrumentation할 수 있고 리눅스와 윈도우 환경을 지원한다[1].

PIN을 이용하여 다양한 연구가 수행되었는데 대표적인 사례를 소개한다. 먼저 소프트웨어에 존재하는 반복 루틴의 반복 횟수와 겹친 루프의 깊이를 분석해 병렬 프로그밍을 도와주는 연구가 있었다[2]. Strube는 프로그램의 수행 단계를 나누어 수행 시간을 분석하여 해당 머신의 특징을 알아내고 성능을 예측하였다[3]. Clause는 실제 사용자가 소프트웨어를 사용하는 상황에서 문제가 생겼을 때 이에 대한 정보를 저장하고 개발자가 재연하여 문제를 해결하도록 해주는 기법을 개발하였다[4].

그림 1은 PIN의 소프트웨어 구조를 보여주고 있다. 굵은 선으로 둘러싸인 부분이 PIN이고 아래에 기반이 되

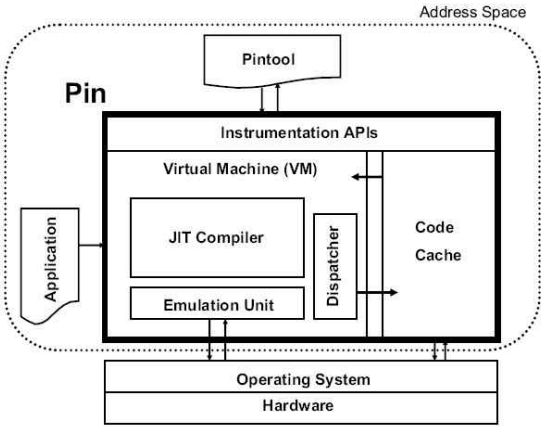

그림 1. PIN의 소프트웨어 구조[1]

는 운영체제와 하드웨어가 있다. Application은 PIN이 Instrumentation을 하게 될 대상 프로그램으로 실행 파일 형태로 입력된다. Pintool은 사용자가 작성하는 부분으로 Instrumentation API를 이용하여 원하는 대로 코드를 삽입하도록 해준다. JIT(Just In Time) Compiler가 가장 중요한 역할을 하는 부분으로 Pintool의 코드와 대상 프로그램의 이진 코드를 합친다. 합쳐진 코드는 Code Cache에 저장되었다가 가상 머신에 의해 수행된다.

이런 원리로 대상 프로그램과 Pintool이 동시에 수행되면서 독립적으로 동작할 수 있는 것이다. Pintool은 대상 프로그램의 메모리에 접근이 가능하기 때문에 마치 통신을 하는 것처럼 데이터를 보내고 받을 수 있다. 이 성질을 이용하여 장치 시뮬레이터를 구현하게 되는 것이다.

3. 장치 시뮬레이터의 구현

그림 1에서 Application의 자리에 검증할 대상 프로그램이 들어가고 Pintool이 장치의 역할을 담당하게 된다. 그림 2를 보면 대상 프로그램이 수행되다가 일정 주기마다 Pintool이 수행되는 동작 흐름을 볼 수 있다.

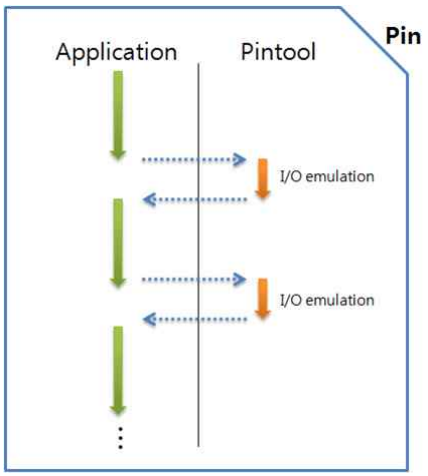

그림 2. 장치 시뮬레이터의 동작 흐름도

3.1 데이터 영역 확보

Pintool과 대상 프로그램이 통신을 하는데 이용할 메모리가 필요하기 때문에 대상 프로그램의 앞부분에 적당한 크기의 전역 변수를 선언한다. 여기에 장치들에 대한 정보와 통신을 위한 버퍼들이 구조체 형태로 위치하게 된다. 그러므로 필요한 크기는 사용할 장치의 개수와 종류에 의해 결정된다. 또한 Pintool이 대상 프로그램과 동기화하여 시작하기 위한 플래그 변수를 이 영역에 만들어 두고 시작할 때에 메모리를 초기화하며 플래그를 특정한 값으로 설정한다.

3.2 입출력 버퍼

Pintool과 대상 프로그램의 통신에서 데이터가 임시 저장될 공간을 제공하고 데이터가 소실되는 것을 방지하기 위해 버퍼를 만들었다. 입출력 각각에 대해 버퍼에 저장된 데이터 개수를 저장하는 변수와 데이터를 위한 공간으로 구성된 버퍼를 만들었고 안정성을 위해 두 개까지 데이터를 저장할 수 있는 더블 버퍼를 선택하였다. 그래서 기본 버퍼 크기를 1KB로 정하면 장치마다 4KB의 메모리가 필요하고 장치별 정보에 버퍼의 시작주소와 그 크기를 저장해둔다.

3.3 장치 드라이버 API

일반적인 장치 드라이버와 같이 장치를 처음 사용할 때 호출하는 Open 함수와 사용을 모두 마치고 호출하는 Close 함수, 그리고 장치로부터 데이터를 읽거나 장치에 데이터를 쓰기 위한 Read/Write 함수가 제공된다. 대부분의 경우 이런 방식으로 장치를 사용하므로 검증 대상 프로그램에서 이 함수들만 교체해주면 검증이 가능하다. 장치를 사용하기 위해서는 장치별 ID를 알아야 하는데 이는 new_device 함수를 통해 장치를 추가할 때에 얻거나 장치 이름을 인수로 get_deviceID 함수를 호출하여 얻을 수 있다. 장치를 사용하기 전에 이런 장치 관리 함수들을 이용하여 원하는 장치들을 구성하면 된다. 표 1에 함수들이 정리되어 있다.

장치 관리	get_deviceID, list_device new_device, destory_device
장치 사용	open, close, read, write

표 1. 장치 드라이버 API 목록

3.4 검증 시나리오 및 로그 생성

여러 가지 장치를 구성하였지만 실제로 동작을 검증하기 위해서는 입력 장치들에 적절한 입력을 주고 임베디드 시스템이 그에 따라 동작하는지 확인해야 할 것이다. 입력 장치의 수가 많아지면 각각의 장치에 적절한 입력을 주는 것도 복잡할 수 있다. 이를 위해 시나리오 파일의 형식을 정의하고 이에 따라 장치에 입력을 주는 시점과 입력할 데이터를 파일에 기록하게 하였다. 그러면 Pintool이 시나리오 파일을 읽으면서 동작 검증을 수행하고 그 결과를 로그 파일을 생성하여 기록한다.

3.5 검증 도구

장치가 많고 복잡한 시스템의 경우 입력하는 시나리오와 출력된 로그를 분석하여 동작이 주어진 시간 안에 정확하게 이루어졌는지 확인하는 것도 쉽지 않을 것이다. 검증에 필요한 조건들을 주어진 형식에 맞게 작성하면 조건에 맞게 동작하였는지 자동으로 확인해주는 도구도 만들었다. 예를 들어 커피 자동판매기의 경우 동전을 넣었을 때 투입동전의 액수를 제대로 표시하는지와 그 소요 시간을 조건으로 정할 수 있을 것이다.

4 결과

4.1 실험 설정

실제 하드웨어로 구현된 주변 장치들과 동일한 동작을 하는지 여부와 검증에 필요한 시간, 동작 속도를 실험으로 비교하고자 커피 자동판매기 시스템을 구축하였다. 주변 장치와 검증할 임베디드 시스템으로 나누어 두 대의 컴퓨터가 각각의 역할을 수행하였고 LAN으로 연결하여 UDP패킷을 이용해 통신하도록 구성하였다.

주변 장치의 역할을 하는 컴퓨터는 임베디드 시스템이 필요로 하는 데이터를 해당 장치의 방식에 따라 전송하거나 명령을 받아 동작하도록 구성하였다. 커피 자판기의 버튼 입력, 투입된 금액, 커피 재료의 잔량, 보유 잔돈의 수량 등의 데이터가 각각의 형식으로 임베디드 시스템에 전송된다.

임베디드 시스템 역할을 하는 컴퓨터는 주변 장치로부터 받은 정보에 따라 각종 램프의 점등 여부, 커피 제조 장치의 동작, 잔돈 반환 등의 명령을 내린다. 이를 구현하기 위해 무한 루프를 돌면서 주변 장치로부터 들어오는 데이터를 확인하고 그에 해당하는 동작을 하도록 프로그램을 작성하였다. 장치 드라이버를 구성하여 이를 통해 데이터를 읽고 쓰도록 하였고 장치 드라이버가 UDP 패킷을 처리하게 된다.

4.2 실험 결과

커피 자동판매기 시스템 검증에 소요된 시간이 표 2에 나타나있다. 앞에서 설명한 것처럼 실제 하드웨어를 만들지 않고 컴퓨터로 하드웨어 역할을 수행하도록 구성하는 것도 4시간이 소요되었다. 실제 하드웨어로 주변장치

를 만드는 것은 더 많은 시간이 필요할 것이다. 이에 비해 장치 시뮬레이터는 소프트웨어로 구현되었으므로 별다른 장치나 환경 구성이 필요하지 않았고 동기화도 알아서 해주기 때문에 간단히 API를 사용하는 것으로 장치들을 대신할 수 있었다. 결과적으로 검증 환경의 구축에 소요된 시간이 10배 이상 단축되었고, 검증 시나리오와 검증 조건을 입력하면 자동으로 동작하고 편리하게 반복해볼 수 있어 검증 시간도 줄어들었다. 그림 3은 커피 자동판매기에 대해 검증 도구가 8가지 조건으로 검증한 결과를 보여주는 화면이다.

	하드웨어 구성	장치 시뮬레이터
환경 구축	4시간	20분
검증 시간	1시간	40분

표 2. 검증에 소요된 시간 비교

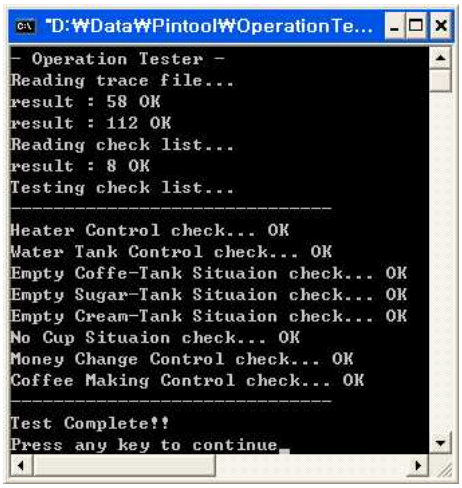

그림 3. 동작 검증 결과 화면

	기본 시스템	장치 시뮬레이터
동작 속도	3,000 MIPS	125 MIPS

표 3. 동작 속도 비교

표 3을 보면 측정한 동작 속도를 알 수 있다. Core 2 Duo 1.8Ghz의 CPU를 탑재한 환경에서 실험한 것으로 기본 시스템은 이 환경을 의미한다. 장치 시뮬레이터의 동작 속도는 수행된 Instruction의 수를 세는 PIN의 기능을 사용하여 측정한 것이다. 기본 시스템에 비해 장치 시뮬레이터가 24배 느린 속도로 동작하는 것으로 나왔지만 125 MIPS는 기존의 시스템 시뮬레이터에 비해 월등하게 빠른 속도이다. 입출력 에뮬레이션 기능을 갖춘 시뮬레이터인 SimpleScalar의 경우, 시뮬레이션 수준에 따라 다르지만 간단한 기능의 시뮬레이션에서 6 MIPS 정도의 동작 속도를 보인다[5]. 결과적으로 장치 시뮬레이터가 20배 정도 빠른 속도로 동작하는 것이다.

5 결론

본 연구에서는 임베디드 시스템을 검증하기 위해 소프트웨어 Instrumentation 도구를 이용한 장치 시뮬레이터를 개발하였다. 실제로 하드웨어가 완전히 구비되어 있지 않은 상황에서도 사용이 가능하며 장치 드라이버와 같은 API를 제공하여 편의성을 증대시켰다. 또한 동작속도가 기존의 시스템 수준 시뮬레이터에 비해 매우 빠르다. 커피 자판기용 프로그램을 검증하는 실험을 통해 사용의 편리성과 빠른 동작 속도를 확인하였다. 앞으로 이 검증 기법을 이용해 임베디드 시스템 개발이 쉬워지고 개발 기간도 단축될 것이다.

감사의 글

이 연구를 위해 연구장비를 지원하고 공간을 제공한 서울대학교 컴퓨터 연구소에 감사 드립니다. 이 논문은 2009년도 두뇌한국21사업에 의하여 지원되었으며, 2009년도 정부(교육과학기술부)의 재원으로 한국과학재단의 지원을 받아 수행된 연구(No. R0A-2007-000-20116-0, R33-2008-000-10095-0)입니다.

참고문헌

[1] Chi-Keung Luk, Robert Cohn, Robert Muth, Harish Patil, Artur Klauser, Greoff Lowney, Steven Wallace, Vijay Janapa Reddi, Kim Hazelwood, "Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation", ACM Programming Language Design and Implementation, 2005.

[2] Tipp Moseley, Daniel A. Connors, Dirk Grunwald, Ramesh Peri, "Identifying Potential Parallelism via Loop-centric Profiling", Proceedings of the 2007 Conference on Computing Frontiers (CF), May, 2007.

[3] Alexandre Strube, Emilio Luque, Dolores Rexachs, "Software Probes: towards a quick method for machine characterization and application performance prediction", 7th International Symposium on Parallel and Distributed Computing (ISPDC 2008), Krakow, Poland, July 2008.

[4] James Clause and Alessandro Orso, "A Technique for Enabling and Supporting Debugging of Field Failures", 29th International Conference on Software Engineering (ICSE'07), 2007.

[5] T Austin, E Larson, D Ernst, "SimpleScalar: an infrastructure for computer system modeling," Computer (IEEE Magazine), Vol 35, Issue 2, pp 59-67, Feb 2002.