임베디드 테스팅을 위한 아키텍처 설계

김지혁*, 김상일*, 류성열*
숭실대학교 대학원 컴퓨터학과
e-mail : ji-hyeok@selab.ssu.ac.kr

Architecture Design for Embedded Testing

Ji-Hyeok Kim, Sang-Il Kim, Sung-Yul Rhew
Dept. of Computing, Soongsil University

요 약

임베디드 소프트웨어 개발의 비중이 점차 높아져 감에 따라 임베디드 테스팅의 중요성이 부각되고 있다. 임베디드 테스팅과 일반적인 애플리케이션 테스팅은 많은 차이점이 있다. 일반 애플리케이션 테스팅은 개발 환경과 테스팅 환경이 동일하다. 그러나 임베디드 테스팅은 테스팅 대상이 플랫폼에 기반한 타겟이라는 점에서 여타의 애플리케이션 테스팅과는 다르다. 따라서 임베디드 테스팅을 위한 아키텍처 설계를 고려할 필요가 있다. 임베디드 테스팅을 위한 아키텍처들이 몇몇 제시되고는 있지만 실제 적용해서 사용하기에는 여러 가지 문제점이 존재한다. 본 연구는 애플리케이션 테스팅의 아키텍처를 비교, 분석하고 임베디드 테스팅에 적용 가능한 요소들을 추출한다. 또한 임베디드 테스팅을 위한 여러 아키텍처를 비교, 분석하여 임베디드 테스팅에 커스터마이즈 된 아키텍처를 설계하고자 한다. 향후에는 본 연구에서 제안된 아키텍처를 기반으로 실제 임베디드 테스팅에 적용하고자 한다.

1. 서론

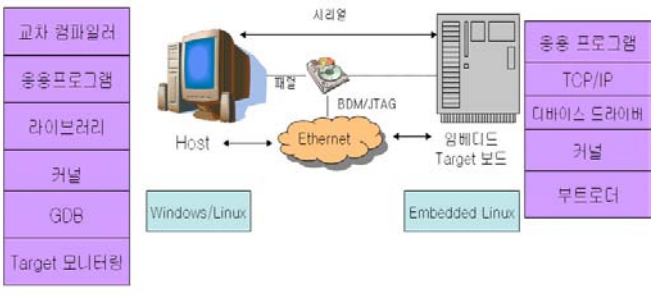
현재의 임베디드 테스팅을 위한 프로세스와 아키텍처가 잘 정립되어 있지 않다. 임베디드 테스팅의 수요는 늘어나고 중요성 또한 부각되고 있지만, 실질적으로 임베디드 테스팅이 잘 이루어지지 않는다. 첫번째 이유로 임베디드는 플랫폼에 의존적이다. 그렇기 때문에 여타 다른 소프트웨어에 비해 임베디드 소프트웨어는 플랫폼, 즉 하드웨어 환경에 많은 영향을 받고 있다. 두번째 이유로 교차개발환경이라는 것이다. 여타의 소프트웨어는 개발환경과 테스팅 환경이 같다. 그러나 임베디드 소프트웨어는 개발환경과 테스팅 환경이 다르다. 임베디드 소프트웨어의 개발환경은 일반 PC이지만, 테스팅 환경은 소프트웨어가 탑재된 플랫폼에서 이루어져야 하기 때문에, 일반적인 소프트웨어 테스팅 방법으로 임베디드 소프트웨어 테스팅을 하기에는 매우 어렵다.

본 논문에서는 현재 제시하고 있는 임베디드 테스팅 방법의 비교, 분석을 통하여 플랫폼에 독립적인 임베디드 테스팅 아키텍처를 설계하고자 한다.

2장에서는 임베디드 테스팅의 아키텍처에 대해 연구하고, 3 장에서는 2장에서의 연구에 기반하여 임베디드 테스팅을 위한 아키텍처를 제시하고, 4장에서 결론 및 향후 연구로 끝맺음을 한다.

2. 관련연구

2.1 교차개발 환경



[그림 1] 임베디드 교차개발 환경

일반적인 임베디드의 교차개발 환경은 [그림 1]과 같다. 즉, S/W가 수행될 시스템과 개발하는 시스템이 서로 다른 개발 환경이다.

2.2 DO-178B 스펙

DO-178B 인증은 항공 프로젝트에 종사하는 벤더에게 요구되는 조건으로서 안전성이 중시 되는 핵, 의학, 통신 프로그램을 포함하는 회사들에게 그 중요성이 증대되고 있다. 모든 안전성이 요구되는 소프트웨어에 적용할 수 있는 DO-178B 표준은 안전성이 요구되는 소프트웨어의 개발에 사용되는 프로세스와 목적에 대한 가이드라인을 수립한다.

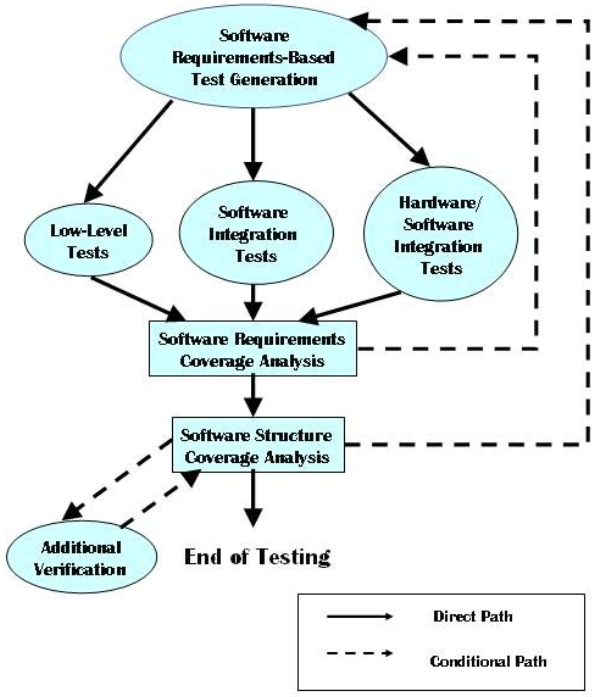
DO-178B 스펙의 생명 주기는 총 6개의 프로세스로 구성되어 있지만, 본 연구에서는 Software Verification Process(소프트웨어 검증 프로세스)만을 참고로 한다.

검증은 간단한 테스팅이 아니다. 일반적으로 테스팅은 에러의 부재를 밝히지는 않는다. 소프트웨어 프로세스 목적이 논의될 때 테스트 대신에 검증이라는 용어는 일반적으로 재검토, 분석, 그리고 테스트의 결합으로 사용한다. 소프트웨어 레벨이 점점 낮아질수록 다음은 덜 강조된다.

❑ 저-수준 요구사항의 검증
❑ 소프트웨어 아키텍처의 검증
❑ 테스트 커버리지의 정도
❑ 검증 절차의 통제
❑ 소프트웨어 검증 프로세스 활동의 독립성
❑ 다중 검증 활동 각각은 같은 종류의 에러를 탐지할 것이다.
❑ 통합 테스팅
❑ 에러 방지나 탐지에 대한 간접적인 영향을 가진 검증 활동.(예, 소프트웨어 개발 표준의 일치)

DO-178B 스펙에서는 다음과 같은 소프트웨어 테스팅을 제시한다.

❑ 하드웨어/소프트웨어 통합 테스팅: 타겟 컴퓨터 환경 안에서의 소프트웨어의 정확한 운용을 검증한다.
❑ 소프트웨어 통합 테스팅: 소프트웨어 요구사항과 컴포넌트 사이에서의 상호관계를 검증 소프트웨어 아키텍처 안에서의 소프트웨어 요구사항과 소프트웨어 컴포넌트의 구현을 검증한다.
❑ 저-수준 테스팅: 소프트웨어 저-수준 요구사항의 구현을 검증한다.

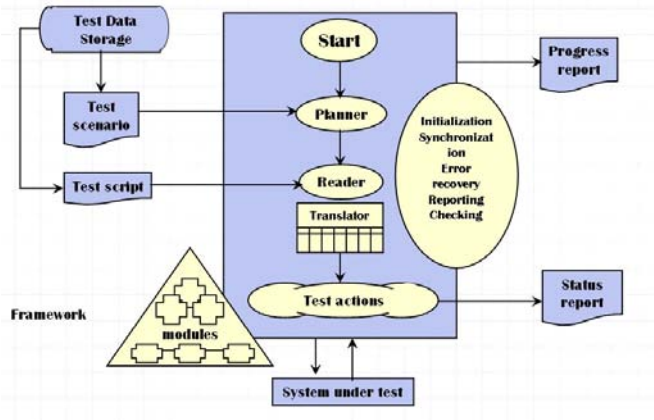


[그림2] 소프트웨어 테스팅 프로세스

[그림 2]에서의 세가지 테스팅 타입의 목적은 다음과 같다.

2.3 Test Suite Generator

테스트 슈트는 일정한 순서에 의하여 수행될 개별 테스트들의 집합, 또는 패키지이다. 슈트는 응용 분야나 우선순위, 내용에 연관된다.



[그림 3] 테스트 슈트의 청사진

[그림 3]에서의 간단한 용어는 다음과 같다.

❑ Start : 테스트슈트(test suite)가 시작되는 동안

에, 환경이 초기화된다.

❑ Planner : 테스트시나리오를 읽는다. planner는 테스트스크립트에 대한 정보를 전달하여 실행이 가능하도록 한다.

❑ Reader: 테스트스크립트로부터 정보를 추출한다.

❑ Translator : 테스트슈트의 라이브러리에서 테스트스크립트에서 언급한 액션과 일치하는 기능을 찾는다.

❑ Test action : system specific function으로 구현되며, 요소들의 집합 표준을 포함하고 있다.

3. 임베디드 아키텍처 제안

3.1 Cross Compile

임베디드 테스팅은 교차 컴파일이 가능해야 한다. 소프트웨어를 개발하는 Host측과 개발된 소프트웨어를 타겟 보드에 올려 돌리는 Target으로 구분되기 때문이다. 따라서 Host측에서 컴파일 한 것을 Target쪽에서 이용할 수 있는 교차 컴파일이 가능해야 한다.

3.2 Agent

Host와 Target으로 구분이 되어있기 때문에 Host와 Taget사이의 통신을 담당하는 Agent가 있어야 한다.

3.3 실시간모니터

임베디드 시스템은 실시간이라는 특징이 있다. 따라서 실시간 모니터는 Build 과정을 통해 생성된 테스트하니스를 타겟시스템으로 다운로드하고 타겟시스템에서 수행된 테스트하니스를 통해 생성된 TestResult를 다시 호스트시스템으로 업로드 하는 과정을 관리한다. 실시간 모니터는 결과를 캡쳐링(Capturing) 하여 Repository에 저장하고 Report생성을 위한 입력으로 이용된다. 즉, 실시간모니터는 타겟시스템에서 호스트시스템으로 TestResult가 언제 넘어오는지 감시를 하고 있다가, TestResult가 넘어오는 순간 TestResult를 캡처해서 Repository에 저장한다.

3.4 Host 측 TestEngine과 Target 측 TestEngine

테스트의 결과는 Target 측에서 실행된 결과를 의미한다. 그러나 교차개발환경이기 때문에 테스트시에 원하는 결과를 얻기 위해서 Host 측에서 원하는 테스트 모듈을 삽입하며 Target 측으로 넘겨야 한다. 또한 Target 측에서도 보드나 Realtime OS에 의존적이기 때문에 Target 측에서 한번 더 테스트 모듈을 삽입하여 테스트를 해야 한다. 따라서 Host 측과 Target측에 각각 TestEngine이 존재한다.

3.5 TestCase Generator

프로그램 테스팅 과정에서 가장 중요한 일 중 하나는 효과적인 테스트케이스를 설계하는 것이다. 테스트케이스 설계 작업이 중요한 이유는 완벽한 테스팅 즉, 모든 에러를 찾는 테스팅이 불가능하기 때문에 프로그램을 테스트할 때는 불완전한 테스팅을 할 수밖에 없으며, 따라서 테스팅 과정은 최대한의 불완전성을 줄이는데 초점이 맞추어 진다.

TestCase Generator는 테스트 하려고 하는 것이 요구에 맞게 개발되었는지 확인하기 위하여 테스트 입력값을 테스트 자동화를 통하여 입력하고 테스트 예상 결과값과 테스트 결과값이 일치하는 지를 확인한다.

TestCase는 테스트케이스 이름, 테스팅 하고자 하는 기능들의 항목(function, processes, services, …), 기능들에 적용될 입력들의 이름과 값의 목록(List), 기능들을 수행한 결과로 얻어질 출력 값들의 목록(List) 등으로 구성된다.

3.6 Repository

Repository는 임베디드 S/W 테스트에 관련된 데이터를 저장하기 위한 것이다. Repository는 다음과 같은 세 가지의 타입으로 분류할 수 있다.

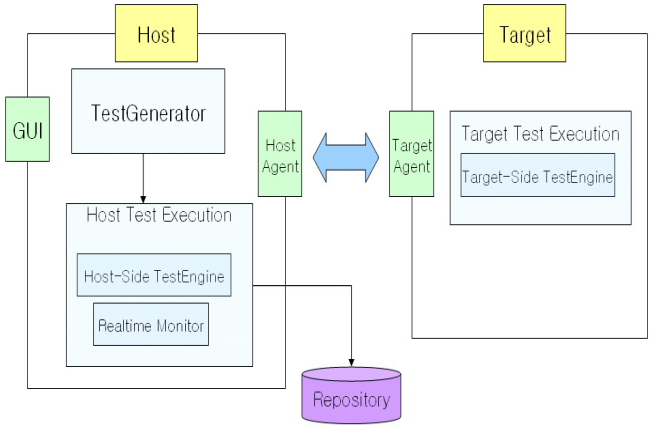
❑ 임베디드 S/W 테스트 수행 과정에서 필요한 Instrumented SourceCode 정보를 저장하는 Repository

❑ 호스트측 테스트엔진을 통해 분석된 테스트 결과를 저장하는 Repository

❑ Report 생성을 위해 Reference File을 저장하는 정제된 테스트 결과를 저장하는 Repository

4. 결론

임베디드 시스템은 일반적인 소프트웨어처럼 개발환경과 테스트 환경이 동일하지 않다. 따라서 일반적인 소프트웨어와 같은 방식으로 테스트 하기는 어렵다. 따라서 본 연구에서는 임베디드 테스팅을 위한 환경과 임베디드 테스팅 시에 필요한 소요들에 대해 논의하였다.



[그림 4] 개략적인 테스팅 아키텍처

[그림 4]는 본 연구에서 제시하는 개략적인 아키텍처이다. 이와 같은 구조를 가지고 세부적으로 더 상세하게 설계해야 할 것이다. 그러나 본 논문에서는 세부적인 아키텍처는 제시하지 않겠다.

향후에는 본 연구에서 제시한 아키텍처를 기반으로 실제 임베디드 시스템 테스팅에 적용해보고, 효과가 있는지 얼마나 테스팅이 효과적으로 이루어졌는지 확인할 것이다.

참고문헌

[1] RTCA SC-167/EUROCAE WG-12 "DO-178B Specification(Software Considerations in Airborne Systems and Eouipment Certification)"

[2] Bart Broekman, Edwin Notenboom, “Testing Embedded Software”, Addison Wesley, 2003

[3] Arnold S. Berger, “Embedded Systems Design : An Introduction to Processes, Tools, and Techniques”, CMP, 2001.

[4] T. Sridhar, "Designing Embedded Communications Software", CMP, 2003

[5] Hollabaugh, "Embedded Linux -Hardware, Software, and Interfacing", Addison Wesley, 2002