신뢰성응용연구
제7권, 제4호. pp.137-148, 2007

고장메커니즘 기반의 임베디드 시스템 SW 테스트 방법에 관한 연구

정시영*, 장중순*, 이상용**

아주대학교 산업공학과*, 국방기술품질원**

A Study of Testing Embedded System Software Based on Failure Mechanisms

Si young Jeong*, Joong soon Jang*, Sang yong Lee**

Department of Industrial Engineering, Graduate school of Ajou University*,
Defense Agency for Technology and Quality**

Abstract

Rapid increase of embedded systems in electronic and mechanical control systems requires reliable and error-free embedded software. State-based testing methods like FSM are usually used to assure the reliability of embedded software. However, because of possibility of explosion of test cases, only partial test cases are considered in practical tests, which cannot guarantee that all the possible errors are investigated. This study proposes a test procedure based on failure mechanisms that may occur in embedded systems, which can not only assure that certain kinds of possible errors are detected but reduce the testing time. The proposed procedure is applied to vehicle air control system.

key word : Embedded SW testing, failure mode, failure mechanism, test methods

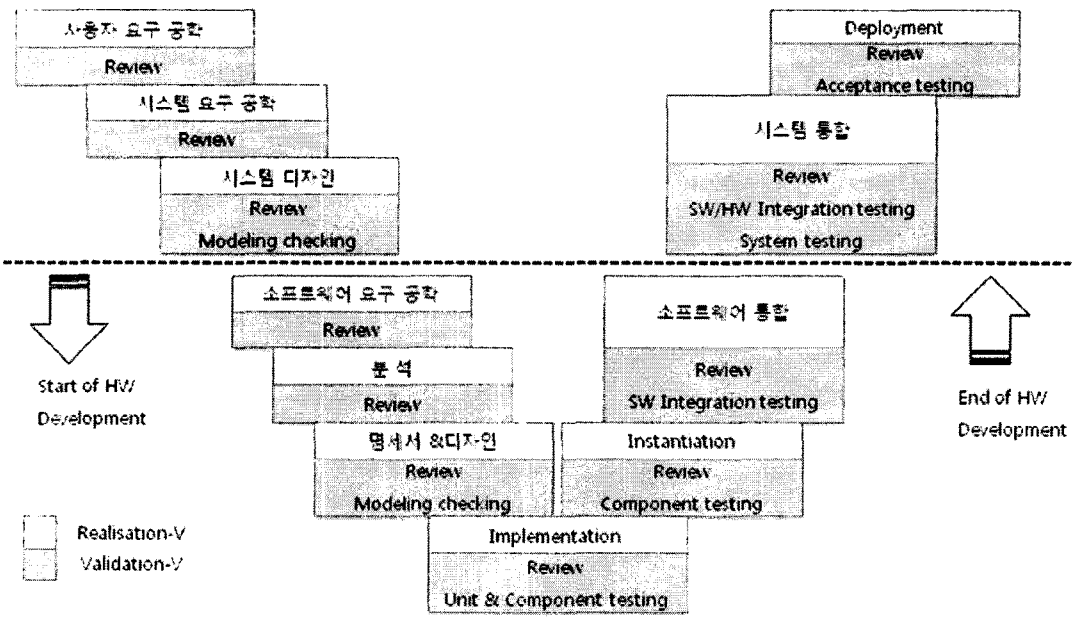
1. 서론

전자부품의 비약적인 기술발전으로 인하여 과거 군수산업, 산업용기계, 항공우주산업 분야에 국한되어 왔던 임베디드 시스템은 가전, 개인 이동통신 단말 등의 사회 전분야로 확산

되고 있다. 과거 단순 기능만이 구현되어 HW의 일부로 취급되었던 펌웨어 수준의 임베디드 SW는 현재 임베디드 시스템의 성능 및 가치를 결정지을 수 있을 만큼 매우 중요한 요소가 되었다. 고도의 HW 기능을 이끌어 내기 위해 SW는 매우 복잡해 졌으며, 이에 따른 잠재적인 오류의 가능성도 높아져, 한정된 개발기간 내에 가능한 많은 오류를 검출하기 위한 체계적인 테스트가 중요한 이슈로 부각되었다.

임베디드 SW의 테스트도 일반 SW와 마찬가지로 체계적인 품질 보증을 위해 개발단계에 따른 검증과 입증(Verification & Validation: V & V)의 방법을 사용한다. 검증은 프로그램이 주어진 명세서와 일치하는가를 점검하는 것으로 제품이 올바르게 만들어가고 있는가를 보는 것이며, 입증은 프로그램이 실제 사용할 수 있겠는가를 점검하는 것으로 고객의 요구사항이 만족되었는가를 보기 위한 것이다. 검증과 입증은 개발 단계별 산출물들의 검토 및 테스트를 통해 수행된다.

그림 1에서 보는 바와 같이 임베디드 SW는 HW와 통합 이전까지는 에뮬레이터와 같은 작동환경과 다른 시스템을 사용하여 개발하기 때문에 시간적인 제약사항, 반응시간, 전송지연, 메모리 크기, 버퍼제한, 요구되는 이벤트의 순서 등의 제약사항들이 충분하게 고려되기가 어렵다.[10] 따라서 임베디드 SW는 작동환경에서의 V&V중의 하나인 시스템 테스트가 매우 중요하며, HW와 SW의 통합이전에 유닛/컴포넌트 테스트가 시스템 수준에서 재 수행될 필요가 있다.



<그림 1> 내장형시스템 개발과정 및 SW 검증과 인증 과정[11]

시스템 테스트는 다양한 외부환경요인과 복잡한 SW 데이터의 흐름을 고려해야 하기 때문에 코드기반의 화이트박스 테스트[2, 16] 보다는 명세서 기반의 블랙박스 테스트[2, 13, 18]가 주로 사용된다. 블랙박스 테스트를 위해서는 명세서를 기반으로 모델링 후 모델에 따

른 테스트를 수행한다. 블랙박스 테스트는 구축된 모델을 바탕으로 입력대비 출력을 확인하기 위한 테스트케이스를 생성하여 오류를 검출하게 된다. 더욱이 상업용 임베디드 SW의 경우 개발기간이 매우 한정적이기 때문에 테스트 시간을 줄이면서 오류 검출도를 향상 시킬 수 있어야 한다. 그러므로 모델의 구조에 따라 모델이 가진 모든 경로에 대하여 테스트를 주어진 시간 내에 수행하면서 오류 검출도를 향상시킬 수 있는 최적화된 테스트케이스 생성이 무엇보다도 필요하다.

블랙박스 테스트케이스 생성을 위해 주로 그래프, 도표 등을 활용한 모델기반의 테스트 방법론이 활용되며, 특히 FSM[6], EFSM[5], Statechart[9, 17], CoRE[7]등과 같은 상태기반의 모델링 기법들이 많이 사용된다. 이러한 기법들은 SW의 오류를 검증하는데 탁월한 효과를 가지고 있음이 알려진바 있다.[4] 그러나 이러한 기법들은 명세서가 시스템의 모든 상황을 표현하지 못하는 경우가 있기 때문에 모델이 커버할 수 있는 오류만을 찾을 수 있다라는 단점이 있다. 또한 상태를 어떻게 정의할 것인가에 따라 모델의 형태가 달라지고, 상태 폭발 및 전이 폭발의 위험이 있는 테스트 케이스가 많아진다. 반면 모든 상태를 포괄적으로 구분한 경우에는 최적화된 테스트 케이스를 생성하기 어렵다. 그리고 임베디드 시스템의 특징인 실시간성과 다중이벤트입력 특성을 모델로써 표현하기 어려운 경우도 생기게 된다.

본 연구에서는 위와 같은 단점을 해결하기 위해서 SW에서 발생 가능한 고장메커니즘에 기반 하여 테스트를 수행하는 방법을 제시하고자 한다. 모델 기반의 블랙박스 테스트는 시스템 전체를 대상으로 입력대비 출력의 정확성을 보고 시스템의 고장유무를 판단한다. 그렇기 때문에 고장이 어느 부위에서 어떤 결함으로 인해 발생되었는가를 파악하기가 어렵다. 그러므로 임베디드 SW 테스트함에 있어 고장의 원인파악이 쉬울 뿐 아니라 구조적으로 접근하여 모델이 커버하지 못하고 있는 잠재된 결함을 검출해 내기 위해서는 SW의 결함과 고장모드를 분석하여 결함이 어떻게 고장모드로 발전해 나가는 과정 즉, 고장 메커니즘을 고려한 테스트를 수행 하여야 한다. 이번 연구에서는 이러한 고장 메커니즘 기반의 테스트 방법에 구조적으로 접근하기 위한 테스트 방법론을 제시하고자 한다.

2. 임베디드 시스템의 SW 로직 분류

테스트는 고장이 발생하였다는 것만을 보여주는 것이 목적이 아니라 그 원인이 무엇이며 무엇을 해결해야하는가를 알려주어야 한다. 시스템 전체를 대상으로 하는 모델 기반의 블랙박스 테스트 경우에는 "특정상태에서 특정 데이터를 주었더니 고장이 발생했다"는 것만 알려 줄 뿐 실제 고장의 원인이 무엇인가 또는 어느 경로를 수행하다 고장이 발생되었는가를 파악하기는 어렵다.

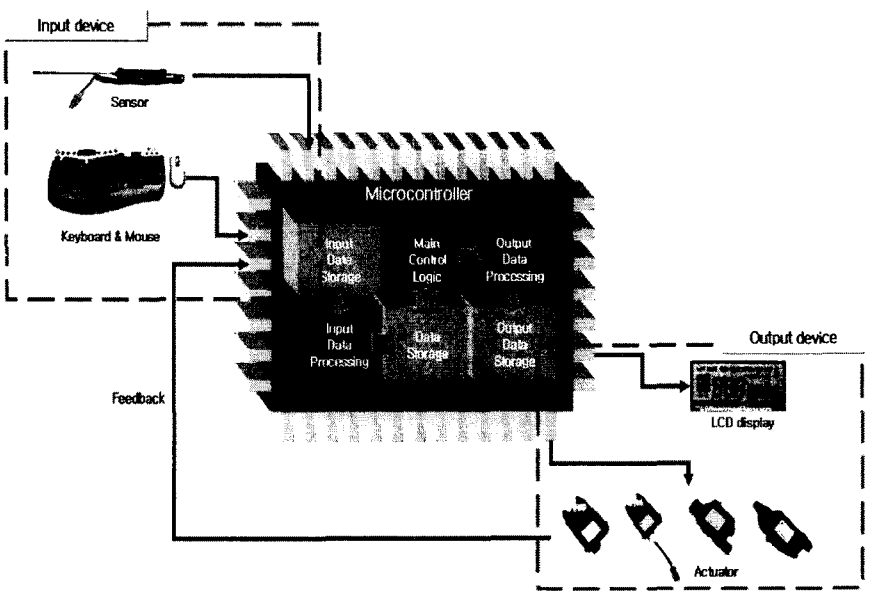
명세서를 통해서 구분이 가능한 컴포넌트들은 기본적으로 특정 출력장치를 구동시키기 위한 것도 있으며 입력장치로부터 데이터를 입력받는 컴포넌트들도 있다. 또한 내부적인 의사결정을 수행하기 위해 여러 가지 제약조건들을 해결하기 위한 컴포넌트들이 존재한다. 이러한 컴포넌트들을 본 연구에서는 로직이라고 말한다

시스템의 결함은 각각의 로직이 지닌 결함과 로직과 로직의 흐름간에 내재된 결함이 있

으며 또한 각각의 결함은 복합적으로 발생하여 또 다른 결함을 만들어낼 수 있다. 따라서 고장의 원인파악을 위해서는 시스템 전체를 대상으로 테스트를 수행하기 이전 세부적인 컴포넌트로 나누어서 먼저 각 컴포넌트들에 대해서 테스트를 수행하고 컴포넌트의 무결성이 확인된 이후 점진적으로 시스템 수준까지 테스트를 수행해야 한다.

이와 같이 범위를 특정 컴포넌트로 한정하고 각 단계별로 테스트를 수행하게 되면 "오류가 어디에서 발생 되었는가"와 "그 오류의 원인은 무엇인가"대해서도 쉽게 파악할 수 있다는 장점이 있다. 그러므로 SW에서 결함이 발생될 수 있는 범위를 각각의 로직으로 한정시키고 각 단계별로 발생 가능한 결함을 고려한다면 전체적인 결함의 범위에 대한 추정이 매우 용이하여 발생가능한 오류에 대한 테스트를 수행하는 것이기 때문에 테스트 수가 줄어들 수 있다는 장점이 있다.

임베디드 시스템은 실시간으로 출력장치들을 제어하기 때문에 이러한 작동메커니즘을 고려하여 임베디드 SW를 분할해 보면 그림 2와 같다. 이를 바탕으로 임베디드 SW 명세서를 통해서 구분이 가능한 출력장치 구동, 입력데이터 처리, 내부 의사결정을 위한 로직으로 구분 할 수 있다.



<그림 2> 임베디드 시스템의 로직

입력데이터 처리 로직은 사용자입력, 환경입력, 피드백입력 등의 물리적인 신호를 디지털 값으로 변화시켜 주는 역할을 수행하면서 그 값들은 지속적으로 특정 저장매체에 축적되거나 인터럽트 방식에 의해 값을 갱신시킨다. 주 제어로직은 입력된 데이터를 바탕으로 명세서에서 정의하고 있는 행위들을 S/W적으로 구현해 놓은 것이며 여러 가지 로직들의 유기적인 결합으로 구성되어 있다. 출력데이터 처리로직은 주 제어로직으로부터 연산된 결과를 기준으로 출력장치를 구동시키기 위한 물리적인 신호를 내보낼 수 있도록 마이크로 컨트롤러의 출력포트를 제어하는 로직이다. 이렇듯 임베디드 시스템은 여러 개의 단위 로직들의

조합으로 작동되고 있으며 이러한 단위로직들은 개별적으로 작동되기도 하고 시스템에서 정의된 바대로 여러 개의 로직들이 동시에 작동되기도 한다. 본 연구에서는 개개의 단위로직을 u-Logic이라 하고 일부 여러 개의 단위로직들을 동시에 제어하는 로직을 c-Logic이라한다.

그러나 일반적인 임베디드 시스템에서는 너무 다양한 u-Logic의 형태들이 존재하고 동시에 수행되는 것들도 있기 때문에 이러한 u-Logic만으로 시스템을 표현하기에는 매우 어렵다. 따라서 시스템의 상태를 좀 더 상위수준에서 추상적으로 표현하기 위해, 사용자에 의해 결정될 수 있는 시스템 상태를 c-Logic이라고 분류하였다. 개념적인 c-Logic 내부에서 실질적인 u-Logic들이 작동하기 때문에 c-Logic과 u-Logic은 계층적 구조를 지닌다.

예를 들어 간단한 임베디드 시스템인 커피 자판기는 버튼에 따라 밀크커피, 블랙커피 등이 나오게 된다. 대기상태에서 사용자에 의해 결정되는 밀크커피 혹은 블랙커피의 동작은 각각 밀크커피 c-Logic, 블랙커피 c-Logic 이며, 버튼을 누르게 되면 시스템 내부적으로는 컵을 밑으로 내리고 c-Logic 동작에 따라 일정양의 커피, 설탕, 물을 부어 넣는 순차적인 수행을 컵을 밑으로 내리는 u-Logic, put_water u-logic, put_coffee u-logic, put_sugar u-logic이라 할 수 있다. 설명과 같이 상위계층의 c-Logic을 통해 상태를 표현할 수 있으며 하위계층은 입력데이터를 조건으로 갖는 u-Logic을 수행 순서에 따라 계층적으로 분류할 수 있다.

3. 임베디드 시스템의 SW 고장메커니즘

시스템을 모델링 한 경우 시스템이 어떠한 상태인지는 관계없이 조건만 맞으면 전이를 일으키거나 어느 단계로 이동한다는 것을 묘사하고 있을 뿐 시스템의 상태를 동적으로 파악하지는 못한다. 만일 현재 시스템의 자원이 모두 고갈된 상태인데 그 점을 반영치 않고 새로운 상태로 전이하기 위해서 자원을 할당한 다면 결국 시스템은 고장으로 갈 것이다. 또한 상태는 변화 없는 가운데 자원누수가 누적이 되고 있는 상황이라면 상당한 시간이 지난 후에야 고장으로 발전한다.

이와 같은 고장은 모델 기반의 테스트로만 테스트하기 어렵거나 불가능한 경우들이다. 따라서 이러한 고장을 발생시키는 원인을 찾아내기 위해서는 어느 루틴에서 무엇이 잘못되면 어떠한 과정을 거쳐 고장이 유발되는가의 메커니즘을 찾아서 이를 테스트하는 것이 필요하다.

HW의 경우 고장과 관련된 속성을 고장부위, 고장원인, 고장메커니즘(고장발생과정), 고장형태로 구분하고, FMEA등과 같은 기법들을 활용하여 발생 가능한 고장속성들을 분석한 이후 고장메커니즘을 따를 수 있는 부하를 가해 테스트를 수행한다. HW는 재료의 물리/화학 및 형이상학적 특성에 따른 고장메커니즘들이 많은 연구를 통해 규명되었고, 가해지는 부하도 힘, 온도, 습도 등으로 매우 명확하여, 테스트의 수가 한정적이면서도 오류검출의 가능성이 매우 높다.

<표 1> HW와 임베디드 SW의 고장 속성 비교

구분	부하	원인(결함)	HW: 고장메커니즘 SW: 고장발생과정	고장형태
HW	힘, 온도 습도 등	Void, 재료의 성질과 같은 물리/ 화학적 특성 및 역학적 특성	크랙, 이온화 등	파단등과 같은 기능수행 불가
SW	Digital/Analog Signal	개발자 결함 디자인 결함 HW 결함	결함을 실행시키는 데이터의 흐름 (로직 실행경로)	명세와 다른 HW의 행위

표 1에서처럼 분석을 하게 되면 어떠한 고장이 일어났으며 그 고장이 일어난 원인은 무엇이며 어떠한 과정을 거쳐서 고장에 이르렀는가를 파악이 용이하다. 특정 원인에 의해서 고장이 일어나는 과정을 알고 있다면 원인이 있을만한 곳에 부하를 가하여 고장이 일어나가는 과정, 즉 메커니즘을 관측함으로써 고장이 발생했는지 여부를 검증할 수 있다.

고장메커니즘을 파악하고 있다면 이를 관찰할 수 있는 테스트를 수행하여 실제 고장이 일어나는가를 검증할 수 있다. 이와 같이 고장메커니즘을 고려한 테스트를 수행하게 되면 그 테스트 수가 매우 한정적이고 고장이 발생하였다면 그 원인에 대한 파악이 매우 수월하다. 임베디드 SW도 이와 마찬가지로 고장 물리적인 접근법을 통하여 고장메커니즘을 알 수 있다면 고장에 구조적으로 접근할 수 있을 뿐만 아니라 테스트의 수를 줄이며 오류의 검출도를 높일 수 있다.

따라서 오류검출 가능성을 높이기 위해서는 단순 최적화 로직 실행경로 선택이 아닌 고장원인에 의해 고장모드가 나타날 만한 로직 실행경로를 선택이 필요하다. 본 연구에서는 이러한 로직 실행경로를 임베디드 SW의 고장메커니즘이라 정의한다.

정의 1. 임베디드 SW의 고장메커니즘

고장원인에 의해 고장모드가 나타날 가능성이 있는 임베디드 SW의 로직 실행경로

임베디드 SW의 고장메커니즘을 찾아내기 위해서, 본 연구에서는 필드 고장데이터를 임베디드 시스템이 가질 수 있는 공통적인 특징을 고려하여 SW 고장을 유발할 수 있는 메커니즘적인 원인에 대해 수집/분석하여 표 2와 같이 일반화된 고장메커니즘을 정리하였다.

표 2에서 임베디드 S/W의 고장원인은 기존의 일반적인 결함분류법에 따라 분류할 수도 있지만 본 연구에서는 Howord[19]와 Richardson[20]의 연구와 유사한 사상으로 임베디드 시스템이 가질 수 있는 특징을 고려하여 본 절에서는 S/W 고장을 유발할 수 있는 메커니즘적인 원인에 대해 분류하고 2절에서 언급한 로직단위 단계적 테스트 등에 적용시켜 각 로직 및 로직 간에 대응시켜 변형될 수 있는 메커니즘적인 고장 원인까지 누락되지 않도록 하였다.

<표 2> 내장형 SW의 일반화된 고장메커니즘

고장 메커니즘	비고
기본 기능의 부정확	FM01
특정 로직을 거쳐 로직 실행시 로직 중 일부 또는 전체 작동 불가	FM02
특정값 입력 시 오작동	FM03
작동 불가 상태에서 작동됨	FM04
전원 인가시 HW의 순간적인 비 정상적 반응	FM05
전원 인가시 급격한 HW의 반응	FM06
전원 차단시 HW의 반응	FM07
특정로직 진입시 HW의 순간적인 움직임	FM08
HW 특정 위치시 부정확한 움직임 (떨림)	FM09
이상 발생 -> 정상복귀 시 이상발생 상황 유지	FM10
순간적인 노이즈 신호에 다른 로직의 작동	FM11

4. 고장메카니즘 기반 테스트 전략

임베디드 SW의 테스트가 추구해야할 목적은 주어진 시간 동안 가능한 많은 오류를 검출하는 것이며 또한 오류가 검출되었다면 그 오류의 원인은 무엇인지를 밝혀내는 것이다. 즉 테스트 수 줄여 시간을 줄이면서 가능한 모든 오류를 검출해내는 것이 목적이다. 이러한 목표를 달성하기 위해서 단계적 테스트 절차를 통해 테스트를 수행하도록 한다.

앞서 2절에서 설명한 바와 같이 스펙을 바탕으로 u-Logic과 c-Logic들을 구분하고 로직간의 검사를 통해 개별적인 단위를 추출하여 검사하고 단계적으로 시스템의 수준으로 확장해 가면서 검사를 수행함으로써 고장의 원인에 구조적으로 접근할 수 있다. 이와 같은 고장은 모두 모델 기반의 테스트로는 테스트하기 어렵거나 불가능한 경우들이다. 모델기반의 블랙박스 테스트의 경우 어떠한 상태인지는 관계없이 조건만 맞으면 전이를 일으키거나 어느 단계로 이동한다는 것을 묘사하고 있을 뿐 시스템의 상태를 동적으로 파악하는 지는 못한다. 만일 현재 시스템의 자원이 모두 고갈된 상태인지의 그 점을 반영치 않고 새로운 상태로 전이하기 위해서 자원을 할당한 다면 결국 시스템은 고장으로 가게 된다. 또한 상태는 변화 없는 가운데 자원자원누수가 누적이 되고 있는 상황이라면 상당한 시간이 지난 후에야 고장으로 발전한다.

그러므로 이러한 고장을 발생시키는 원인을 찾아내기 위해서는 입력대비 발생 가능한 출력을 보는 블랙박스 테스트의 경우 어느 루틴에서 무엇이 잘못되면 어떠한 과정을 거쳐 고장이 유발되는가의 메커니즘을 찾기란 힘들다. 즉 블랙박스 테스트만으로 이를 테스트하여 잠재적인 오류들을 제거할 수 없다는 것이다.

위와 같은 블랙박스 템스트의 단점을 보완하기 위하여 먼저 단계적 테스트를 하는 단계인 검사와 전체 시스템 수준의 테스트를 수행하는 단계인 시험 두 가지로 구분하고 각 단계적으로 시행하여야 한다.

1) 검사
0단계 관찰 변수의 선정
1단계 데이터 검사
2단계 u-Logic 검사
3단계 연관된 u-Logic간의 검사
4단계 c-Logic 시험

2) 시험
5단계 시나리오 기반의 시험

각 단계에서 로직에 입력해야할 신호와 관찰해야할 출력이 무엇인가를 명확히 하여 개별적으로 테스트를 수행했기 때문에 어느 로직에서 어떠한 고장이 발생했는가를 쉽게 알아낼 수 있으며, 또한 개별적으로 테스트를 수행했기 때문에 어느 로직에서 어떠한 고장이 발생했는가를 쉽게 알아낼 수 있으며, 또한 개별 로직의 결함이 다른 로직으로 전파되는 것을 막아 결함과 결함이 결합되어 발생하는 또 다른 결함에 대하여 시나리오 기반으로 전체 시스템 수준의 테스트를 시행함으로써 불필요한 테스트의 수를 줄일 수 있다.

단계적 테스트와 더불어 검출도를 높이기 위해서 고장 물리기반의 테스트 데이터를 생성하도록 한다. 고장 물리기반 테스트는 고장이 발생해 가는 과정을 파악하고 이를 테스트함으로써 오류의 검출도를 높일 수 있다. 이러한 장점을 이용하기 위해서는 먼저 고장 메커니즘에 대한 정리가 필요하다. 임베디드 SW 고장 메커니즘을 정리하기 위해서 먼저 정보의 흐름을 통해 임베디드 SW가 대응해야 할 외부적인 원인과 임베디드 SW에서 제거해야할 내부적인 원인으로 구분하여 분류하고 이를 바탕으로 결함이 고장으로 발전해 가는 과정을 정리하여 메커니즘적인 고장 원인을 단계적 테스트에 적용해서 u-Logic, u-Logic 과 u-Logic, c-logic과 c-Logic에서의 메커니즘적인 고장원인을 재 불리 하고 메커니즘적인 원인별 테스트 데이터를 생성하도록 한다.

각 단계에서 로직에 입력해야할 신호와 관찰해야할 출력이 무엇인가를 명확히 하여 개별적으로 테스트를 수행했기 때문에 어느 로직에서 어떠한 고장이 발생했는가를 쉽게 알아낼 수 있다. 개별 로직에서의 결함과 로직 사이의 결함의 전파를 사전에 방지함으로써 결함과 결함이 결합되어 발생하는 또 다른 결함에 대한 불필요한 테스트의 수를 줄 일 수 있다.

앞서 분류한 메커니즘적 고장원인은 직접적으로 고장모드로 발생될 수도 있으며 그 결함(원인)이 또 다른 결함(원인)으로 전파되어 최종적으로 고장모드로 발생될 수 있다. 앞서 u-Logic별로 발생할 수 있는 메커니즘적인 고장원인은 각각의 u-Logic의 고장모드에 대한 원인으로 파악될 수 있다.

따라서 단계적인 테스트에 고장 물리를 고려한 테스트를 적용하기 위해서는

1) S/W 전체의 고장모드 파악
2) S/W 전체의 고장모드로 발전할 수 있는 u-Logic의 상황 설정 (u-Logic의 고장모드)
3) u-Logic의 고장모드를 발생시킬 수 있는 메커니즘적인 원인의 유추
4) 메커니즘적인 원인을 따를 수 있는 테스트 데이터의 생성

의 절차를 거쳐 수행 되어야 한다.

5. 사례연구

지금까지 제안된 단계적 테스트 전략과 분류된 고장메커니즘을 통해 테스트 데이터 생성 방법의 성능을 보고자 온도컨트롤이 탑재된 임베디드 SW를 대상으로 테스트를 수행하였다. 에어컨디셔너 혹은 히터라고 불리는 공조시스템은 실내로 유입되는 공기의 온도, 풍향, 풍속을 조절하여 쾌적한 차내 환경을 만들어 주는 기능을 수행한다. 이러한 공조시스템을 작동시키는 컨트롤러는 온도컨트롤러(TC: temperature control)라 부린다. TC는 센서와 사용자의 입력을 통해 고조시스템을 작동시키며, 현재의 작동상태를 알려준다. 작동방식에 따라 자동 및 수동 TC로 분류된다.

<표 3> 구체화된 고장메커니즘 (A/C 제어기능 예)

일반고장 메커니즘	A/C를 제어하는 u-Logic에 대한 고장메커니즘	발생가능 고장모드
FM01	자동, 습도, EVA센서의 제어선도	ON/OFF 작동안됨
	수동선택	ON/OFF 작동안됨
	제습, 최저온도 설정	A/C ON 작동안됨
FM02	자동, 수동, 제습, EVA, 최저온도 설정 기능들의 전이	이전상태 미 복귀
	MODE 혹은 DEF키를 사용하여 제습기능 해제	이전상태 미 복귀
FM03	센서 범위를 벗어나는 값 입력	작동안됨
FM04	시스템 OFF 상태	A/C 작동
FM05	자동 및 수동 A/C제어동작 중 시동 또는 시스템 OFF->ON	IND 깜박임
	제습 및 EVA기능 동작 중 시동 또는 시스템 OFF->ON	IND 깜박임
	냉방 기동제어 동작 중 시동 또는 시스템 OFF->ON	IND 깜박임
FM07	시동 OFF 진입	IND 계속 ON
FM08	자동 및 수동제어 중 다른 로직 진입	IND 깜박임
	제습 및 EVA제어 중 다른 로직 진입	IND 깜박임
FM09	자동제어 선도 ON/OFF 기준점 도달	IND, DSP 깜박임
	습도 센서 제어선도 ON/OFF 기준점 도달	IND,DSP 깜박임
	EVA센서 제어선도에 ON/OFF 기준점 도달	IND,DSP 깜박임
FM10	EVA센서 제어선도에 따라 않음(FM01과 중복)	ON/OFF 작동안됨
FM11	EVA센서 노이즈 입력	EVA로직 작동/해제
	내외기 온도 및 광센서 노이즈 입력	A/C ON/OFF 작동

자동 방식 TC의 시스템 테스트를 위해 시스템 명세서를 분석하여, 각 출력장치를 제어

및 보호를 위한 27가지의 u-Logic과 사용자에 의해서 지정될 수 있는 자동제어, 수동제어, 제습, 시스템 작동중지의 4가지 c-Logic으로 분류하였다.

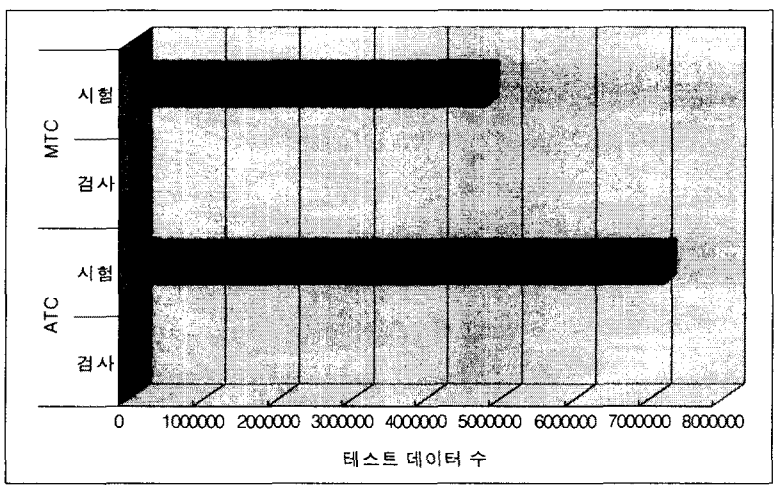
먼저 관찰해야 하는 변수들을 선정한 후 시스템 내부적으로 참조하는 센서값들에 대한 검사를 수행한 후 2-4단계의 검사 및 시험에서 각 단계별로 일반 고장메커니즘을 구체화시켜 테스트 데이터를 생성하였다. 표 3은 에어건(A/C)을 제어하는 u-Logic들에 대한 구체화된 고장메커니즘을 정리한 것이다.

본 연구에서 제시한 방법의 효용성을 확인하기 위해서, 작동 시나리오를 고려한 테스트(SCT)와 고장메커니즘 기반 테스트(FMT)를 수행하여 비교하였다.

<표 4> TC 테스트 결과

구분	테스트 데이터	중요도기준		
		MAJOR	MINOR	ETC
ATC	76,800 여건 (7,200,000 여건)	7	1	1
MTC	22,000 여건 (4,800,000 여건)			1

표 4에서 MAJOR 항목의 오류는 사양서에 명시된 내용과 TC가 전혀 다르게 작동하는 오류이며, MINOR는 순간적으로 오작동하나 곧 정상적인 제어상태로 복귀하는 오류이다. ETC는 명세서를 개발자가 잘못 이해하여 발생한 오류이다.



<그림 3>테스트 성능 비교

표 4와 그림 3에서 보는 바와 동일한 오류를 검출하면서도 FMT가 SCT수의 약 1.09%(ATC)및 0.46%(MTC)밖에 되지 않는다.

6. 결론

본 연구에서는 임베디드 SW가 일반 SW와는 다른 개발환경을 갖기 때문에, HW/SW 통합이전의 컴포넌트(유닛) 테스트와 통합 테스트의 개념을 포함시킨 임베디드 SW의 시스템 테스트 절차를 제시하였다. 또한 내장형 시스템의 SW로 인한 고장데이터를 분석하여 고장에 관점을 두고 최적화시킨 고장 메커니즘기반 테스트 방법을 제시하였다. 향후 다양한 임베디드 S/W의 메커니즘적인 고장원인을 여러 가지 카테고리로 분류하여 데이터베이스화한 후 유사한 로직에 대해서는 검증 패턴으로 활용이 될 수 있는 방안을 연구하고 아울러 고장 물리기반의 테스트 데이터를 자동으로 생성하는 방안에 대해서도 연구하고자 한다.

참고문헌

[1] Avizienis, J. C. Laprie, B. Randell(2001) Fundamental Concepts of Dependability, Technical Report 1145 , LAAS.

[2] Beizer(1990) Software Testing Techniques , International Thomson Publishing Inc 2nd edition.

[3] Bernardeschi, A. Fantechi, S. Gnesi(2002) Model checking fault tolerant systems, Software Testing, Verification and Reliability , Vol. 12, Iss. 4, pp. 251-275.

[4] M. Blackburn, R. Busser, J. Fontaine(1997), "Automatic Generation of Test Vectors for SCR Style specifications", Proceedings of IEEE COMPASS '97, pp. 54-67.

[5]. K. T. Cheng and A. S. Krishnakumar(1996), "Automatic generation of functional vectors using the extended finite state machine model," ACM Transactions on Design Automation of Electronic System, Vol. 1, Iss. 1, pp. 57-79.

[6] J. Esch(1996) "Prolog to Principles and Methods of Testing Finite State Machines-A Survey," Proceedings of the IEEE, Vol. 84, Iss. 8, pp. 1089.

[7] S. Faulk, J. Brackett, P. Ward, J. Kirby(1992) "The CoRE method for real-time requirements," IEEE Software, Vol. 9, Iss. 5, pp. 22-33.

[8] V. Giri(2003) "Bug Taxonomics : Use Them to Generate Better Tests", STAREAST.

[9] Harel(1987) "Statecharts: a visual approach to complex systems," Science of Computer Programming, Vol. 8, pp. 231-274.

[10] S. Herrmann(2003) Software Safety and Reliability, Published by the IEEE Computer Society.

[11] Hermmann J.(2001), guideline for Validation & Verification Real-time Embeded software systems, ITEA, Vol.1.

[12] IEEE Standard 1044.1-1995(1996), IEEE guide to classification for software anomalies, IEEE.

[13] Keidar, R. Khazan, N. A. Lynch, A. A. Shvartsman(2002) "On fault classes and error detection capability of specification based testing," ACM Transactions on Software Engineering and Methodology, Vol. 11, No.1, pp. 58-62.

[14] B. Legeard, F. Peureux, M. Utting(2004) "Controlling test case explosion in test generation from B formal models," Software Testing, Verification and Reliability, Vol. 14, Iss. 2, pp. 81-103.

[15] T. Nakajima, Y. Bessho, H. Yamanaka, K. Hirota(2003) "Automatic testing of embedded software based on state-transition requirement specifications," Electronics and Communications in Japan (Part II: Electronics) Vol. 86, Iss. 9, pp. 64-75.

[16] S. C. Ntafos(1988) "A comparison of Some Structural Testing Strategies," IEEE Transactions on Software Engineering, Vol. 14, No. 6, pp. 868-874.

[17] J. Offutt, A. Abdurazik(1999) "Generating tests from UML specifications," Proceedings In 2ed International Conference on the Unified Modeling Language, Vol. 1723, pp. 416-429.

[18] J. Offutt, S. Liu(1999) "Generating test data from SOFL specifications," The Journal of Systems and Software, Vol. 49, No. 1, pp. 49-62.

[19] D. Howard(1997). "An Analysis of Security Incidents on the Internet," Ph.D. dissertation, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213 USA, April.

[20] T. W. Richardson(2001) "The Development of a Database Taxonomy of Vulnerabilities to Support the Study of Denial of Service Attacks," PhD thesis, Iowa State University.