가상 테스트를 위한 프레임워크

박창우*, 최경희*, 정기현**
*아주대학교 정보통신전문대학원
**아주대학교 전자공학부
e-mail : pismute@gmail.com

A framework for virtual testing

Chang-Woo Park*, Kyung-Hee Choi*, Ki-Hyun Jung**
*Graduate School of Information and Communication, Ajou University
**Division of Electronics Enginnering, Ajou Unniversity

요 약

기존의 방법으로 소프트웨어를 개발하는 것은 매우 비생산적이다. 소프트웨어의 기능과 성능에 상관없이 PC 에서 시뮬레이션을 통해서 개발할 수 있다면 하드웨어를 구현해야 하는 조건이 없어지기 때문에 시간과 비용이 절감되고 제품의 질도 향상될 것이다. 이 논문에서는 PC 에서 시스템 모델을 실행시켜 소프트웨어를 개발하고 테스트할 수 있는 프레임워크를 제안하고 실제로 비데를 테스트하는데 적용시켰다.

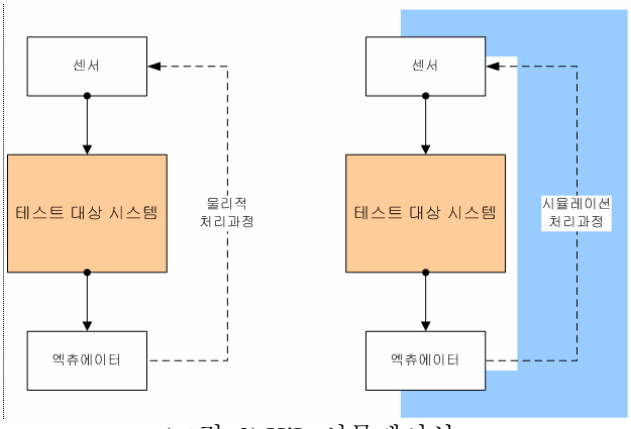
1. 서론

오늘날의 시스템은 매우 복잡해지고 있고 테스트 비용도 점점 증가하고 있다. 이 것을 해결할 혁신적인 해결책을 요구되고 있고 해결방법으로써 가상 테스트(virtual test) 기법[1]이 대두되고 있다.

시스템을 테스트하려면 모든 주변 장치를 갖춘 프로토타입 시스템을 구축하는 수 밖에 없었고 구축하는 것에도 매우 많은 시간과 비용을 들여야 했다. 하드웨어가 만들어지는 몇 달 동안에 소프트웨어 개발자는 시간을 낭비해야 했으며 하드웨어를 만들었다고 해도 모든 소프트웨어 개발자를 충족시킬만한 하드웨어를 공급하는 것은 비용이 너무 많이 들었다. 또 하드웨어 성능이 발전함에 따라서 전자 장치에서 소프트웨어가 차지하는 비중이 계속 증가하고 있다. 전통적으로 소프트웨어의 비중이 높았던 서버, 네트워크 장비, PC 등 뿐만 아니라 항공기, 자동차, 핸드폰, 비디오 게임기 등의 일반 가전기기에서도 소프트웨어 비중은 커지고 있다.

1.1. HiL(Hardware-in-the-loop) 시뮬레이션

HiL 시뮬레이션은 복잡한 시스템을 구현하는데 있어서 점점 증가하는 개발비용을 감소시키고 테스트를 매우 정교하게 수행하기 위해서 1950 년대 미국 방위산업과 우주항공산업에 도입되었다[2]. HiL 시뮬레이션은 프로토타입 하드웨어를 만들지 않고 PC 에서 시뮬레이션을 하기 때문에 다음과 같은 장점[3]을 가지고 있다. 첫째 테스트 환경을 소프트웨어로 구현함으로써 상대적으로 값 싸게 테스트 대상 시스템을 테스트 할 수 있고 원하는 시나리오 대로 환경을 구축할 수 있다. 둘째, 블랙박스 테스트에서는 테스트 대상 시스템의 내부를 모니터링 할 수 없으므로 테스트 대상 시스템과 HiL 시뮬레이터 사이의 주고받는 데이터를 기록 분석할 수 있다. 셋째, 시뮬레이션 환경을 구축하면 거의 무한하게 테스트 할 수 있다. 넷째, 아직 개발되지 않은 장치를 시뮬레이션 함으로써 테스트 대상 시스템을 테스트할 수 있다. 다섯째, 테스트해보기 어려운 상황도 시뮬레이션으로 연출하여 테스트 대상 시스템을 테스트 할 수 있다. 예를 들어, 배나 비행기가 침몰하거나 추락하는 상황에서 엔진을 제어하는 ECU(Electric Control Unit)을 테스트하려 한다면 천문학적인 비용이 소요될 것이다. 그러나 HiL 시뮬레이션을 이용함으로써 테스트를 저비용으로 완료할 수 있다.



(그림 2) HiL 시뮬레이션

가상 테스트 기법을 이용하면 시스템의 스펙을 작성하는 단계에서 시스템 모델을 테스트할 수 있다[4].

* 본 연구는 정보통신부 및 정보통신연구진흥원의 해외교수요원초빙사업의 연구결과로 수행되었음

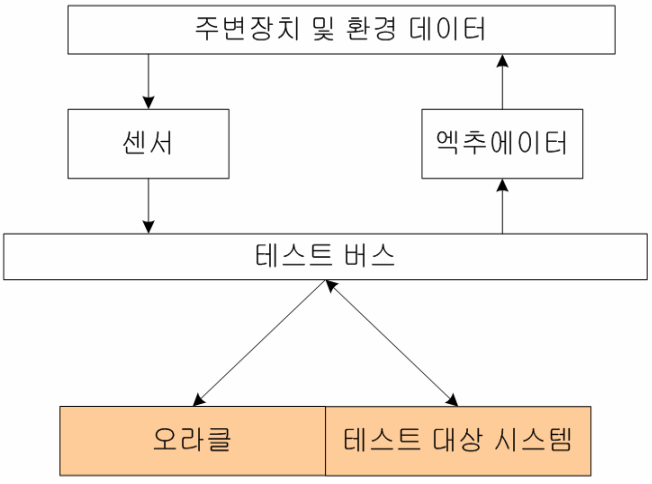
스펙을 Simulink, 사양 적합성 자동 검사[5]같은 방법을 이용하여 실행할 수 있게 작성한다면 우리는 그 모델을 실행시켜서 모델 자체를 테스트 할 수 있다. 프로토타입 하드웨어를 구현하기 전에 스펙을 작성하는 단계에서도 많은 오류를 검출할 수 있다.

테스팅 기법중 널리 쓰이는 방법으로 명세 기반 테스팅(specification based testing)이다. 시스템 명세는 설계자, 개발자, 고객에게 까지 모두 필요하며 널리 사용된다. 우리는 이미 작성된 시스템 명세를 사양 적합성 자동 검사 방법[5]를 이용할 수 있도록 테이블 형식으로 시스템 모델을 작성하여 이 모델을 시뮬레이션하는데 이용하였다.

이 논문에서는 HiL 시뮬레이션 기법을 이용하는 가상 테스트 프레임워크를 제안하고 설명한다. 2 장에서는 가상 테스트 프레임워크의 구조를 설명하고 3 장에서는 비데에 적용한 사례를 소개한다.

2. 가상 테스트 프레임워크

테스트를 진행하기 위해서는 먼저 테스트 대상 시스템이 있어야 하고 테스트 대상 시스템을 검증하기 위한 테스트 오라클[6]이 있어야 한다. 그리고 테스트 대상 시스템 및 오라클과 상호작용하는 가상 장치인 센서와 엑추에이터들이 있을 수 있다. 장치들이 서로 메시지를 교환하기 위해서는 통신 채널이 필요하다. 여기서는 그 통신 채널을 테스트 버스라고 부른다. (그림 3)은 가상 테스트 프레임워크를 그림으로 표현한 것이다.



(그림 3) 가상 테스트 프레임워크

2.1. 가상 장치

테스트 대상 시스템과 연동되는 장치의 입출력을 가상으로 가상으로 만들어 주는 장치이다. 이 장치는 하드웨어가 아닌 소프트웨어가 된다.

장치는 입출력 형태에 따라 센서와 엑추에이터로 구분할 수 있다[2]. 센서는 테스트 대상 시스템과 오라클에 값을 입력하는 장치를 의미하고 엑추에이터는 테스트 대상 시스템과 오라클이 일정 신호를 보내는 장치를 의미한다.

센서 역할만을 수행하는 장치도 있고 엑추에이터 역할만을 수행하는 장치도 있지만 센서, 엑추에이터 두 가지 역할을 모두 하는 장치도 있다. 모터의 예를 들면, 모터는 1000rpm 으로 동작하라는 신호를 받기 때문에 엑추에이터이지만 모터가 정말 1000rpm 으로 제대로 동작하는지 확인되어야 하기 때문에 센서도 되어야 한다. 센서와 엑추에이터의 구분은 장치의 다양성으로 볼 때 형식적인 한계를 벗어 날 수 없다. 대부분의 장치는 모터같이 센서이면서 엑추에이터인 형태로 구현되어야 할 필요가 있다.

또 우리는 장치의 예상하지 못한 오류를 테스트 해보기 위해서 장치가 동적으로 테스트 버스에 연결시킬 수 있는 기능을 가상 장치에 포함시켰다. 가상 장치는 <표 1>의 네 가지 명령을 테스트 스크립트로부터 입력 받을 수 있고 이 기능을 이용하여 장치에 문제가 생기는 오류를 입력하여 오류 테스트도 할 수 있다.

명령어	설명
ON	장치를 작동시킨다.
OFF	장치를 중지시킨다.
FAULT	장치를 중지시키는데 오류를 주입하기 위해 테스트 스크립트에 명시된 신호를 발생시키고 장치를 중지시킨다.
HOLD	FAULT 와 마찬가지로 오류를 주입하기 위해 사용되지만 장치의 작동을 중단하지 않고 테스트 스크립트에 명시된 신호를 계속 발생시킨다.

<표 1> 가상 장치 명령어

2.2. 테스트 오라클과 테스트 대상 시스템

테스트 결과, 테스트 대상 시스템이 올바르게 동작했는지를 판단하기 위해서 오라클과 테스트 대상 시스템은 동일한 데이터가 입력되어야 한다. 동일한 데이터가 입력되지 않으면 오라클과 테스트 대상 시스템이 동일한 결과를 산출할 것이라고 기대할 수 없다.

2.3. 테스트 모드

하나의 시스템을 테스트할 때 테스트 대상 시스템뿐만 아니라 주변장치까지 무결할 수 없다. 따라서 가상 장치를 이용하여 테스트 할 수도 있고 실제 주변장치를 연결시켜서 테스트 할 수 있는 실제 장치 테스트 모드와 가상 장치 테스트 모드로 구분하여 테스트를 수행할 수 있어야 한다. 주변장치의 문제가 발견되어도 테스트를 계속 진행 할 수 있다.

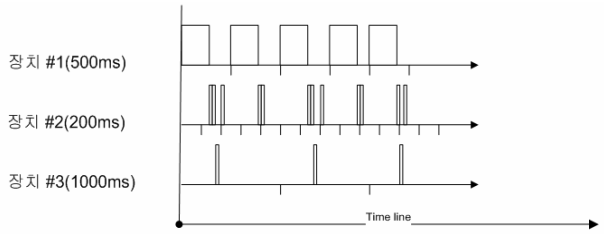
2.4. Inspector

모든 하드웨어가 완벽하게 갖추어진 상태에서 테스트를 진행 한다면 장치는 테스트 대상 시스템에게 연속적인 신호를 보내게 되지만 PC 에서 소프트웨어로 시뮬레이션 하는 경우에는 연속적인 신호를 보내지 못한다. 그래서 이산적으로(주기적으로) 신호를 보낼 수 밖에 없다.

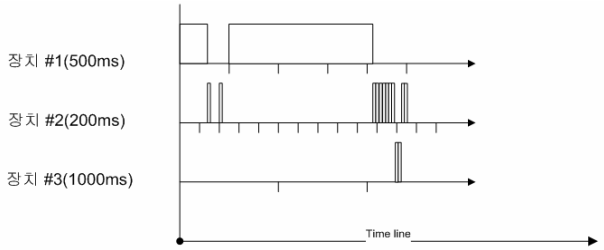
다 수의 장치를 사용하는 경우에 적어도 두 개 이상의 장치에서 동시에 신호를 발생 시킬 수 있지만

PC 에서 시뮬레이션을 하는 경우에는 동시에 신호를 발생시킬 수 없다. 세 개의 장치가 있고 각각 500ms, 200ms, 1000ms 를 가지고 신호를 발생시키는 장치가 있다고 가정하자. 동시에 신호를 발생시켜야 하는 경우 우리는 순차적으로 수행시켰고 그리고 특정 장치에서 신호를 발생시키는데 많은 시간이 소요됐을 때 단순한 배치작업으로 해결하였다.

(그림 4-1)과 (그럼 4-2)는 이 알고리즘을 그림으로 나타낸 것이다. 직사각형은 실행 시간을 의미하고 장치 #1 은 장치 #2, 장치 #3 보다 실행시간이 길다. 먼저 장치 #1 을 실행시키고 장치 #1 이 끝난 후 장치 #2, 장치 #3 을 수행한다.



(그림 4-1) Inspector 의 좋은 수행주기

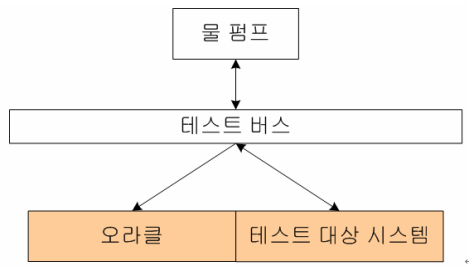


(그림 4-2) Inspector 의 나쁜 실행주기.

현재의 Inspector 는 두 가지 문제를 해결 할 수 없다. 첫째, 동시에 모델을 실행 시킬 수 없고, 장치의 주기마다 정확하게 실행되도록 보장하기 어렵다. 각 장치들의 실행 횟수를 유지시키는데 초점을 맞추었다. 실행시간이 오래 걸리는 장치가 있다면 (그림 4-2)에서처럼 장치 #2, 장치 #3 은 제때 실행 되지 못한다. 그러므로 적절한 주기를 알아내는 것이 필요하다.

3. 적용 사례

우리는 이 프레임워크를 적용하여 비데를 테스트하였다. 우리가 테스트 한 비데의 경우 이미 완벽하게 구현된 제품이기 때문에 비데의 ECU 를 테스트 대상 시스템으로 삼은 것이 아니라 물 펌프를 제외한 비데의 모든 장치를 테스트 대상 시스템으로 설정했다.



(그림 5) 비데의 가상 테스트 프레임워크 구성

물 펌프는 모터를 동작시켜야 한다는 메시지를 비데로부터 받기 때문에 엑추에이터인 동시에 비데는 기대한 대로 펌프가 작동하는지 확인해야 하기 때문에 센서도 된다.

우리는 이미 상용화된 제품을 테스트했는데도 오류를 검출 할 수 있었으며 설계자가 원하는 제품의 품질을 스펙 기준으로 평가한다면 품질 면에서 뛰어난 제품이라 하기에 부족한 점이 많았다.

4. 향후 과제

우리는 이 프레임워크를 비데에 적용시켜 성공적으로 테스트를 마쳤다. 그러나 아직 몇 가지 문제점이 존재한다. 첫째, 리얼타임 OS 에 이식하여 테스트의 정확도를 높여야 한다. 이번에 테스트한 비데의 경우 높은 정확도는 요구되지 않아서 리얼타임 운영체제를 이용하진 않았다. 하지만 리얼타임 운영체제에서도 문제가 없는지 적용해봐야 한다. 둘째, Inspector 의 스케쥴 알고리즘을 좀 더 개선 할 필요가 있다. 특정 장치의 수행 시간이 매우 길어진 다면 다른 장치가 올바르게 작동하지 않을 수도 있다.

참고문헌

[1] P.S. Magnusson. The virtual test lab, Computer Volume 38, Issue 5, May 2005 Page(s):95 – 97

[2] S. Nabi, M. Balike, J. Allen, and K. Rzemien. An overview of hardware-in-the-loop testing systems at Visteon. SAE technical paper series, SAE International, 400 Commonwealth Drive, Warrendale, PA 15096-0001 USA, March 2004.

[3] M. Schlager, W. Elmenreich, I. Wenzel. Interface Design for Hardware-in-the-Loop Simulation, Industrial Electronics, 2006 IEEE International Symposium on Volume 2, Page(s):1554 - 1559, July 2006

[4] C. Guehmann. Model-Based Testing of Automotive Electronic Control Units, 3rd International Conference on Materials Testing: Test 2005

[5] 김영수, 김장복, 최경희, 정기현, 장중순, 박승규, 사양 적합성 자동 검사 방법, 제 24 회 한국정보처리학회 추계학술발표대회 논문집 제 12 권 제 2 호 , 2005. 11

[6] L. Baresi and M. Young. Test oracles, Technical Report CIS-TR01 -02, Dept. of Computer and Information Science, Univ. of Oregon, http://citeseer.ist.psu.edu/474054.html, 2001.