임베디드 소프트웨어 테스트 케이스 단계적 생성

장세헌, 장중순, 이상용, 고병각, *최경희, *박승규, **정기현, ***이명호
아주대학교 산업공학과 / *아주대학교 컴퓨터공학과
**아주대학교 전자공학과 / ***세명대학교 인터넷 정보공학부

Stepwise test case generation for embedded s/w

S.H. Jang, J.S. Jang, S.Y. Lee, B.G. Ko
*K.H. Choi, *S.K. Park, **K.H. Jung , ***M.H. Lee
Industrial Eng., Ajou Univ. / *Computer Eng., Ajou Univ.
** Electronics Eng., Ajou Univ. / ***Div. of Internet Information, Semyung Univ.

Abstract

Automatic test case generation for testing an embedded software is considered. Existing tools for test case generation such as finite state machine or mutant test usually adopt top down approach and depend upon graphical transition and decision table, which makes it difficult to find out where the bugs exist. Also it is hard to describe the special features of embedded systems such as concurrent execution of individual components.

Most of embedded systems interacts with the real world, receiving signals through sensors or switches and sending output signals to actuators that somehow manipulate the environment. Embedded software controls the entire system based on the logics such as interpreting the sensor inputs and making the actuators to start or stop their intended operation. This study proposes an automatic test case generation procedure that tests the system starting from the control logics of sensors, switches and actuators and then their concurrent execution controls, and finally the entire system operation. Such a stepwise approach makes it easy to generate test cases to tell where the bugs of embedded software exist.[1]

Keywords: Embedded Software, Test Case Generation

1. 서론

임베디드 시스템은 마이크로프로세서를 탑재하고 센서와 액추에이터를 통하여 환경과 지속적으로 상호 작용하는 시스템을 말한다. 활용 분야가 넓어져 핸드폰, 철도 신호 시스템, 미사일 추적 시스템 등에 사용되고 있다. (Broekman, Notenboom ,2003) 특히, 의료, 국방, 우주공학 등의 분야에서는 시스템 오작동이 곧 생명을 위협 할 수 있으므로 높은 신뢰성이 필수적이다. 임베디드 소프트웨어는 임베디드 시스템에 내장된 소프트웨어로 물론 이러한 신뢰성 향상을 하드웨어 신뢰성 향상과 소프트웨어 신뢰성이 보장 되어야 하며, 그를 위해서 소프트웨어 버그를 찾는 적합한 테스트가 필요하다.

소프트웨어 테스트는 크게 두 가지로 나뉘는데, 프로그램 소스나 내부 구조, 데이터에 대한 지식 없이 스펙에 근거하여 기능과 작동 상태를 테스트하는 블랙박스 테스트와 소스코드를 바탕으로 테스트 하는 화이트 박스가 있다. 일반적으로 화이트 박스는 분석에 대한 어려움으로 작은 모듈 단위 테스트에 사용되며, 전체 시스템을 대상으로 하였을 때는 일반적으로 블랙박스 테스트가 이루어 진다. (Shin, S. G. ,2003) 또, 사람이 소프트웨어를 동작 시키며 결과를 판단하는 수동 테스트와 테스트 시작 후 사람이 관여하지 않고 결과를 산출하는 자동화된 테스트가 있다. 본 논문에서는 완성된 전체 소프트웨어를 대상으로 자동으로

[1] 장세헌, 수원시 영통구 아주대학교 팔달관 229호 생산정보실험실 FAX: 031-219-1610 E-mail: twinkle @ ajou.ac.kr

테스트하는 자동 블랙 박스 테스트를 다루고자 한다.

자동 블랙박스 소프트웨어 테스트를 위해 반드시 필요한 것이 대상 시스템에 대한 모델링과 테스트 케이스 이다. 모델링을 통해 요구사항을 정형화 하고 컴퓨터가 이해할 수 있게 만들어주어 자동 테스트가 가능하게 하며, 테스트 케이스는 특정 조건을 부여하여 소프트웨어가 관련 요구사항을 만족하는지 확인하기 위해 필요한 입력 값으로 테스트의 최소단위 이며, 테스트 케이스 생성 방법과 어떤 테스트 케이스로 테스트하는 가에 따라 테스트 시간과 성과가 달라진다. (Shin, S. G., 2003)

지금까지 많이 사용된 모델링 방법론은 Finite state machine 방식으로 이것은 소프트웨어의 상태를 정의하고 작동 방식을 상태간의 변환으로 표현하고 있다. 그러나 다양한 다중 이벤트가 발생하는 임베디드 소프트웨어의 상태를 정의하는 것이 무척 어렵다. 그 외에도 Finite state machine 을 확장한 EFSM (Luay , Vaysburg, Korel , Bader ,2001), 과 SCR (Blackburn, Busser ,Fontaine ,1997), MSC(Cheng, K-T, Jou, J-Y ,1990) 을 활용하여 모델링을 하고 있지만, 이것들은 모두 임베디드 시스템이 가지는 많은 동시 실행 및 다양한 내외적 상호작용 과 사용자 컨트롤에 대해 반영하기 어렵다. (Cheng, K-T, Jou, J-Y ,1990)

위 모델링 방법론을 사용하여 테스트 케이스를 생성하는 방법론들은 모두 그래픽 기반의 트래블링이나 결정 테이블의 변수 조합을 사용하여 테스트 케이스를 생성하고 있다. 이것들은 소프트웨어 작동양식에서 적당한 케이스를 뽑아서 실행 과정에서 우연히 벅을 찾고자 하는 개념에서 벗어 나지 못하고 있다.

이와같은 문제를 해결하기 위해 임베디드 소프트웨어가 가지는 사용자 설정과 센서입력 값에 대한 액추에이터의 반응이라는 특수한 동작을 작은 단위로 나누어 모델링 하고, 이를 이용하여 작은 단위부터 점진적으로 범위를 넓혀나가 어디에 문제가 있는지 유추할 수 있도는 단계적 테스트를 제안 하고자 한다.

2. 단계적 테스트 케이스 생성

2.1 임베디드 소프트웨어

임베디드 시스템은 과거엔 실시간 시스템으로 많이 불렸을 정도로 가장 큰 특징은 환경에 대한 실시간 반응이다. 기본 구조는 Figure 1 과 같이 키 입력을 포함한 센서단에서 입력을 받아 마이크로 프로세서의 컴퓨팅을 통해 액추에이터를 제어한다. 컨트롤은 일반적으로 사람이 제어하는 오픈 – 루프 컨트롤과 센서에 대한 액추에이터에 만응에 사람이 관여 하지 않는 클로우즈 루프 컨트롤로 나뉜다.

이런 시스템을 움직이는 임베디드 소프트웨어는 다양한 루프 컨트롤로 인한 많은 동시 실행 속에 내외적 상호 작용이 혼재하여 모델링과 테스트 수행의 어려움을 제공한다.

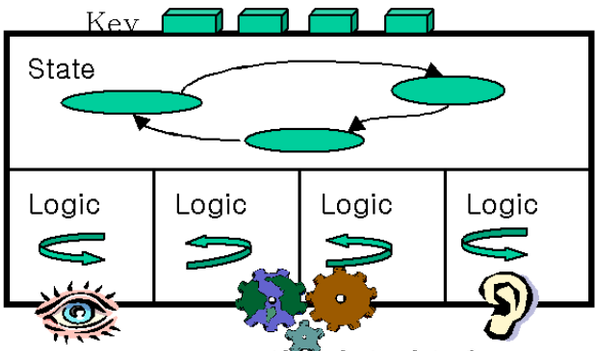

Figure 1. 임베디드 시스템

2.2 FSM 과 테스트 케이스 생성의 문제

FSM 은 Figure2 처럼 소프트웨어 모델링을 상태를 나타내는 노드와 상태변화를 노드 간의 링크로 나타낸다. 그러나 임베디드 시스템이 가지는 특성에 대해 각 상태를 정의 하기가 어렵다. 또, FSM 을 바탕으로 한 일반적인 테스트 방법론은 그래프 기반으로 노드와 링크에 대한 트래블링 문제를 풀어 최소의 테스트 케이스로 모든 노드와 링크를 커버하는 것에 중점을 두고 있다. 이와 같은 테스트는 다층적인 성격을 가진 임베디드 시스템에서는 문제가 발생을 한다고 해도 정확히 어디서 시작 된 것인지 추측하기가 어렵다.

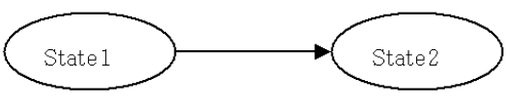

Figure 2. FSM

2.3 단계적 테스트

임베디드 소프트웨어는 전술했듯이 많은 동시 작동과 상호 작용이 존재 하며 각각 상호 작용을 하지만 독립적이다. 따라서 전체 소프트웨어를 하나로 묶어 모델링 하기에는 많은 어려움이 따른다. 그래서 각각의 기능과 알고리즘을 작게 나누어 모델링을 한다. 전체 소프트웨어의 작동은 각각 단위의 상호

작용으로 표현한다. 이 작은 단위를 로직이라 부른다.

우리가 제안하는 단계적 테스트는 로직을 기반으로 하드웨어 신뢰성 테스트 개념을 도입하여 Figure 3 에서 나타낸 작은 단위에서 시작하여 상호 관계로 넓혀 나가 테스트 하는 것으로 고장 발생 위치와 원인을 밝힐 수 있다.

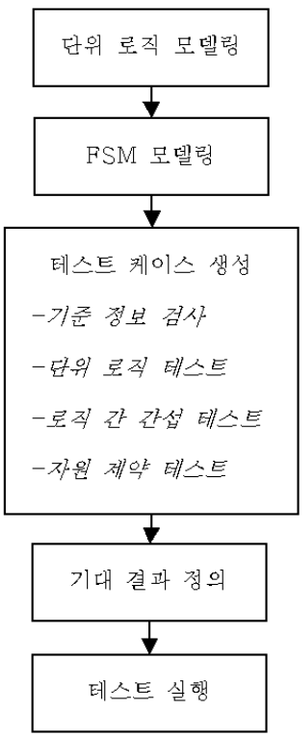

Figure 3 단계적 테스트

3. 구체적인 테스트 방법론

3.1 단위 로직 모델링

여기서 정의하는 로직은 스펙에서 정의하는 최소단위의 알고리즘이다. 예를 들어 가정용 에어컨에서 사용자가 키를 눌러 설정한 온도와 현재 온도를 비교하여 바람의 강도를 조절하는 것, 바람의 차가운 정도를 조절하는 것과 램프를 키를 누르면 등이 켜지는 것 등이 각각 단위 로직이 된다.

본 모델링의 목적은 테스트 의 정답을 만드는 것이다. 단위 로직에 있어서도 실제 구현에 중심을 두기 보다는 관찰자 시점에서 if.. then.. 개념으로 묘사된다. 즉 단지 로직이 작동되는 조건과 끝나는 조건, 그리고 작동 중에 조건에 따라 일어나는 사건을 기술 하는 것이다. 이런 로직 정의들을 통해 조건 별 소프트웨어의 작동 양태를 추출할 수 있다.

3.2 2 스테이지 모델링

임베디드 시스템에서 사용자 입력단과 외부환경에 반응하여 컨트롤이 수행되는 부분은 분리되어 있으며 작동 양식도 다르다. 이것을 무리하게 하나로 묶기에는 어려움이 따르므로 각각에 적합한 다른 방식으로 모델링을 구현 하였다. 이것을 2 스테이지 모델링이라 부른다.

사용자 입력 단인 키의 상태는 윈도우 어플리케이션과 같은 메뉴-드리분 방식에 적합한 방식 (Beizer,1995)인 finite state model 을 사용한다. 즉 Figure 1 과 같이 키의 상태는 Finite state machine 으로 스펙을 정리하고 내부 작동을 로직으로 표현하는 것이다.

Finite state machine 에 따라 로직이 작동되는 사용자 정의 환경이 결정이 되고, 로직은 사용자가 정의한 키 상태와 센서로부터 얻어 지는 환경에 조건에 따라 반응하게 된다.

3.3 기준 정보 검사

일반적으로 임베디드 시스템은 센서값에 대한 변환 테이블 등의 기준 정보를 가지고 있다. 여기서는 기준 정보들이 정확하게 설정이 되어 있는지 검사 하는 것이다.

중간에 값이 생략 되었다던가, 혹은 잘 못 입력 된다면 다른 부분이 정상적인 상태에서도 잘못된 컨트롤이 발생할 수 밖에 없다. 모든 소프트웨어 작동의 기반이 되는 데이터 정보를 사전에 점검한다.

3.4 단위 로직의 알고리즘 테스트

이 테스트는 로직에 정의된 각 컨디션 별로 소프트웨어를 돌려 보는 것이다. 이것은 기본적인 알고리즘에 오류가 있는가에 대한 테스트를 실시하는 것으로, 만약 여기서 고장이 발생한다면 알고리즘에 문제가 있다고 판단 할 수가 있다.

일반적으로 거론되는 단위 테스트는 구조적 모형에 기반을 둔 모듈 테스트 이지만, 단위 로직 테스트는 스펙 기반의 논리적인 단위를 테스트 하는 것으로 블랙 박스에서 적용할 수 있다.

테스트 케이스는 각 컨디션을 모두 실행 시킬 수 있는 입력 값의 집합이 된다.

3.5 로직간 간섭 테스트

로직간 간섭에 대한 테스트 이다. 기존의 인티그레이션 테스트는 단위 모듈 간의 인터페이스만을 테스트 하는 것이었다.

이 테스트는 로직간에 발생 할 수 있는 동일한 메모리 주소 공유, 공유 디바이스에 대한 점유와 이전이 원활이 이루어지지 않아 발생하는 문제들을 발견 하고자 한다. 모든 로직간의 관계를 볼 수 있지만, 내장형

소프트웨어의 특성상 로직의 입력과 출력은 센서 디바이스와 액추에이터 디바이스이다. 로직간 동일한 디바이스에 대한 메모리 영역을 다룰 때 문제가 발생할 소지가 높기 때문에 동일한 디바이스를 사용하는 것에 대해서만 실시하여도 충분히 벅을 잡을 것으로 보인다.

여기서 고장이 발생한다면 이전 단계에서 단위 로직 알고리즘에 대한 문제는 테스트 하였으므로 로직간 교체 중 발생한 문제라고 추정할 수 있다.

테스트 케이스는 각 로직 단위 테스트 케이스를 활용하여 동일한 디바이스를 공유하는 로직간 교체가 발생 하도록 한다.

3.6 자원 제약 테스트

임베디드 소프트웨어의 자원은 경제성 등의 이유로 극히 제한적이다. 이런 자원이 제약이 프로그램 활동에 문제를 발생 시킬 수 있다. 메모리 할당이나 가비지 문제와 같은 이런 문제는 단순히 하나의 사건으로 발생하는 문제가 아니라 작은 문제들의 누적으로 발생한다. 테스트 방법은 해당 자원에 더미를 쌓아 놓거나 임의로 용량을 줄이는 등 가혹 조건을 제공하여 자원의 관리 능력을 테스트 하는 것이다.

3.7 기대 결과 정의

지금까지 로직에 대한 개념과 모델링을 살펴 보았다. 그럼 단위로직을 바탕으로 전체 소프트웨어의 작동을 표현하는 기대 결과 생성에 대해 이야기 하겠다.

우선 기대 결과 생성을 위한 전체 구조를 설명 하겠다. Figure 4 처럼 FSM 과 로직을 정의 하는데 있어 사용된 변수들을 입력으로 넣고 FSM 과 로직을 통해 각 변수들의 값을 바꾸어 기대 변수 값을 정하게 된다.

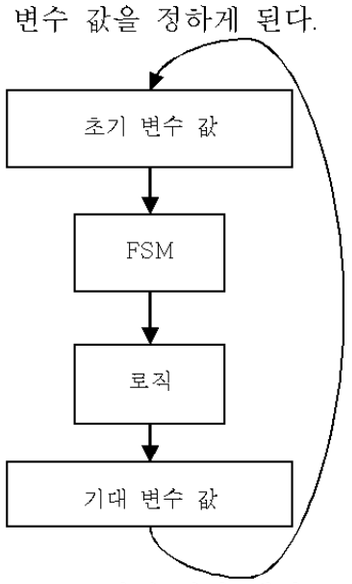

Figure 4 기대 결과 생성

이와 같은 구조 속에서 각 로직의 연관성은 두 가지로 나눌 수 있다. 하나는 로직의 출력이 다른 로직의 입력이 되는 경우이다. 다른 하나는 동일한 액추에이터의 변수 값을 설정 하는 것이다. 두 가지 모두 로직의 순서를 적당히 배치 함으로써 해결할 수 있다. 즉, 입력과 출력의 관계는 입력을 제공하는 로직을 앞 단에 배치하고 동일한 변수를 가지는 로직은 우선 순위가 높은 것을 뒤에 배치하여 덮어 쓰기를 통해 우선 순위가 높은 로직에 의한 값이 최종 결과 값에 반영 된다.

4. 결론

본 논문은 일반적인 임베디드 소프트웨어 방법론들이 가지는 문제점인 모델링의 어려움과 테스트케이스 생성시 테스트 결과로부터 찾아낸 고장의 발생 원인 위치 파악이 쉽지 않다는 문제를 해결하기 위해 단위 로직이라는 개념과 단계별 테스트 전략을 소개 하였다.

이 전략의 성패는 정확한 고장의 예측이다. 이를 위해 앞으로 임베디드의 소프트웨어 고장 모드에 대한 연구가 필요하다.

Reference

Arnold S. Berger(2002) , Embedded Systems Design, CMP Books

Bart Broekman, Edwin Notenboom (2003), Testing Embedded software, ADDISON-WESLEY, pp 5~6

Boris Beizer, (1995)"Black-Box Testing," John Wiley & Sons, Inc., pp 204

Cheng, K-T, Jou, J-Y (1990), Functional test generation for finite state machines, Test Conference, IEEE, pp 162-168

Shin, S. G. (2003), Software test expert technique, TTA

Luay H, Boris Vaysburg, Bogdan Korel ,Atef J. Bader (2001), Requirement-based automated black-box test generation , Computer Software and Applications Conference, 2001. COMPSAC 2001. the 25th Annual International, IEEE, pp 489-495 , 2001

Mark R. Blackburn, Robert D. Busser , Joseph S. Fontaine (1997), Automatic Generation of Test Vectors for SCR – Style specifications, Computer Assurance, 1997. COMPASS '97. 'Are We Making Progress Towards Computer Assurance?'. Proceedings of the 12th Annual Conference, IEEE, pp 54-67

Nam Hee Lee, Sung Deok Cha (2003), Generation test sequences form a set of MSCs, Computer Networks, ELESEVIR, pp 405-417