KIPS Tr. Software and Data Eng.
Vol.6, No.7 pp.361~370 pISSN: 2287-5905

https://doi.org/10.3745/KTSDE.2017.6.7.361

Architecture for Simulink/Stateflow Model Based Test Case Generation Considering Feedback

WooWon Choi† · Kihyun Chung†† · Kyunghee Choi†††

ABSTRACT

This paper proposes the architecture of test case generator that can generate test cases, considering feedback signals from subsystems controlled by an embedded system. In general, a closed system decides the next test input to its subsystem under its control referencing feedback signals from its subsystem. In such systems, it is hard to use the typical test cases generated without referencing feedback. The architecture proposed in this paper re-produces test cases in real time using feedback signals. The architecture is implemented and its effectiveness is verified through experimenting a demo system.

Keywords : Embedded System, Testing, Feedback, Test Case, HILS

피드백을 고려한 테스트 케이스 생성 시스템 구조

최 우 원† · 정 기 현†† · 최 경 희†††

요 약

본 논문은 임베디드 시스템이 제어하는 서브시스템의 실시간 피드백을 반영하여 테스트 케이스를 생성할 수 있는 테스트 생성기 구조를 제안한다. 일반적으로 폐쇄형 임베디드 시스템은 자신이 제어하는 서브시스템의 피드백을 참조하여 서브시스템의 다음 상태를 제어하는 값을 결정한다. 그와 같은 시스템에서는, 피드백을 고려하지 않는 전통적인 테스트 케이스는 사용하기 어렵다. 본 논문에서 제안하는 테스트 케이스 생성기 구조는 서브시스템의 피드백을 이용하여 다음에 사용할 테스트 케이스를 실시간 재구성한다. 제안하는 구조를 구현하고 데모 시스템을 이용하여 유용성을 검증한다.

키워드 : 임베디드 시스템, 테스팅, 피드백, 테스트 케이스, HILS

1. 서 론

오늘날의 소프트웨어가 내장된 임베디드 시스템은 우리 생활의 곳곳에 존재 한다. 가전제품, 자동차 전자제어장치, 국방 유도 무기 제어장치 등 생활 곳곳에 있는 대부분을 임베디드 시스템으로 분류할 수 있다. 이러한 임베디드 시스템은 안정성과 신뢰성이 보장되어야 한다. 특히 인명 피해에 관련되거나 정확한 동작이 요구되는 자동차 제어기, 유도 무기 제어장치 등에 대해서는 신뢰성과 안정성이 더욱 더 필수적이다.

※ 본 연구는 방위사업청(UD150042AD)의 지원으로 수행되었음.
† 준 회 원 : 아주대학교 전자공학과 석사과정
†† 정 회 원 : 아주대학교 전자공학과 교수
††† 정 회 원 : 아주대학교 소프트웨어학과 교수
Manuscript Received : October 21, 2016
First Revision : December 12, 2016
Accepted : January 2, 2017
* Corresponding Author : Kihyun Chung(khchung@ajou.ac.kr)

임베디드 시스템 오류에 의한 피해를 보면 1992년 비행제어시스템(FCS)의 임베디드 시스템 소프트웨어 오류로 인한 F-22 착륙사고[1], 1996년 데이터 오버플로우로 인한 아리안 5호 폭발 사건[2], 2009년 전자제어장치(ECU)의 소프트웨어 오류로 인한 Toyota 급발진 사고[3] 등이 있다. 이러한 문제는 경제적, 인적 피해를 유발하는데 이를 방지하기 위하여 다양한 테스트 방법으로 에러를 검출하고 신뢰성 및 안정성을 검증을 위해 노력하고 있다.

소프트웨어가 동작을 제어하는 임베디드 시스템을 테스트하기 위해서는 테스트 목표에 맞는 테스트 환경과 조건이 요구된다. 여러 가지 테스트 조건 중에서, 테스트 환경 구축을 어떻게 할 것인지, 구축된 환경에서 테스트 하고자하는 목표에 맞는 테스트 케이스(Test case)를 어떻게 만들 것인지가 중요한 현실적인 문제이다. 왜냐하면, 테스트 환경 구축에 따라서 검증되는 범위가 달라질 수 있고 구축된 환경에서 어떤 테스트 케이스를 사용하느냐에 따라서 테스트 되

는 조건 및 테스트의 합목적성을 결정할 수 있기 때문이다.

테스트 케이스는 테스트하고자 하는 목표에 따라서 만들어야 한다. 예를 들어, 임베디드 시스템이 그 시스템이 처할 수 있는 상태에 도달하는 지를 테스트할 목적으로 만들어지는 테스트 케이스와 상태 간 전이에 필요한 조건을 테스트할 목적으로 만들어지는 테스트 케이스는 달라진다. 또한, 소프트웨어 소스코드를 대상으로 하는 화이트박스 테스트(White box test)를 위한 테스트 케이스와 소프트웨어가 요구사항에 맞게 구현되었는지를 확인하는 블랙박스 테스트(Black box test)를 위한 테스트 케이스는 다르다[4, 5]. (본 논문에서는 블랙박스 테스트에 대해서 다룬다.)

블랙박스 테스트를 위한 테스트 케이스는 실제 임베디드 시스템에 임베디드 소프트웨어를 내장한 상태로 동작시키면서 그 임베디드 시스템의 요구사항에 맞게 동작하는 지를 테스트하기 위한 것이다. 이러한 테스트 케이스는 시스템 입력의 조합으로 구성되며 그 시스템 입력에 따른 출력 정보도 테스트 결과 확인을 위해 필요하다.

임베디드 시스템은 메인 시스템과 메인 시스템이 동작 시키는 서브시스템을 가질 수 있다. 예를 들면, 모터 제어기는 모터 속도를 제어하는 임베디드 시스템인 메인 시스템과 시스템의 명령에 따라 동작하는 서브시스템인 모터로 구성되어 있다. 이러한 임베디드 시스템은 서브시스템으로부터 피드백을 받아서 메인 시스템 제어 방법을 실시간으로 바꾸는 경우(폐쇄형 시스템: Closed system)와 서브시스템의 피드백을 받지 않고 메인 시스템이 일방적으로 서브시스템을 제어하는 경우(개방형 시스템: Open system)로 나눌 수 있다.

개방형 시스템을 테스트 할 경우, 피드백 정보와 무관하게 메인 시스템만을 고려하면 되고 이러한 테스트 케이스를 본 논문에서는 비실행 테스트 케이스(Static test case)라 정의한다.

반면, 폐쇄형 시스템을 테스트 할 경우, 메인 시스템뿐만 아니라 서브시스템의 피드백도 고려하여 테스트를 진행하여야 한다. 만약 피드백이 있는 시스템 테스트에 미리 정해진 비실행 테스트 케이스를 사용한다면, 실제 서브시스템의 피드백을 고려하지 않은 테스트 케이스가 입력되고, 시스템은 실제와 다르게 동작하여 우리가 원하는 테스트를 진행하기 어렵다.

예를 들면, 모터제어기 메인 시스템이 서브시스템인 모터에 ‘1000rpm’(revolution per minute) 이라는 명령을 주면 모터는 정상적인 상태에서는 ‘레벨 1’이라는 출력을 메인 시스템에 피드백으로 보내야하고, ‘레벨 1’이라는 피드백을 받은 메인 시스템은 이전에 보낸 명령에 서브시스템이 정상 동작한다고 판단하여, 다시 ‘2000rpm’이라는 명령어를 보낸다고 가정하자. 비실행 테스트 케이스를 생성할 때 테스트 케이스 생성기는 서브시스템의 출력을 (보통은 임의로 정함) 정하여 생성할 수밖에 없다. 정상적이라고 정하여 피드백을 ‘레벨 1’이라고 생성할 경우, 이 비실행 테스트 케이스는 실제 시스템 테스트 시, 서브시스템이 정상적일 때는 사용할 수 있을 것이다. 하지만, 고장일 경우는 사용하기 어렵다. 이러한 경우, 시스템이 고장인지 아닌지를 알 수 없을 뿐만 아니라, 고장일 경우는 어떠한 피드백을 사용해야 할지 모르기 때문에 원하는 비실행 테스트 케이스로는 테스트를 진행할 수가 없다. 이 때는 서브시스템에서 오는 정확한 피드백 값을 사용하여 테스트 케이스를 생성하여야 고장난 서브시스템을 정확히 제어하는지를 테스트 할 수 있다.

본 논문에서는 피드백이 필요한 시스템에 사용하는 테스트 케이스 생성을 위한 구조를 제시한다. (이러한 테스트 케이스를 실행 테스트 케이스라 정의한다.) 실행 테스트 케이스(run-time test case) 생성을 위해서 기존의 방법들에서는 흔히 서브시스템을 모델로서 사용한다. 즉, 실제 서브시스템을 사용하는 대신 서브시스템 모델을 활용하여 피드백 값을 구하는 MILS(Model-In-the-Loop-Simulation) 구조를 많이 활용한다. 이 방법은 서브시스템의 특성이 잘 알려져 정확한 모델 구현이 가능하거나, 대략적인 모델을 사용하여도 테스트 케이스를 생성하는데 지장이 없는 경우에는 매우 유용한 방법이라 할 수 있다. 하지만 서브시스템의 정교한 피드백 값이 필요한 경우 또는 특성을 정확하게 모델링하기 어려운 복잡한 서브시스템의 경우는 모델을 활용하여 피드백을 구하기 어렵다.

서브시스템의 피드백을 활용하는 또 하나의 방법은 서브시스템을 메인 시스템에 연결하여 사용하는 HILS(Hardware-In-the-Loop-Simulation) 구조이다. 이 방법은 메인 시스템의 입력을 받은 서브시스템이 메인 시스템에 피드백 값을 직접 전달하여 메인 시스템이 정상적인 피드백 값을 이용할 수 있다. 하지만 HILS는 테스트 케이스를 실행시키면서 시스템을 테스트하는 용도로 사용하고 있고, 테스트 케이스를 생성하는 데는 사용하지 않았다.

본 논문에서는 이와 같은 경우에 서브시스템의 실제 피드백을 직접 받아 서브시스템의 동작을 실시간으로 모니터링하면서 테스트 정책에 필요한 실행 테스트 케이스를 생성할 수 있는 케이스 생성기 구조를 제안한다.

제안하는 구조의 가능성을 검증하기 위하여 테스트 케이스 생성기 시스템을 구축하고, 서브시스템으로 모터를 가지는 메인 시스템에서 모터의 피드백을 이용한 테스트 케이스 재생성(실행 테스트 케이스) 후 테스트 시나리오 재구성하는 실험을 진행하였다.

본 논문의 기여는 다음과 같이 정리할 수 있다.

1) 기존 HILS에서 메인 시스템의 서브시스템 피드백을 이용하여 실행 테스트 케이스 생성 방법 및 구조 제안
2) 제안 구조 구현 방안 제시
3) 실제 시스템을 대상으로 제안 테스트 케이스 생성 방법 검증

본 논문의 구성은 2장에 관련 연구, 3장에서는 제안하는 실행 테스트 케이스 생성기 구조에 대해 자세히 다루고 4장에서는 실험을 통한 검증을 보인다. 그리고 마지막 장에서 결론으로 본 논문을 마무리 한다.

2. 관련 연구

2.1 테스트 케이스 생성 방법

테스트 케이스 생성 방법으로는 크게 소스코드를 분석하여 진행하는 화이트박스 테스트에서 사용되는 테스트 케이스 생성과 요구사항에 초점을 맞춰 진행하는 블랙박스 테스트에서 사용되는 블랙박스 테스트 케이스 생성으로 나눌 수 있다[4, 5].

화이트박스 테스트 케이스는 소스 코드 수준에서 코딩 규칙이나 코드의 실행 오류(Run time error) 등을 파악하기 위해 개발 단계에서 많이 사용한다[5]. 하지만 소스 코드를 획득하기 어렵거나 하드웨어에 내장되어 시스템의 동작 오류를 검증하는데 사용하기는 어렵다. 또한 구현된 소프트웨어가 요구사항에 맞게 동작하는지 등을 검증하기 위해서는 추가적인 정보가 필요하다.

블랙박스 테스트 케이스는 소스 코드가 아니라 완성된 시스템이 요구사항 또는 소프트웨어 요구사항 명세서와 일치하는지를 테스트 하는데 사용된다[4]. 특히, 임베디드 시스템과 같이 하드웨어에 내장된 소프트웨어가 요구사항 대로 동작하는 지를 테스트하는 경우에 사용된다. 목표에 부합되는 테스트 케이스 생성의 어려움은 있으나, 설계문서와 같은 시스템의 구현 상세 정보 없이 테스트 케이스를 만들 수 있는 장점도 있다. 블랙박스 테스트 케이스 생성 방법으로는 임의(Random) 생성 방법, 시스템 입력 조합(Combinatorial)을 이용하는 생성 방법 및 모델 기반(Model-based) 생성 방법 등이 있다[18].

2.2 피드백이 있는 시스템 테스트

개방형 시스템에서는 테스트하고자 하는 SUT(System Under Test: 테스트 대상 시스템)를 구동하지 않고 SUT의 모델이나 요구사항을 기반으로 테스트 케이스를 생성한 비실행 테스트 케이스를 이용하여 테스트한다. Fig. 1에서 보는 바와 같이 테스트 케이스를 SUT에 입력만으로 시스템의 동작 정확성 여부를 확인한다. 그리고 서브시스템의 동작 결과와 무관하게 사전에 정해진 테스트 케이스를 실행시키면서 동작여부의 정확성을 확인한다.

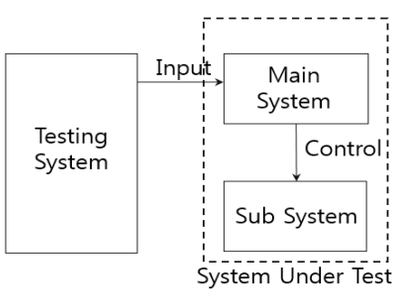


Fig. 1. Testing Set-up for Open System

예를 들어, 메인 시스템의 온도 센서 값이 10도 이하일 때 서브시스템인 히터를 작동 시키고 11도 이상일 때 히터 작동을 중지 시키며, 히터에서 메인 시스템으로 보내는 피드백 신호가 없는 개방형 시스템인 경우, 테스트 입력으로 온도 센서 값에 10도 이하의 값을 지속적으로 주면 된다.

비실행 테스트 케이스를 사용하는 이와 같은 경우의 테스트에서는 서브시스템 상태의 피드백 값은 고려하지 않고 오직 입력 값만을 보고 시스템이 정상적인 동작을 하는지를 테스트한다.

하지만 만약, 테스트 입력 값뿐만 아니라 서브시스템의 상태를 고려하여 테스트 입력을 계산해야 하는 경우라면, 시스템을 동작시켜 서브시스템의 상태를 피드백 받지 않고는 입력 상태를 제대로 제어할 수 있는 테스트 케이스를 생성하기는 힘들고, 따라서 비실행 테스트 케이스로는 SUT 동작의 정확성 여부를 판단하기 힘들다.

비실행 테스트 케이스 생성과는 달리 실행 테스트 케이스 생성은 시간 T_k에 서브시스템에 가해진 입력의 결과로 메인 시스템이 구동하는 서브시스템의 상태를 피드백 받아 T_{k+1}시점에 시스템의 입력(테스트 케이스)을 재구성하는 것을 말한다.

예를 들어, 메인 시스템의 온도 센서 값이 10도 이하일 때 서브시스템인 모터를 제어하는 시스템이라면, 메인 시스템은 조건에 맞는 모터의 출력 RPM(Revolution Per Minute)을 설정하고 해당 RPM을 생성하기 위한 신호를 발생 시킨다. 하지만, 대부분의 서브시스템(여기서는 모터)에서는 처음 설정된 값과 다르게 실제 출력은 오차가 발생한다. 오차의 발생을 최소화하기 위해서 일반적으로 엔코더(Encoder)를 부착 후 피드백을 이용하여 PID(Proportional Integral Derivative)제어한다. 이 경우, 엔코더의 출력을 피드백 받지 못한다면 메인 시스템에서 PID 제어 프로그램이 원하는 대로 동작하지 못할 것이고, 이에 대한 정확한 테스트도 어려울 것이다.

Fig. 2의 [CASE 2]는 피드백이 없는 경우의 상황이고 [CASE 1]은 피드백을 고려한 상황이다. 제어기가 서브시스템인 모터에 지령 값으로 50rpm을 주었을 때, 일반적으로 모터는 지령 값인 50rpm을 목표로 동작하려 시도한다. 하지만 모터 부하의 변동 등 다양한 이유로 모터 속도는 목표 값과 차이를 나타낼 수 있다. 이 때, 모터는 현재 자기의 속도를 제어기에 보내고 제어기는 그 차이를 보정하기 위해 출력 값을 가감하여 모터에 출력한다.

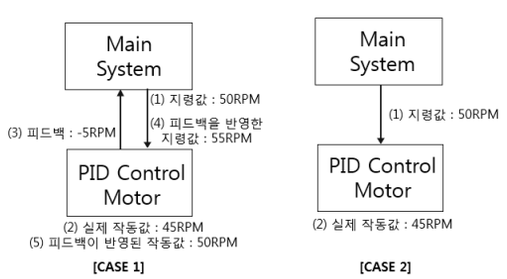


Fig. 2. Example System with and Without Feedback

[CASE 2]에서 제어기는 모터의 피드백이 없어, 속도를 알 수 없기 때문에 제어기가 출력한 값을 유지 시키게 되며 결과적으로 모터는 원하는 속도인 50rpm을 유지하기 어렵다. 이 경우 피드백을 참조하지 않아, 다음 지령 값은 임의로 정할 수밖에 없다. 이와 같이 피드백 값을 임의로 할당하는 경우, 할당 값은 실제 동작과는 거리가 있어 다음에 수행해야 할 지령 값을 제대로 생성하기가 힘들고 결과적으로 모터의 정확한 동작을 위한 실험이 어렵다.

위의 예시와 같이 테스트 케이스도 마찬가지이다. 피드백을 고려하지 않고 생성된 테스트 케이스는 실제 상황에 맞는 테스트 시나리오를 구성하기가 어렵다. 따라서 본 논문에서는 실제 환경에 근접한 테스트 시나리오를 만들기 위해서 피드백을 이용하여 테스트 케이스를 재구성(실행 테스트 케이스) 한다.

실제 서브시스템 없이, 실제 테스트 시나리오와 근접하게 테스트하기 위해서는 테스트 케이스 재구성이 필요하고, 이는 즉 서브시스템의 피드백이 필요하다. 서브시스템의 피드백을 대체 할 수 있는 현실적인 몇 가지 대안이 있다.

첫 번째, 서브시스템을 검증할 때 사용한 다양한 환경에서의 입출력 값을 이용하는 방법이다. 즉, 발생 가능한 다양한 입력에 대한 출력 값(이 출력은 SUT로의 피드백 입력임)을 미리 확보하고 있다가 피드백 생성 시, 메인 시스템의 출력(서브시스템의 입력)을 보고 그에 해당하는 서브시스템의 출력을 메인 시스템의 피드백으로 공급하는 방법이다. 메인 시스템으로부터의 발생 가능한 모든 입력에 대한 서브시스템의 출력을 다 확보할 수 없는 경우, 인접한 두 구간 사이를 선형 시스템으로 가정하고 출력을 추정하는 등의 방법을 사용한다. 하지만, 서브시스템의 실제 데이터 수집이나 모의실험이 어려운 경우도 많이 있어 이 방법을 사용하는데 많은 제약이 있다.

두 번째, 서브시스템의 모델을 사용하는 방법이다. 모델은 C와 같은 언어로 작성된 소프트웨어 모형을 사용할 수도 있고, UML이나 Simulink와 같은 모델링 도구를 사용하여 작성된 모델일 수도 있다. 서브시스템이 간단한 시스템이라고 하면 모델을 이용해서 모든 상황을 표현할 수 있지만, 복잡한 모델에 대해서는 모든 상황을 다 표현하기가 어렵다는 단점이 있다. 따라서 모델을 사용하는 경우, 서브시스템의 개략적인 행동 모델을 사용한다. 모델링의 어려움은 있지만, 비교적 현실적인 방법이다. 하지만 이 역시 정확한 제어 상황을 테스트 하는 데는 한계가 있다.

3. 실행 테스트 케이스 생성기 구조

3.1 실행 테스트 케이스 생성기 구조

본 절에서는 메인 시스템, 서브시스템을 구동하고 서브시스템의 피드백을 이용하여 테스트 케이스(실행 테스트)를 생성하는 방법 및 구조를 제안한다.

Fig. 3은 본 논문에서 제안하는 실행 테스트 케이스 생성 환경을 보여 주고 있다. 테스트 대상인 메인 시스템에 입력을 인가하고 결과를 수집하는 역할을 하는 테스트 시스템(Testing System), SUT 내의 제어기 메인 시스템, 서브시스템 및 실행 테스트 케이스 생성 모듈로 구성된다. 실행 테스트 케이스 생성 모듈은 테스트 케이스 생성 전략에 따라 테스트 케이스 생성기가 시스템 입력과 서브시스템의 피드백을 사용하여 실행 테스트 케이스를 생성한다. 실행 테스트 케이스는 시간 T_k에 SUT에 입력되고 SUT는 그 입력을 활용하여 서브시스템을 제어하기 위한 신호를 만들어 내고, 그 신호를 서브시스템에 인가하면서 SUT가 서브시스템을 제어할 수 있는 지를 테스트한다. 그리고 서브시스템은 인가된 신호로 동작하고 서브시스템은 출력을 SUT에 피드백 한다. 이 피드백 신호는 실행 테스트 케이스 생성 모듈에도 전달되어 T_{k+1}에 사용될 실행 테스트 케이스 생성에 사용된다.

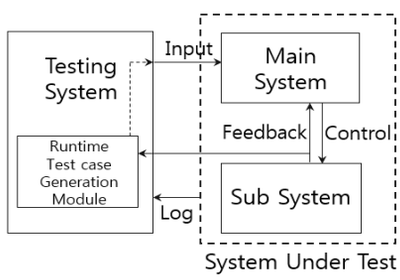


Fig. 3. Runtime Test Case Generator Configuration

이와 같이 서브시스템의 피드백을 받아 상황에 맞는 실시간 테스트 케이스 재구성할 수 있어야 하는 조건을 고려할 때, 기존의 HILS(Hardware-In-the-Loop-Simulation)나 MILS(Model-In-the-Loop-Simulation)와 같은 구조들은 실행 테스트 케이스 생성을 위한 플랫폼 구조로는 적합하지 않다[13, 14].

본 논문에서 제안하는 구조는 테스트 시스템 내부에 모델을 탑재한 실행 테스트 케이스 생성 모듈을 이용하여 실시간 피드백에 대한 테스트 케이스 재구성하는 방법이다. Fig. 4는 테스트 케이스 생성 내부 구조를 보여주고 있다.

기존의 테스트 케이스 입력 방식에서는 미리 작성되어진 비실행 테스트 스크립트를 미리 저장하여 두고, 피드백과 무관하게 순차적으로 테스트를 진행하는 방식이다. 하지만 본 논문에서는 모델에서 서브시스템 피드백 값으로서 생성하여 테스트 케이스를 재구성하는 방법이다. SUT의 실행 시간에 맞추어 테스트 스크립트가 입력되어야 하므로 제안하는 시스템은 RTOS(Real-Time Operating System) 상에서 동작한다.

실시간 테스트 케이스 재구성할 수 있는 모듈을 RTMS(Real-Time Model Service)라 칭한다. RTMS의 구조는 Fig. 4와 같이 테스트 케이스 재구성 엔진(TC Regen), CFG(Configuration), FDBK 모델 DLL(Dynamic Linked Library) 및 Test case table

로 이루어져 있다.

Test case table에는 사전에 만들어진 테스트 케이스가 저장되어 있다. 하나의 테스트 케이스는 피드백과 무관하게 만들어져 있으며, 피드백과 관련된 값은 임의로 구성되어 있다. 테스트 케이스는 테스트 용도에 따른 테스트 케이스 생성 정책에 따라 만들어진다. 테스트 케이스 생성은 본 논문에서 범위를 벗어나므로 여기서는 거론하지 않는다. (본 논문은 피드백 관련 테스트 케이스 생성에 관한 것이다) 피드백과 관련된 값은 TC Regen이 서브시스템의 피드백을 받아 FDBK 모델을 사용하여 적절한 값으로 바꾸어 임의로 작성된 피드백 값을 치환하여 하나의 완전한 실행 테스트 케이스가 재구성된다. 그리고 이 테스트 케이스는 다음 실행 시, SUT의 테스트 케이스로 사용된다.

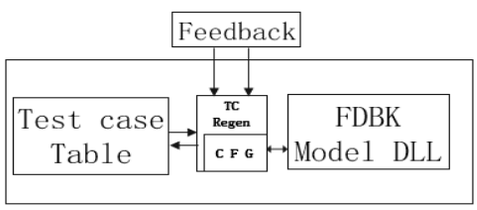


Fig. 4. Architecture of Real-Time Model Service

Fig. 5는 일반적으로 사용하는 비실행 테스트 케이스와 실험에서 진행 될 실행 테스트 케이스의 차이를 보여준다. Fig 5(a)를 미리 생성되어진 비실행 테스트 케이스의 예라 가정하자. 실행 테스트 케이스는 두 가지 유형으로 만들어 질 수 있다. 첫 번째는 비실행 테스트 케이스에 피드백을 활용하여 재구성하여 추가된 부분을 가진 테스트 케이스이며 (Fig 5(b)), 두 번째는 비실행 테스트 케이스와는 전혀 무관하게 피드백만을 이용하여 전체를 재구성한 테스트 케이스로 나타낼 수 있다(Fig. 5(c)). (Fig. 5의 점선으로 표기된 테스트 입력은 테스트 실행 중 피드백을 참조하여 생성된 실행 테스트 입력이다)

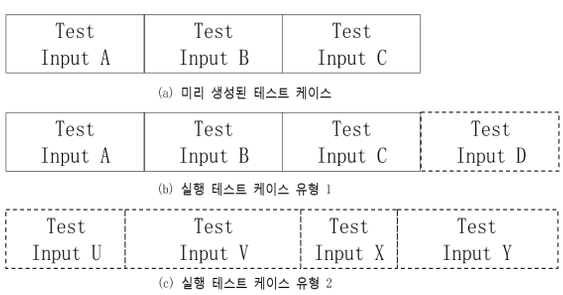


Fig. 5. Test Case Types

Fig. 4의 CFG는 SUT의 피드백과 모델 DLL을 연결시켜주는 인터페이스 정보를 담고 있다. 즉, DLL에 관한 모델의 개수, 데이터 타입, 통신 주기, 변수 정보가 담겨 있는 파일로서 CFG에 작성된 정보로 모델 DLL의 정보를 파악할 수 있다.

FDBK model DLL은 피드백과 관련된 시스템 기능 모델이 DLL로 변환한 모듈이다. TC Regen에서 받은 서브시스템의 피드백 값을 받아 다음 테스트 케이스에서 요구하는 필요한 피드백 관련 값을 생성한다.

테스트 케이스 재구성엔진 TC Regen 모듈은 실제 서브시스템의 피드백 값을 받아 FDBK model DLL에 입력하여 실행 시키고, 그 결과를 받아 대응되는 값을 다음 테스트 케이스(Test Case Table)의 값으로 치환하여 테스트 케이스를 재구성한다. Fig. 6은 실행 테스트 케이스 생성 순서도를 보여 준다.

테스트 케이스 생성 순서는, (1) 테스트 케이스 생성 시작 명령을 Test case table에 있는 다음 테스트 케이스를 테스트 케이스의 명령 형태인 테스트 스크립트로 변환하여 테스팅 시스템을 통하여 메인 시스템으로 보내면, (2) 메인 시스템(Main system)은 테스트 스크립트를 입력으로 하여 동작하게 되고 서브시스템을 동작 시킬 필요가 있으면, 서브시스템에 명령을 내린다. (3) 서브시스템은 입력에 따라 동작하고 그 결과를 RTMS(Real-Time Model Service)에 피드백한다. (4) RTMS는 피드백을 받아 FDBK model DLL에 입력하여 모델을 시뮬레이션하고 그 결과를 Test case table의 다음 테스트 케이스의 피드백 부분에 저장하여 다음 수행 테스트 케이스를 재구성한다. (5) RTMS에서 생성된 실행 테스트 케이스를 실행 시켜 다음 테스트 케이스 생성을 반복한다.

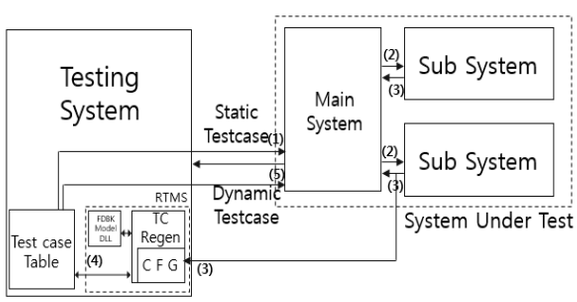


Fig. 6. Sequence of Runtime Test Case Generation

4. 실 험

4.1 실험 환경 구성

본 실험에서는 제안한 실행 테스트 케이스 생성 방법의 가능성을 보이기 위해 SUT로 사용될 임베디드 시스템이 서브시스템인 모터를 제어하고 모터(서브시스템)의 피드백을 받아 실행 테스트 케이스를 생성하는 실험을 진행한다.

서브시스템의 피드백을 받아 실시간으로 처리하여, 실행 테스트 케이스로 사용하기 위해 RTOS가 필요하다. 실험에서는 NI사에서 제공하는 Labview Real-time OS 12.0[12]를 RTOS로 사용하였으며, 기타 테스팅 시스템의 구성은 Table 1과 같다.

Table 1. Components of the Implemented Test Case Generator

Part	Model
CPU	Intel Core i7 (Sandy Bridge Architecture)
RAM	Samsung 2GB x2
HDD	WD 1TB
Ethernet	Intel PRO/100+
OS	NI Labview Real-time OS 12.0
DAQ Card	NI PCI-MIO-16E-4, NI PCI-8430
SUT	MTRONIX MK-A1
모터 (서브시스템)	MTS-R350se

4.2 모의 미사일 시스템을 이용한 실행 테스트 케이스 검증

테스트하고자 하는 SUT는 모의 미사일 시스템으로서 서브시스템은 모터와 고도센서를 대체하는 모델로 구성되어 있다. 모터의 입력은 테스트 케이스에 의한 값으로 구동되고, 모터의 출력은 속도를 피드백 한다. 모터로부터 피드백된 속도 값은 RTMS의 입력으로 들어가 현재의 고도를 계산하고, 현재 고도에 대한 모터의 입력 값을 실시간 피드백에 맞는 테스트 케이스로 재구성(실행 테스트 케이스)한다. 생성된 실행 테스트 케이스는 다시 모터로 입력되어 모터를 구동한다. 즉, 기존 HILS에서 미리 생성되어진 테스트 케이스를 이용하여 순차적으로 실험을 진행하는 것과는 다르게 실시간 상황에 따라서 테스트 시나리오를 재구성하여 테스트를 진행할 수 있다.

Table 2는 실행 테스트 케이스를 생성하기 위한 모델의 고도(Altitude)에 따른 모터 구동 출력 값(Motor Output)을 나타낸다. Fig. 9는 Table 2의 값을 구현한 모델이다.

Table 2. Motor Output to Altitude

Altitude	Motor Output
400 - 5,000	Motor_Stage1
5,000 - 9,000	Motor_Stage2
9,000 - 18,000	Motor_Stage3
18,000 - 30,000	Motor_Stage4
30,000 이상	Motor_Stage5

실행 테스트 케이스 생성을 위해서는 먼저 테스트 시스템과 SUT 그리고 모터 피드백을 받기 위한 하드웨어 연결이 이루어져야 한다. 그 전에, 하드웨어 신호를 테스트 케이스 생성 플랫폼에서 테스트 입력으로 사용할 변수에 할당하여야 한다. Fig. 7은 본 실험에서 사용한 플랫폼의 할당 변수이다. “Name”은 변수명이며, “Scope”에서 “SYSTEM_INPUT_DATA”는 SUT 입력이며, “SYSTEM_OUTPUT_DATA”는 SUT 출력이다. “Variable ID”는 테스트 케이스 생성 플랫폼이 자동으로 생성한 내부 변수 ID이다.

초기 테스트 케이스인 비실행 테스트 케이스와 초기 피드백 값으로 임의로 할당된 테스트 케이스(Fig. 10)로 SUT를 구동하고 그 이후부터는 모터의 피드백(MT_MotorFeedback)을 모델(Fig. 9)로 받아 Table 2에 대응되는 Motor_Output을 실시간 환경에 맞게 테스트 시나리오를 재구성하며 테스트를 진행한다.

SUT 모터의 피드백 값은 모터의 속도이며, 이 속도 및 현재의 고도를 이용하여 고도를 계산하고, 계산된 결과가 해당되는 고도 구간에 맞는 모터의 속도를 계산하여 다시 출력한다. 정해진 시나리오대로 동작하는 기존 테스트와는 다르게, 실시간 상황에 맞는 값을 계산하여 재생성된 테스트 케이스를 사용한다.

테스트 케이스를 재구성과 동시에, 테스트가 진행되는 중에 테스트 케이스 생성 플랫폼에서는 실시간으로 결과 값을 측정하여 로그(Log)로 남긴다. 구동이 종료되면 로그를 분석하여 실행 테스트 케이스를 확인할 수 있다.

Fig. 7에서 보는 바와 같이 SUT를 동작시키기 위해서 총 18개의 입력 및 출력 변수가 필요하다. 이들 중 실행 테스트 생성에 필요한 변수는 3개이다. 이들은 모터 구동에 필요한 입력 ‘SysIn_MotorControl’, 모터가 구동 된 후 모터의 속도 값을 피드백하는 ‘MT_MotorFeedback’, 피드백 값을 모델로 입력 후 모델 내에 있는 고도 계산 알고리즘을 실행시켜 얻은 현재 고도 ‘Present_Altitude’이다. 이들 중‘SysIn_MotorControl’은 ‘Present_Altitude’을 참조하여 SUT 내의 모터 제어 알고리즘에서 현재 고도에 맞는 출력을 제어하기 위해 실행 테스트 케이스 형태로 다시 모터 입력이 된다. 즉, ‘SysIn_MotorControl’은 자신의 입력으로 동작하는 모터의 피드백을 받아 다시 자신의 입력 값을 실행 시 마다 실시간 변하는 변수이다. 위 세 개의 변수 이외의 변수 값은 피드백으로 재구성되지 않음으로, 실험내용에서 설명을 생략한다.

	A	B	C
1	Name	Scope	Variable_ID
2	SysIn_SW1	SYSTEM_INPUT_DATA	16777216
3	SysIn_SW2	SYSTEM_INPUT_DATA	16777217
4	SysOut_Motor	SYSTEM_OUTPUT_DATA	16777218
5	SysOut_LED	SYSTEM_OUTPUT_DATA	16777219
6	SysIn_AccelerateSensor	SYSTEM_INPUT_DATA	16777220
7	SysIn_FiberSensor	SYSTEM_INPUT_DATA	16777221
8	SysIn_AltitudeSensor	SYSTEM_INPUT_DATA	16777222
9	SysIn_MotorControl	SYSTEM_INPUT_DATA	16777223
10	MT_AccelerateSensor	SYSTEM_OUTPUT_DATA	16777224
11	MT_MotorFeedback	SYSTEM_OUTPUT_DATA	16777225
12	MT_AltitudeSensor	SYSTEM_OUTPUT_DATA	16777226
13	MT_UltrasonicSensor	SYSTEM_OUTPUT_DATA	16777227
14	SysOut_AccelerateSensor	SYSTEM_OUTPUT_DATA	16777228
15	SysOut_Ultrasonic	SYSTEM_OUTPUT_DATA	16777229
16	In_AltitudeSensor	SYSTEM_INPUT_DATA	16777230
17	Motor_Stage	SYSTEM_OUTPUT_DATA	16777231
18	Present_Altitude	SYSTEM_OUTPUT_DATA	16777232
19	Test_Start	SYSTEM_OUTPUT_DATA	16777233

Fig. 7. Variable Name and ID

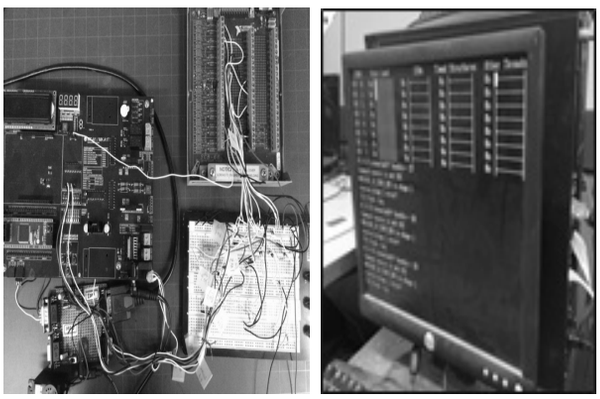


Fig. 8. The Implemented Testing System

Fig. 8과 같이 인터페이스 회로를 거쳐 SUT에 연결하면 실험 준비가 완료된다. 테스트 시작에서 먼저 'SysIn_MotorControl'를 제외한 테스트 스크립트(Fig. 10)를 준비한다. Fig. 10의 테스트 스크립트는 각 신호의 입력을 나타내는 'SysIn_*'으로 구성되어 있다. 여기서 'SysIn_MotorControl'는 임의의 초기 값(0.1)을 가지지만, SUT와 모터가 구동되면서 피드백을 값을 받아 현재고도와 피드백 값으로 다시 계산되어 치환되며 다음 실행 테스트 케이스의 일부가 된다.

Fig. 9는 현재의 고도 값(Present Altitude)과 모터의 피드백(MT_MotorFeedback)으로 다음 모터 입력 값(SysIn_MotorControl)을 계산하는 간단한 모델이며 Matlab사의 Simulink/Stateflow [20]를 이용하여 모델링되었다.

수행 결과는 NI사의 Labview에서 지원하는 TDMS파일형태[21]의 로그로 기록되었다. Fig. 11은 테스트가 종료 된 후의 TDMS 파일 형태의 로그 파일을 엑셀의 형태로 나타낸 내용이다. 'Present Altitude'와 'MT_MotorFeedback'에 따라 생성되는 모터 입력의 테스트 케이스가 재생성되어 'MotorStage'가 계산되어가는 과정을 보여주고 있다.

Fig. 10에서 'SysIn_MotorControl'은 비실행 테스트 케이스의 입력으로 인가해야 하는 값이고 Fig. 11의 'SysIn_MotorControl'은 실제 실행 테스트 케이스로 인가된 결과 값을 나타낸다. 테스트가 시작되면 Fig. 10의 'SysIn_MotorControl'의 0.1, 0.1, 0.1, 0.2, 0.3 순으로 메인 시스템에 입력 되고, 입력에 따른 모터의 피드백 값이 Fig. 9의 'MT_MotorFeedback'으로 입력된다.(모터의 피드백 값은 실제 SUT의 알고리즘에 따른 값이므로 블랙박스 테스트에서는 알 수 없다) 입력된 피드백 값은 모델(Fig. 9)에 따라서 실행 테스트 케이스가 생성되고 Fig. 11의 'SysIn_MotorControl'의 0.2, 0.2, 0.2, 0.3, 0.4, 0.4 순으로 재생성된다.

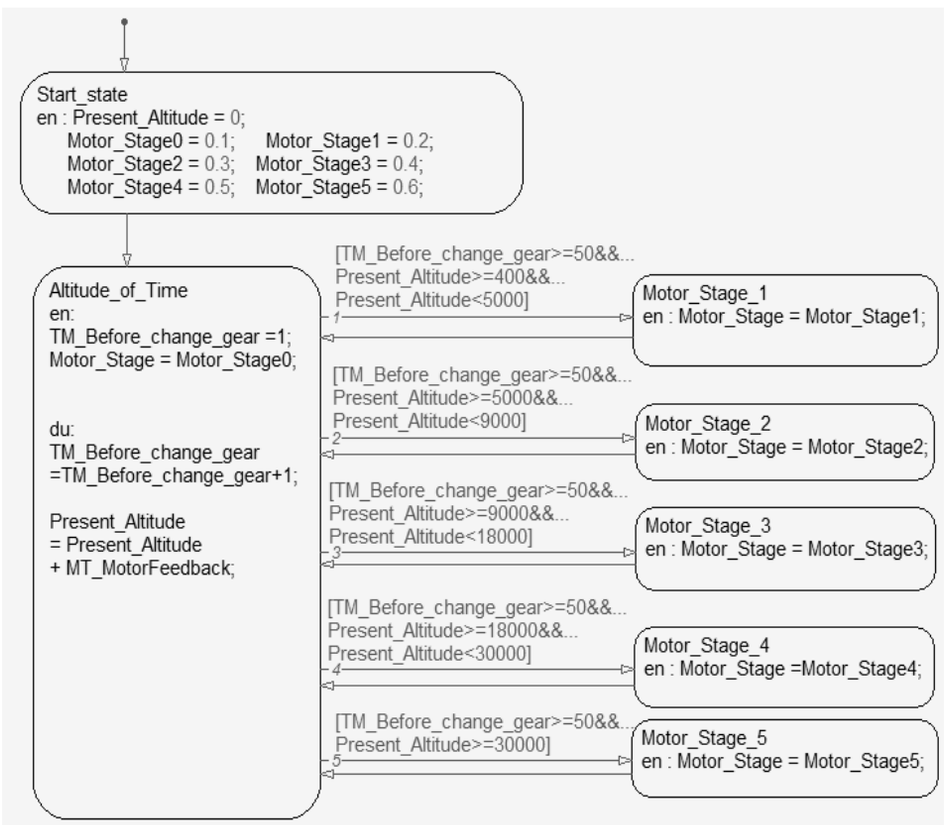


Fig. 9. Model for Calculating 'SysIn__MotorControl' Referencing 'Present Altitude'

SysIn_MotorControl	SysIn_SW1	SysIn_SW2	SysIn_FiberSensor
0.1	0	0	0.1
0.1	0	1	0.15
0.1	1	0	0.3
0.2	0	0	0.1
0.3	0	0	0.25
0.3	1	0	0.2
0.3	0	0	0.1
0.4	0	0	0.1
0.4	0	0	0.2
0.5	0	0	0.1

Fig. 10. Static Test Script

SysIn_MotorControl	Present_Altitude	SysIn_SW1	SysIn_SW2	SysIn_FiberSensor
0.2	1324	0	0	0.1
0.2	2806	0	1	0.15
0.2	4366	1	0	0.3
0.3	7761	0	0	0.1
0.4	10243	0	0	0.25
0.4	13275	1	0	0.2
0.4	16475	0	0	0.1
0.5	22975	0	0	0.1
0.5	26783	0	0	0.2
0.6	30704	0	0	0.1

Fig. 11. Test Output Log

4.3 기존 HILS의 테스트 케이스 사용과 제안한 환경에서의 테스트 케이스 사용의 차이

위의 시나리오를 보게 되면, 기존 테스트 케이스(Fig. 10)의 경우에는 실시간 환경을 고려하지 않고 작성한 테스트 케이스이고, 재구성 테스트(Fig. 10)는 실시간 환경을 고려하여 시나리오에 맞게 재구성되어 실행된 테스트 케이스이다.

현재의 데모보드를 이용한 실험은 순차적으로 발생하는 피드백을 이용하여 테스트 케이스를 생성하였지만, 복잡한 피드백을 이용하여 테스트 케이스를 생성하는 모델을 설계하게 되면, 실시간 환경에 맞는 테스트 케이스가 생성되며, 이는 테스트 플랫폼에 적용시킬 수 있는 환경이 구성된다. 또한 이와 같이 재구성되어진 테스트 케이스는 SUT 알고리즘에 대해서 유동적으로 대응하여 생성되는데, 이러한 점은 기존 HILS의 결과를 단순히 Log로 남기는 것과는 다르다는 것을 알 수 있다.

Table 3에서는 실험에서 진행된 비실행 테스트 케이스와 실행 테스트 케이스의 차이점을 보여준다. 처음의 피드백을 고려하지 않고 생성한 Table 3의 왼쪽 비실행 테스트 케이스의 경우, Table 2의 고도-모터 표의 대응되지 않게 모터가 제어 된다. 하지만 Table 3의 오른쪽 실행 테스트 케이스 결과는 고도에 대응되는 모터 제어 값이 입력되었음을 알 수 있다. 만약 테스트를 진행하면서 오류와 같이 원하지 않는 방향으로 테스트가 진행될 경우에는 실행 테스트 케이스와 비실행 테스트 케이스는 더욱 명확한 차이를 나타낼 것이다.

Table 3. Static and Run-Time Test Cases

Static test case (Fig. 10. SysIn_MotorControl)	Run-time test case (Fig. 11. SysIn_MotorControl)
0.1	0.2
0.1	0.2
0.1	0.2
0.2	0.3
0.3	0.4
0.3	0.4
0.3	0.4
0.4	0.5
0.4	0.5
0.5	0.6

예를 들면 Present_Altitude의 값(테스트 실행 중 자동으로 생성된 피드백) 7761(Fig. 11의 4번째 입력 참조)의 경우 비실행 테스트 케이스(Table 3 왼쪽) 입력을 보게 되면 '0.2'를 실행하고 있다. Table 2를 참조하면 '5000-9000'에서의 모터 입력 값은 0.3이 되어야 한다. 즉, 비실행 테스트 케이스를 사용하게 되면 고도에 대응되는 모터 입력 값을 정확하게 출력할 수 없다. 하지만 실행 테스트 케이스의 경우(Fig. 11의 SysIn_MotorControl)에는 고도-모터 입력 값에 대응되는 '0.3'이 입력됨을 알 수 있다.

5. 결 론

본 논문에서는 피드백이 있는 임베디드 시스템의 테스트에 사용하는 테스트 케이스 생성에 대한 방법을 제시하였다. 기존의 테스트 케이스 생성 방법으로는 피드백은 참조하지 않고 테스트 케이스를 생성하여 서브시스템의 동작과는 무관한 테스트 케이스를 생성하는 반면 본 논문에서 제안하는 방법에서는 실시간 환경의 피드백을 참조하여 테스트 시나리오를 실시간 재구성하는 방법을 사용한다.

기존의 테스트 방법은 메인 시스템의 피드백을 실제 테스트에 참조하여 테스트 케이스를 재생성하거나 테스트 환경에 반영한다. 하지만 메인 시스템과 연결되어 있는 서브시스템의 피드백은 테스트의 결과인 로그로만 남기고 있고, 메인 시스템에 연결되어 있는 서브시스템의 피드백까지 고려하여 테스트하기가 어렵다. 그래서 기존의 테스트에서는 서브시스템의 피드백을 로그로 남기게 되고, 추후의 테스트에서 피드백을 참조로 테스트 케이스를 엔지니어가 수동으로 수정하여 피드백을 고려한 테스트를 진행한다. 하지만 본 논문에서는 서브시스템의 피드백을 MILS의 환경인 RTMS에 연결하여 실시간 테스트 케이스를 재구성하여 서브시스템까지 고려가 가능함을 실험을 통하여 증명하였다.

제안된 방법의 실용화를 위해서 보다 다양한 환경에서의

실험을 진행하여 구현 시 발생할 수 있는 문제를 정교히 해결할 필요가 있을 것으로 판단된다.

References

[1] Seungbae Sim and Cheonsoo Yoo, "Software Process Improvement for Defense System Safety," *Korean Institute of Information Scientists And Engineers, Conference Procedding*, pp.94-96, 2015.12.

[2] Wikipedia [Internet], http://en.wikipedia.org/wiki/Ariane_5_Flight_501.

[3] Wikipedia [Internet], https://en.wikipedia.org/wiki/Sudden_unintended_acceleration.

[4] Glenford J Myers, Corey Sandler and Tom Badgett, "The Art of Software Testing," Wiley 3 edition, pp.49-84, 2012.

[5] Glenford J Myers, Corey Sandler, and Tom Badgett, "The Art of Software Testing," Wiley 3 edition, pp.42-49, 2012.

[6] Janusz Rajski and Jerzy Tyszer, "Arithmetic Built-In Self-Test for Embedded Systems," Prentice Hall, 1997.

[7] M. Schlager, W. Elmenreich, and I. Wenzel, "Interface design forhardware-in-the-loop simulation," in *Proc. IEEE International Symposium on Industrial Informatics (ISIE 2006)*, Vol.2, pp.1554-1559, Montreal, Canada, July, 2006.

[8] Yervant Zorian, "Built-In-Self-Test," Microelectronic Engineering, Elsevier Publishers, BV., Amsterdam, NL, Vol. 49, No.1-2, pp.135-138, Nov., 1999, XP004182057, ISSN: 0167-9317.

[9] Paul Baker, Zhen Ru Dai, Jens Grabowski, Øystein Haugen, Ina Schieferdecker, and Clay Williams, "Model-Driven Testing Using the UML Testing Profile," Springer, pp.7-8, 2008.

[10] So-Young Jeong, "Test Case Generation Technique Based on State Transition Model for Embedded System," *Journal of Korean Institute of Information Technology,* Vol.9, No.4, pp.11-21, 2011.

[11] J. R. Noseworthy, "The Test and Training Enabling Architecture (TENA) Supporting the Decentralized Development of Distributed Applications and LVC Simulations," In *12th IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications*, pp.259-268, October, 2008.

[12] Maneesh Varshney, Kent Pickett, and Rajive Bagrodia, "A Live-Virtual-Constructive(LVC) Framework for Cyber Operations Test, Evaluation and Training," *Military Communications Conference,* pp.1387-1392, 2011.

[13] Sung-Chan Song, Young-Jin Na, and Tae-Hwan Yoon, "The Development of HILS and Test Equipment for Millimeter-Wave (Ka-Band) Seeker's Test and Evaluation," *The Journal of Korean Institute of Electromagnetic Engineering and Science*, Vol.23, No.1, pp.47-55, 2012.

[14] Soo-Jin Lee, "Development of a Hardware-in-the-loop Simulator for ABS ECU Using xPC Target and Virtual Reality Toolbox of MATLAB," *The Korean Society of Automotive Engineers*, Vol.3, pp.1493-1498, 2002.

[15] Yang Inseok, "Test Execution Machine Development Research to Perform a test of multiple," *The Institute of Electronics and Information Engineers*, Vol.37, No.1, pp. 1732-1734, 2014.

[16] Kyoung Jin Kim, "Reconfigurable test execution machine for embedded system," *KIPS Tr. Software and Data Eng.*, Vol. 3, No.7, pp.243-254, 2014.

[17] Paik Jun Hyun, "Test Executor and Testing Platform performance improvement suggestions using the UDP," *The Institute of Communication Sciences*, pp.1532-1533, 2015.

[18] A. P. Mathur, "Foundations of Software Testing," Pearson Education, 2008.

[19] National Instrument, RTOS [Internet], http://www.white-paper/3938/ko/.

[20] MATLAB, Simulink Stateflow [Internet], http://www.mathworks.com/products/stateflow, 1994-2016, The Mathworks Inc.

[21] National Instruments, TDMS [Internet], http://www.ni.com/white-paper/3727/ko/.

[22] Corina S. Pasareanu, Johann Schumann, Peter Mehlitz, Mike Lowry, Gabor Karsai, Harmon Nine, and Sandeep Neema, "Model Based Analysis and Test Generation for Flight Software," *Proceedings of the Third IEEE International Conference on Space Mission Challenges for Information Technology*, pp.83-90, July, 19-23, 2009.

[23] Ji-Hyun Lee, "Test Case Generation Technique for Interoperability Testing," *Journal of Korea Information Science Society*, Vol.33, No.1, pp.44-58, 2006.

최 우 원

e-mail : davewchoi7@naver.com

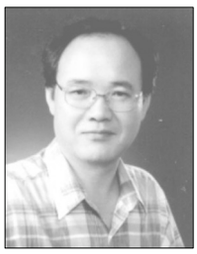
2015년 아주대학교 전자공학과(학사)

2015년~현 재 아주대학교 전자공학과 석사과정

관심분야 : 임베디드 시스템 설계, 임베디드 시스템 테스팅, 아날로그 회로, 임베디드 소프트웨어

정 기 현

e-mail : khchung@ajou.ac.kr
1984년 서강대학교 전자공학과(학사)
1988년 미국 Ilinois주립대 EECS(석사)
1990년 미국 Purdue대학 전기전자공학부 (박사)
1991년~1992년 현대반도체 연구소
1993년~현 재 아주대학교 전자공학과 교수
관심분야 : 임베디드 시스템 테스팅, 실시간 시스템



최 경 희

e-mail : khchoi@ajou.ac.kr
1976년 서울대학교 수학교육과(학사)
1979년 프랑스 그랑데꼴 Enseeiht대학 (석사)
1982년 프랑스 Paul Sabatier대학 정보공학부(박사)
1982년~현 재 아주대학교 소프트웨어학과 교수
관심분야 : 운영체제, 분산시스템, 실시간/멀티미디어 시스템