열차 제어 임베디드 시스템에서의 신뢰성 검증에 관한 연구

The study of verification for reliability in train control embedded system

홍 효 식 *

Hyo Sik Hong

1. Abstract

Since the embedded system is more intelligent, the importance of the reliability in the embedded system is a lot more visible. Especially the reliability of the embedded system in the train control system is even better important. As the special quality of the embedded system, the hardware of the system is directly controlled by the software in the embedded system. As the expansion of complexity , the expense amd the time for verification is required more and more. This paper is presented the verification of the reliability as the method with background of failure

Embedded system is gradually, the importance of 'A built-in on the system embedded system's reliability is focused . In particular, in the train control system built-in of the embedded system, reliability is even more important. The embedded system of the system controls over hardware, software with built-in directly. The more complex system is , the more increasable of depending on the reliability of the time verification and expensive is . This paper is about he characteristics of the reliability and verification of embedded system under the failure mechanisms, based on the verification methodology suggested by in the train control system,

1. 서 론

임베디드 시스템의 응용 소프트웨어 개발은 임베디드 시스템이 갖는 특징인 자원의 한정성과 실시간 제어로 인해 일반적인 시스템의 응용 소프트웨어의 개발과는 달리 많은 제약 사항이 발생하며, 이러한 소프트웨어 개발의 제약에 의해 임베디드 시스템의 응용 소프트웨어의 개발은 오류의 가능성이 매우 크다고 할 수 있으며 이러한 오류를 제거하기 위하여 지금까지 많은 연구가 있어왔다. 아울러 임베디드 시스템의 소자들이 점차 지능화 및 다 기능화가 되고 제어의 복잡도가 증가함에 따라 임베디드 시스템이 갖는 무결성이 매우 중요하게 대두되고 있다. 즉, 임베디드 시스템의 응용 소프트웨어의 오류는 치명적인 결과를 초래할 수도 있다. 아울러 소프트웨어의 무결성을 검증하는게 매우 중요하며 제어의 복잡도가 증가함에 따라 소프트웨어의 복잡도가 증가되므로 이 응용 소프트웨어에 대한 검증에 필요한 시간과 비용의 급격한 증가를 초래한다.

특히 열차제어 시스템에 사용되는 임베디드 시스템의 경우에는 소프트웨어의 오류에 의해 열차의 탈선 또는 추돌 등과 같은 매우 큰 치명적인 결과로 이어지기 때문에 더더욱 소프트웨어의 무결성과 안정성의 확보를 위한 검증작업이 필요하다. 본 논문에서는 이러한 필요성에 의해 제기되는 검증작업에서의 검증절차와 검증방법에 관해 고찰 할 것이다.

*. 한국철도대학 교수(철도학회 정회원)

특히 열차제어 시스템에 사용되는 임베디드 시스템의 경우에는 소프트웨어의 오류에 의해 열차의 탈선 또는 추돌 등과 같은 매우 큰 치명적인 결과로 이어지기 때문에 더더욱 소프트웨어의 무결성과 안정성의 확보를 위한 검증작업이 필요하다. 본 논문에서는 이러한 필요성에 의해 제기되는 검증작업에서의 검증절차와 검증방법에 관해 고찰 할 것이다.

임베디드 시스템의 소프트웨어의 무결성 검증작업은 소프트웨어 모듈 및 모듈간 연결시험에 의해 검증되어져야 하며, 예측 가능하지 못한 경우는 에러가 발생하는 경우라면 필히 안전성을 보장해야 한다. 즉, 에러가 발생한 경우에는 안전측(Fail safe)으로 동작 되어져야만 한다.

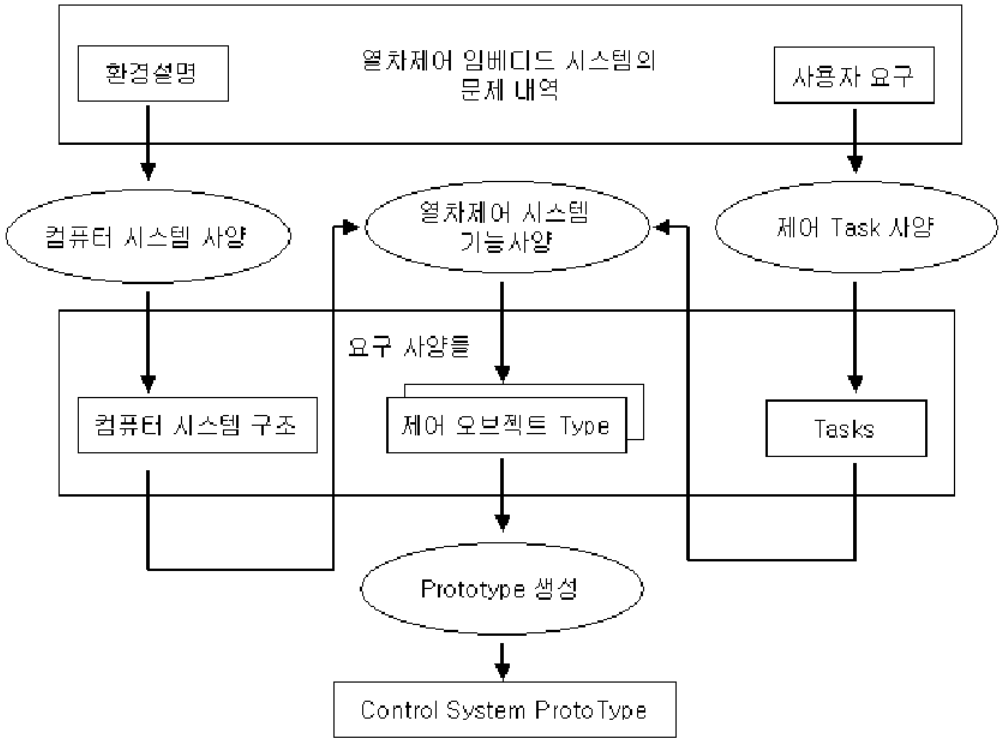
기존의 임베디드 소프트웨어의 무결성 검증작업은 일반적인 소프트웨어를 개발에 의한 방법을 이용하거나, 소프트웨어를 개발하는데 필요한 그래프 이론을 사용하여 모델을 생성한 후 이 모델들의 시험경우들을 생성하고 이를 검증하거나, 소프트웨어에서 발생할 수 있는 고장 메카니즘을 추정할 수 있는 시험경우들을 생성하고 이를 검증하는 방법들을 사용하고 있으나 본 연구에서는 이러한 방법을 종합적으로 보다 쉽고 확실하게 Fail-Safe 와 안정성 및 신뢰성을 검증할 수 있는 방법을 제시하고자 한다.

이는 열차제어 임베디드 시스템을 여러 가지 입력장치로 부터 정보입력을 받는 블랙박스로 간주하고 이 블랙박스의 소프트웨어의 기능을 시험하고 논리적 결함이 있는 경우 안전측으로 동작되는 것을 확인하는 방법이다. 이 방법은 소프트웨어의 논리적 결합의 원인은 찾을 수 없으나 안전측 동작의 확인을 위한 시간과 비용의 발생을 최소화 시킬 수 있다. 본 연구에서는 열차제어 임베디드 시스템의 정의 되어진 기능에 맞추어진 시험경우에 관한 절차를 제안하고 소프트웨어를 개발할 때 발생될 수 있는 여러 가지 기능들이 서로 간에 상충되어 발생되어 질수 있는 요소들을 사전에 배제한다. 열차제어시스템의 개발의 복잡성 즉 제어의 복잡성이 증가함에 따라 상호간 상충되는 요소들을 수동적으로 일일이 찾아서 제거할 수 가 없다. 이는 정해진 절차에 의해 순차적으로 제거 되어져야 하므로 본 연구에서는 이러한 시험절차에 관해서도 제안하고자 한다. 또한 본 연구에서 제안하는 블랙박스 검증방법은 여러 가지 입력요소 즉 정형적인 입력자료인 키값과 여러 가지 센서에 의한 입력값을 받는 복잡한 임베디드 시스템을 블랙박스로 간주한다고 전제하였다. 이 블랙박스로 간주하는 방법은 소프트웨어의 소스코드를 기반으로 이루어지는 화이트박스 시험에 의해 발견되어지는 소프트웨어의 논리적 결함을 쉽게 찾을 수 있다는 장점에 비해 어렵다는 단점이 있지만 간단히 입출력만으로 소프트웨어의 기능시험이 가능하고 소스코드를 모르더라도 시험이 가능하며, 변경되어지는 소프트웨어의 기능시험도 가능하다는 장점이 있다. 본 연구의 2절에는 철도제어 시스템의 임베디드 시스템을 모델링하고 즉 블랙박스로 만들고 3절에서는 열차제어 시스템의 고장 메카니즘을 분석하고 4절에서는 고장 메카니즘 기반의 시험경우들을 제안한다.

2. 철도제어 임베디드 시스템의 소프트웨어 모델링

열차제어 시스템의 임베디드 소프트웨어를 개발하는데 있어서 개발과정을 정확하게 이해함으로써 소프트웨어의 검증을 보다 명확하게 정의 내릴 수 있다. 이러한 개발방법에 있어서 명확하게 표현 가능하게 하는 것이 개발을 위해 요구사량의 요소들을 나타내는 Product Model 이다. 이러한 Product Model 을 이용하여 소프트웨어의 모델을 정의내리고 상호 연관성을 이용하여 표현을 하고자 한다.

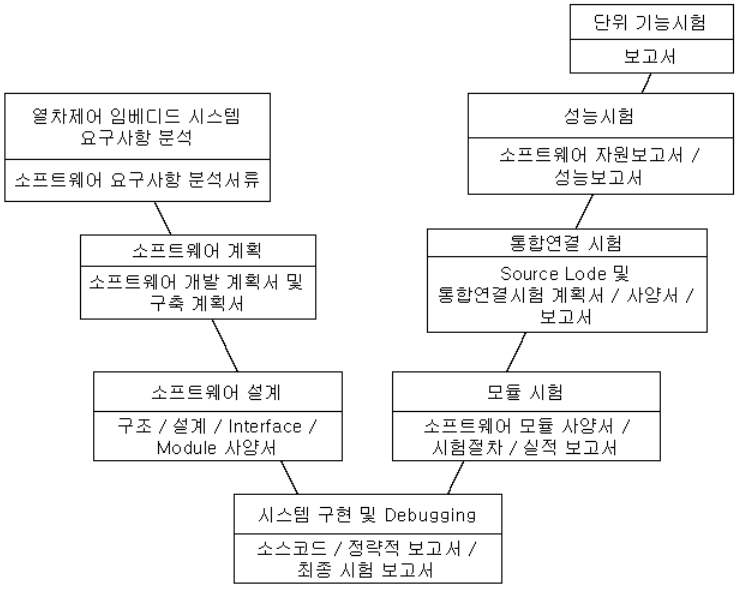
개발되어지는 열차제어 임베디드 시스템의 소프트웨어가 갖는 특성과 기능을 나타내는 Product Model 은 그림 2.1과 같이 나타낼 수 있다.



< 그림 2.1 열차제어 System에서 Product Model >

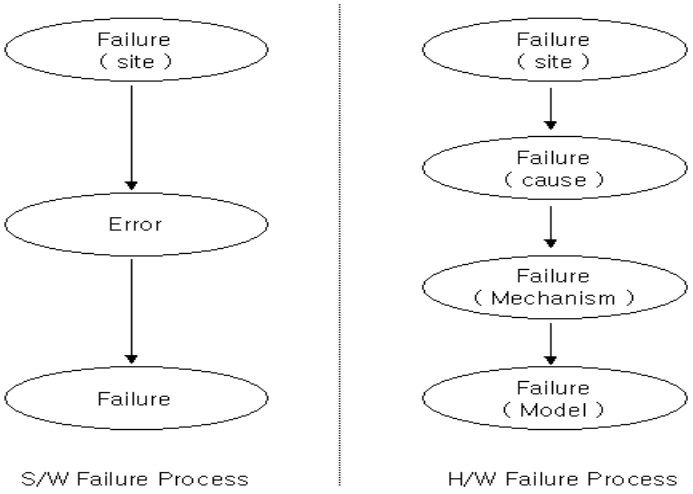
임베디드 시스템을 크게 세 가지 요소로 구분하여 나타낸다면 입력요소, 처리요소 및 출력요소이다. 이러한 요소를 Product Model 화하여 각각 구분하여 시스템을 개발한다고 정의를 내리고 이중 처리요소를 블랙박스 화하여 검증절차를 제안한다. 그림 2.1에서 보여지는 것처럼 Product Model 방법에 의한 시스템 개발은 시스템 환경을 정의하고 요구사항들을 제시 하는데서부터 시작되고 이러한 요소들의 집합을 문제내역 이라고 한다. 환경은 정의하는데 있어서는 환경의 구조를 포함하여 개발대상으로 하는 시스템 환경을 정의한다.

열차제어 임베디드 시스템에 있어서는 개발모델은 그림 2.2와 같다.



<그림 2.2 소프트웨어의 개발계획>

그림 2.2에서 나타난 바와 같이 각 단계에서는 해당 서류가 존재한다.
임베디드 시스템에서 H/W와 S/W의 고장모드가 발생하는 프로세서를 비교하면 그림 2.3과 같다.



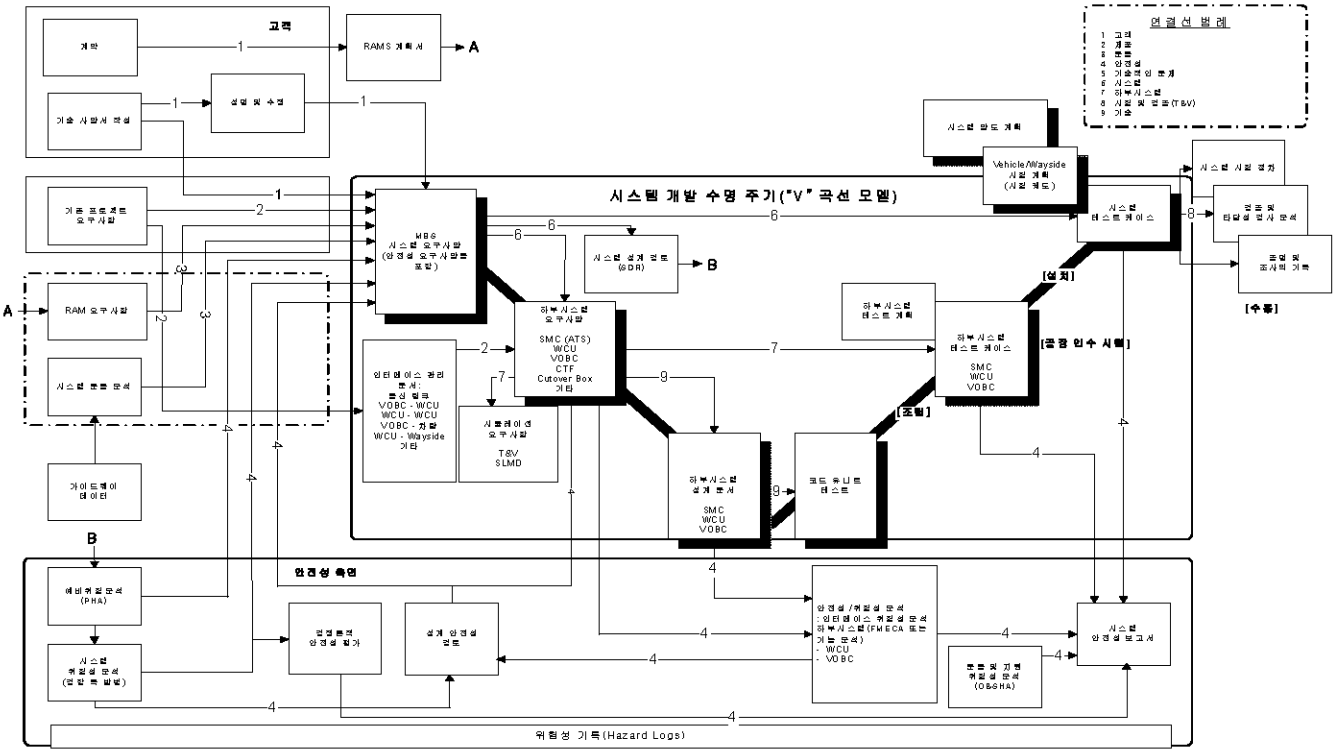
< 그림 2.3 S/W 및 H/W의 Failure Process >

소프트웨어에서의 결함은 하드웨어의 고장원인에 해당되고 소프트웨어의 오류(Error)는 하드웨어 고장 메카니즘과 대응할 수 있다.

소프트웨어의 오류라는 것도 소프트웨어에서 발생할 수 있는 일반적인 것들은 이미 잘 알려져 있다. 그러나 오류에 해당되는 부분도 단순히 문법, 연산자, 데이터 처리, 조건 등의 오류로만 구분되어 있을 뿐 실제 이것으로 인해 어떠한 고장들이 발생할 수 있는지의 여부는 언급되어 있지 않으나 이는 시스템이 모두 틀리기 때문에 오류로 인하여 발생할 수 있는 고장들을 일반적으로 표현하기가 어렵기 때문이다.

소프트웨어의 개발과정을 도식화한 그림 2.3에서 우측 상단에 있는 단위 기능시험이 맨처음 시작된다. 이는 시험을 시행할 때는 작은 단위부터 시작하여 차츰 큰 단위로 진행되어야 함을 말한다. 시험에서 발생되는 오류 중 약 80%가 전체원인의 20%내에서 야기된다는 파레토의 원리가 존재한다. 또한 작은 단위에서의 시험에서 발견되는 오류의 원인을 찾는게 큰 단위에서의 경우보다도 훨씬 쉽기 때문이다. 이와 같은 생각을 바탕으로 임베디드 소프트웨어를 작은 단위로 구분해서 각각에서 발생할 수 있는 고장 메카니즘을 찾아내는 것이 전체단위에서 찾는 것 보다 훨씬 수월하다. 즉 블랙박스로 간주된 전체기능을 단위기능으로 나누어서 이를 하나의 조직으로 단위화 하여 시험할 수 있게 한다.

임베디드 소프트웨어를 작은 단위로 구분하기 위해서는 소스코드 모듈을 알아야만 하지만 블랙박스 시험을 한 단위로 이루어진 실행파일을 바탕으로 수행되는 것이므로 소스코드 모듈을 구분한다는 것은 매우 어려워 불가능에 가깝다. 그러나 검증하고자 하는 것은 기능사양서에 정의된 기능이기 때문에 사양서에 정의된 기능사양을 단위로 구분하여 각 기능에 대한 논리에 대하여 고장 메카니즘들을 파악하고 기능별 단위 논리를 중심으로 정확한 고장 메카니즘을 추적할 수 있는 시험경우를 만들 수 있다면 소프트웨어의 오류를 찾아낼 수 있는 가능성이 높아지며 찾은 오류에 대한 원인도 쉽게 찾아낼 수 있다. 이러한 시험 항목에 대한 예로서 부록에 의해 나타냈다. 아울러 시스템 개발 수명주기에 따른 V곡선 모델을 그림 2.4 로제시하여 이를 바탕으로 고장 메카니즘에 따른 시험 경우들을 제안하였다.



< 그림 2.4 시스템 개발 수명주기 V모델에 따른 위험기록물 산출>

3. 열차제어 임베디드 시스템의 고장 메카니즘

3.1 고장 메카니즘

일반적으로 소프트웨어 고장에 관련된 속성들은 다음과 같이 정의된다.

1) 결함 (Fault) : 오류의 알고리즘적인 원인

2) 오류 (Error) : 시스템의 고장 발생 상황

3) 고장 (Failure) : 정의된 행위를 수행하지 않는 것

결함은 오류의 원인이 되지만 실제 오류로 전이되지 않는다면 고장이 일어나지 않는다. 이러한 결과에 따라 원인을 추적하여 임베디드 시스템의 고장을 방지하기위하여 고장 메카니즘의 분석이 필요하다.

임베디드 시스템의 하드웨어의 고장 메카니즘은 그동안 수많은 연구에 의해 규명되어지고 있으며 현재에도 계속되고 있다. 임베디드 시스템의 소프트웨어는 해당 시스템마다 기능이 다르고 필요로 하는 시스템 환경이 다르므로 임베디드 시스템의 소프트웨어의 고장 메카니즘에 관한 연구는 매우 드물게 이루어지고 있다. 임베디드 소프트웨어만이 갖고 있는 고유한 고장 메카니즘이 정의되고 고장 메카니즘을 추적할 수 있는 시험경우를 생성하고 이를 실행하게 된다면 모든 경우의 수를 실제 행해보지도 않고서도 훌륭한 시험성적의 결과를 얻을 수 있게 될 수 있다. 그러므로 본절에서는 열차제어 시스템의 단위조직을 구분하고 이 단위조직에 의한 고장 메카니즘을 분석하고자 한다.

3.2 단위기능 고장 메카니즘

철도제어 임베디드 시스템에서 발생할 수 있는 입출력조직에서 발생할 수 있는 고장 메카니즘을 표 3.1과 같이 정리하였다.

고 장 부 위	고 장 원 인	고장 메카니즘
입력 Data 처리조직	AD Sampling	Sampling 오류
	AD 변환	연산오류 또는 참조오류
	Fail - Safe	입력기기 고장시 처리부재
	Interrupt	처리후 재수행 불가
		동기 오류
		Interrupt Level 오류
출력 Data 처리조직	출력 Port 지정	연산결과 오류
		동기 오류
	출력중단	시간 부정확
		동기 오류
		Feedback 제어 오류
	Fail - Safe	출력기기 고장시 처리부재
		Fail - Safe 측 설정 오류

< 표 3.1 입출력조직의 고장 메카니즘 >

표 3.2는 열차제어 임베디드 시스템의 소프트웨어의 단위조직에서 발생되는 고장 메카니즘을 정리한 것이다.

고장 부위	고 장 원 인	고장 메카니즘
제어조직	연산	잘못된 입력에 의한 오류
		열차속도 허용한도에 미설정
		0으로 나눔
		이상치의 처리부재
		열차시간 Table의 오류
	Data 처리	잘못된 참조
		기본값의 미지정
		Over Flow or Under Flow
		미규정 열차
	분기(조건) 제어	조건의 불만족
		하드웨어 구동점 오차 부재
		잘못된 서브루틴 호출시 대처 부재
		분기문간의 기간차이
	Loop(반복) 제어	탈출조건 부재
		사용변수의 중복사용
		만족조건 부재

< 표 3.2 제어조직 고장 메카니즘 >

3.3 개발주기에 따른 RAMS 관리

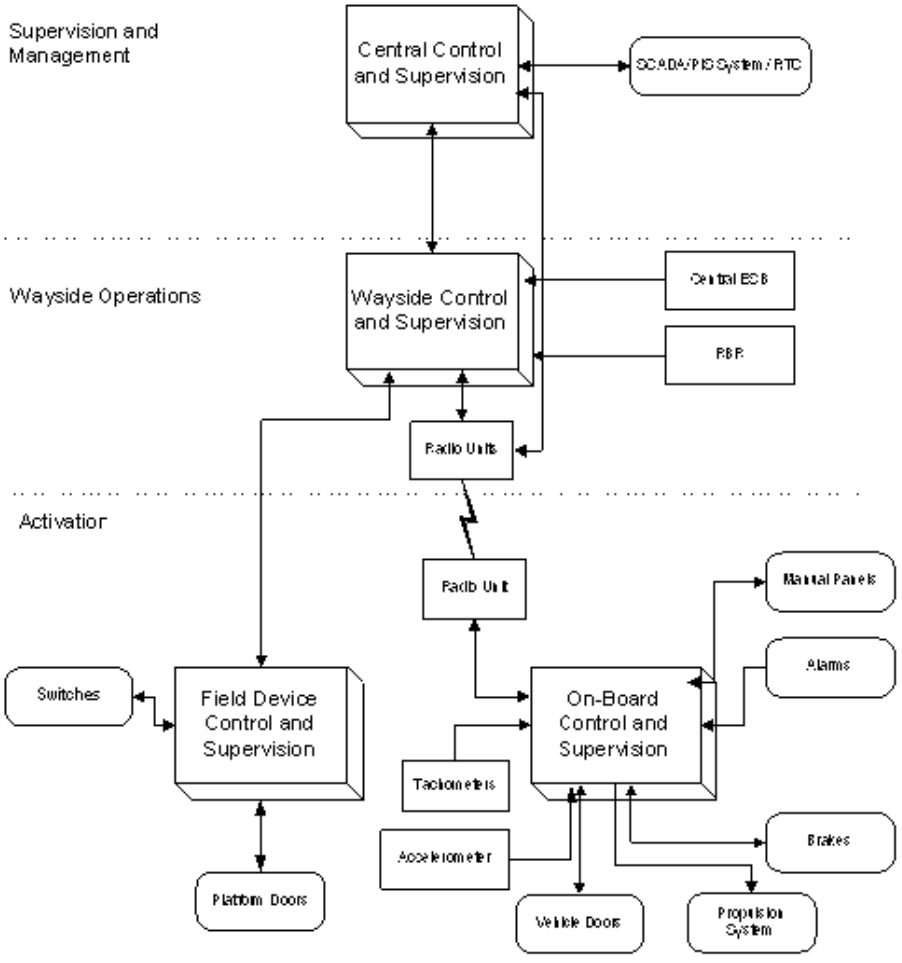
RAMS 관리는 시스템 수명주기에 걸쳐 이루어진다. 초기 고객의 요청에 따른 시스템 요구사항을 정립함으로써 RAMS 요구사항을 수립하게 된다. 이러한 시스템 및 RAMS 요구사항에 대한 요구사양이 수립되면, 이러한 요구사양을 바탕으로 시스템의 설계 검토가 기술적인 측면에 따라 수행된다. 이러한 설계 검토를 통하여 시스템에 대한 예비위험분석이 이루어진다. 이러한 분석결과를 통해 발생된 시스템 위험성과 발생요인을 참고로 시스템의 위험성 분석이 수행된다. 안전성 측면에서 수행되는 예비위험 분석과 시스템 위험성 분석결과를 통하여 시스템 요구사양을 재 고려하게 되며, 이러한 전체적인 시스템

RAMS 요구사양은 하부시스템 요구사양에 할당된다. 안전성 측면에서 고려된 시스템 위험성 분석은 안전성 평가를 통하여 실제 설계시 안전성이 적합한지의 유무를 검토하여 시스템 요구사양과 하부시스템 요구사양에 할당된다. 이후, 설계시 하부시스템에 대한 안전성 분석이 여러 가지 분석기법을 이용하여 수행되며 이러한 분석결과는 설계 안전성 검토로 할당된다.

설계를 바탕으로 시스템의 조립, 설치과정을 거침으로써 초기 시스템 요구사양에 대한 타당성 검증을 위해 시험 케이스(Test Case)를 형성하며 이 결과와 더불어, 운용 및 지원 위험성 분석, 각 하부시스템별 위험성 분석결과, 초기 단계의 안전성 평가 결과를 바탕으로 시스템의 안전성 보고서를 작성하게 된다. 이러한 시스템의 수명주기 동안 RAMS, 특히 안전성 관리를 통하여 시스템에 영향을 미치는 위험스러운 상황 및 위험요인을 규명함으로써, 위험성 기록(Hazard Log)을 관리한다. 이러한 위험성 기록은 시스템 안전성 관리 측면에서 중요한 부분을 차지한다.

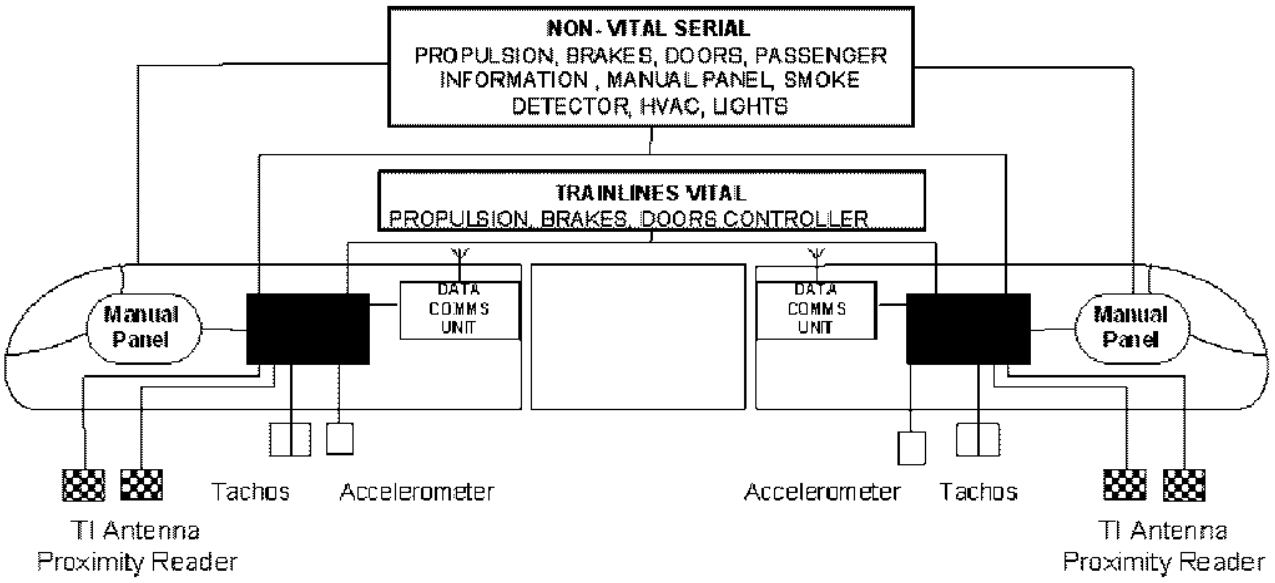
3.4 열차제어 시스템의 모델링

열차제어 시스템을 구분한다면 감시와 관리 레벨 시스템(Supervision and Management), 선로변 운영 레벨 시스템(Wayside operation) 및 동작 레벨 시스템(Activation)으로 구분할 수 있다.(그림 3.1 참조) 이중 동작 레벨 시스템(Activation)에서는 현장기기 제어 및 감시 시스템과 차상 감시 제어 시스템으로 나누어진다. 이중 임베디드 차상 감시 제어 시스템을 대상으로 그 소프트웨어의 고장 메카니즘을 활용하고자 한다.



< 그림 3.1 열차제어시스템 >

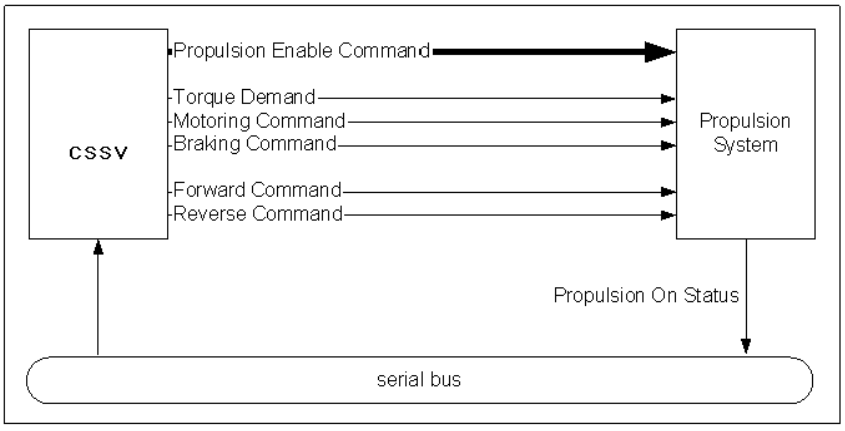
그림 3.1에서의 차상 감시 제어 시험(CSSV : Control and Supervision System on Vehicle)은 그림 3.2 와같이 나타내어진다.



< 그림3.2 CSSV : Control and Supervision System on Vehicle >

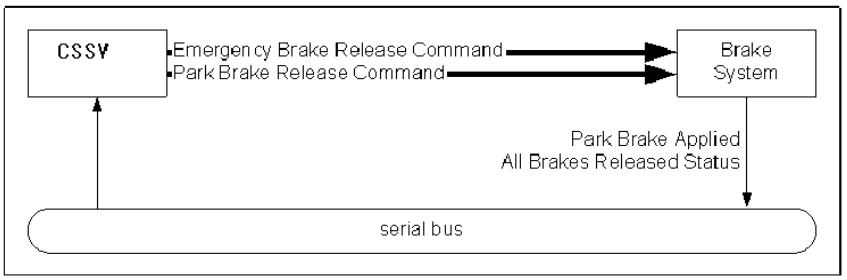
CSSV는 그림 3.2에 나타난 바와 같이 수동조작 판넬, 경보시스템, 제동기기, 추진시스템 및 차량문 제어 시스템과 인터페이스 되고있다. CSSV의 하드웨어는 임베디드화되어 보드상에서 설계되어져 이에대한 신뢰성검사를 통해 검증되지만 소프트웨어는 실제 입출력을 명확히 하여 기능시험을 통해 고장 메카니즘을 설정하여 검증을 해야 한다.

CSSV와 추진 시스템간의 입출력은 그림 3.3과 같다. 여기서 추진 시스템으로부터 직렬버스를 통해 입력되는 정보는 상태정보이다.



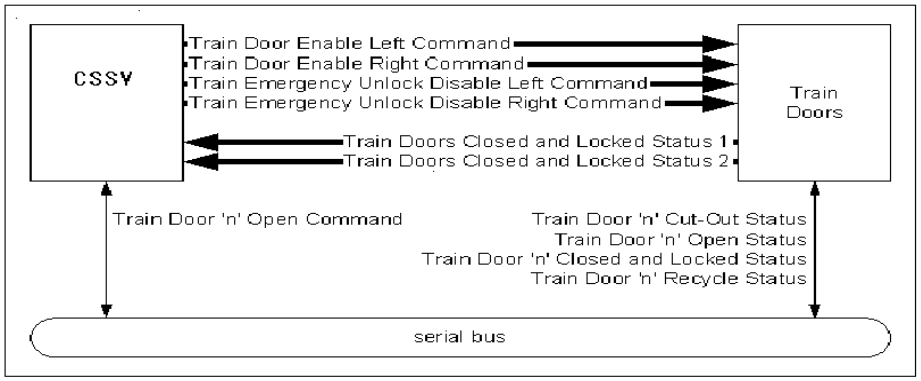
< 그림 3.3 CSSV와 추진시스템간의 Interface>

그림 3.4는 CSSV와 제동 시스템간의 인터페이스를 나타낸다.



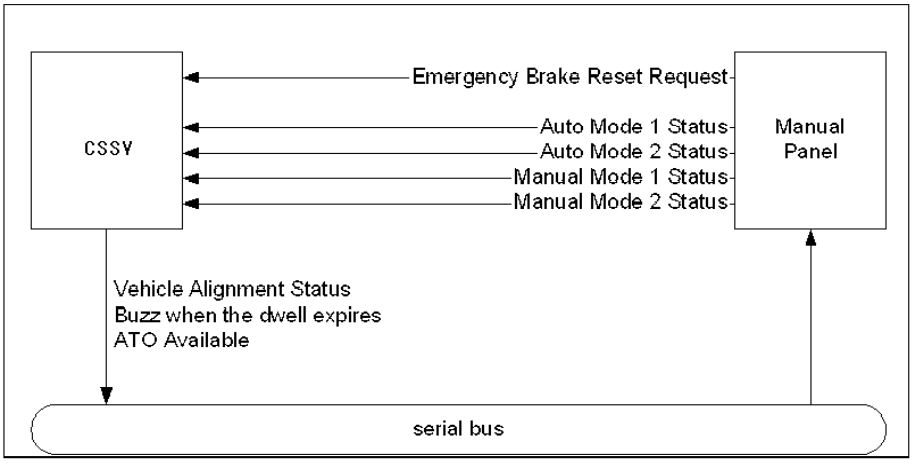
<그림 3.4 CSSV와 제동시스템간의 Interface>

그림 3.5는 CSSV와 차량문 제어 시스템간의 인터페이스를 나타낸다.



<그림 3.5 CSSV와 차량문제어 시스템간의 Interface>

그림 3.6은 CSSV와 수동조작 판넬간의 인터페이스를 나타낸다.



<그림 3.6 CSSV와 수동조작판넬 시스템간의 Interface>

임베디드 소프트웨어가 가질 수 있는 고장 메카니즘은 단위기능 뿐만 아니라 기능과 기능의 전이관계, 특수한 외부환경 요소의 변화 및 그 변화에 의해 변화되는 입출력에 의해서도 나타낼 수 있다. 기능 이전 또는 기능 전이간에 일어날 수 있는 고장 메카니즘은 임베디드 시스템이 개발목적에 따라 각기 다른 형태로도 존재하기도 하므로 시스템별로 각각 고장 메카니즘을 구축하는 것이 중요하다.

4. 고장 메카니즘 기반 테스트 절차 및 생성전략

본 절에서는 위의 3절에서 언급되어진 CSSV 임베디드 시스템의 소프트웨어의 고장 메카니즘을 바탕

으로 하여 테스트 절차를 확립하고 시험 경우의 생성전략을 나타낸다.

4.1 입력되는 자료의 정확성 검사

입력자료 처리조작의 고장 메카니즘 중에는 AD변화시 발생할 수 있는 참조오류가 있다. 이는 센서와 같은 범위를 갖는 입력이 존재하는 경우 센서입력 전압과 매핑되는 전압이 있을 경우 이때는 실제 입력 전압과 매핑되는 실제값들이 존재하는 경우에 발생할 수 있는 고장 메카니즘이다.

보통 이러한 것들은 AD값을 인덱스로 하는 테이블의 형태로 입력되어져 있으며 열차제어 시스템에서는 그림 3.2에 나타난 바와 같이 가속매타와 회전속도매타에서 야기되는 입력값들이다. 만약 이중 하나의 데이터가 잘못 입력되었다면 테스트케이스를 통해서 찾아내는게 아주 어려울 것이다. 이는 입력되는 데이터의 정확성을 미리 전제로 로직을 구성하기 때문이다. 따라서 임베디드 소프트웨어가 참조하고 있는 데이터들 간의 상호관계 즉 고속에서의 회전속도매타의 변화와 가속매타와의 상관관계를 미리 참조하여 스펙에서 정의하고 있는 값들과 비교하는 것이 우선순위이다. 이러한 테이블상의 2개값을 비교하는 단순한 작업이 오류를 찾아내는 시험 경우들을 줄일 수 있게 해준다.

4.2 단위로직 무결성

앞서 언급한 바와같이 시험은 작은 단위로부터 시작하여야 한다. 본 연구에서는 3절에 언급한 바와같이 열차제어 시스템의 소프트웨어를 작은 단위의 로직으로 구분하였다. 또한 각 로직에서 발생할 수 있는 일반적인 고장 메카니즘에 관하여 알아 보았다. 그러나 제반 고장은 이러한 일반적인 고장 뿐만아니라 시스템의 기능에 따라 일어날 수 있는 고장 메카니즘을 파악하여야만 한다. 이것은 로직이 구동되는 메카니즘을 파악해야 된다. 또한 고장 메카니즘이 파악되었으면, 이를 추적할 수 있는 시험경우들을 생성해야 한다. 예를들어 입력 데이터 처리 로직에서 입력장치 고장시 발생할 수 있는 문제점을 처리하기 위한 Fail-Safe 로직이 존재하는가를 알기 위해서는 입력값을 최고의 극한 값으로 줄 수 있는 시험경우를 생성해야 한다.

그러나 단순히 Fail-Safe 기능이 동작하는 입력신호만으로 시험경우를 구성하면 Fail-Safe 로직 동작중에 정상적인 신호가 입력되었을 때 다시 입력 데이터 처리 로직이 실행되는지 여부를 파악할 수 없기 때문에 반드시 정상입력신호와 함께 조합하여 시험경우들을 생성해야만 한다.

제어로직 중에는 분기문에의해 동작되는 로직이 있다. 이러한 로직을 일반적인 관점에서 시험 할 경우에는 로직이 분기될 수 있는 입력신호를 기준으로 시험경우가 생성될 것이다. 그러나 분기시 시간차이 고장 메카니즘과 같이 로직의 분기가 올바르게 동작되는 것과 별개로 분기문이 반복 수행 된다면 시간에 오차가 발생한다. 이것은 일부 임베디드 시스템이 내부의 클럭을 갖지 않고 소프트웨어적으로 메인루프가 수행되는 횟수로 시간을 산출하는 방식을 사용하기 때문이다.

이러한 오류는 시간의 오차가 누적되어 나타나는 것이기 때문에 한번 분기문을 수행하는 시험경우로서는 찾아내기가 아주 어렵다. 따라서 메인루프 카운터로서 시간을 측정하는 임베디드 소프트웨어가 있다면, 각각의 조건에 따라 수행되는 분기문을 반복적으로 수행할 수 있는 시험경우를 작성하고 또한 자원의 누수현상도 분기문의 시간오류와 비슷한 맥락에서 시험경우를 생성한다.

소프트웨어의 변수저장소로 사용되어지는 메모리의 미해제 고장 메카니즘으로 인해 반복수행시 메모리 누수가 일어나 결국에는 시스템 중단이라는 사태가 발생 되기도 한다.

임베디드 소프트웨어는 하드웨어를 직접 제어한다. 하드웨어로서는 열차제어 시스템 경우에는 Non Vital 기기 및 Vital 기기로 출력되는 데이터가 존재한다. Vital 이든 아니든 출력되는 정보에 방향성이 있는 경우에 특정방향으로 조절하기위해 움직이다가 반대방향으로 동작하라는 명령의 신호가 들어온 경우 변경하는 기준이 상 또는 하 부분에서 한점으로 존재한다면 이는 기기들의 기계적인 특성으로 인하여 히스테리시스현상(떨림현상)이 발생된다. 따라서 방향에 따라 변경되는 구동점을 일정기준 만큼 차이를 두어야 하고 시험경우에서는 이를 바탕으로 하여 방향성을 고려하여야 한다.

4.3 연관된 로직의 무결성 검증

sean은 요소별로는 올바르게 작동한다고 하더라도 요소들이 관계에 의하여 발생하는 다양한 고장을 열거하면서 왜 어디에서 고장이 났는가를 지적하고 있다. [2]

따라서 단위로직별로 검증을 수행한 후에 로직간에 발생할 수 있는 고장 메카니즘을 고려하여 시험을 수행하여야 한다. 임베디드 소프트웨어를 단위로직들로 도식화 하면 로직간의 연관관계를 알아 볼 수 있다. 서로 연결되어져 있는 로직은 로직간 입출력 데이터의 불일치로 인해 오류가 발생할 수 있으며, 동일한 출력기기들을 사용하는 로직인 경우에도 로직간에 오류가 발생할 수 있다. 로직간의 고장 메카니즘은 주로 변수들의 오류나 공용으로 사용하고 있는 전역변수의 처리과정 또는 메모리의 불일치로 인해 오류가 발생한다. 메인루프가 계속 수행하다가 특정 입력장치로부터 신호가 들어오면 메인루프에 인터럽트를 걸어 입력신호에 해당되는 값을 할당하여 처리되게끔 한다. 만약 특정로직이 인터럽트 이전의 변수나 메모리를 사용하여 실행되고 있는 동안에 인터럽트에 의해 변수의 값들이 바뀐다면 바뀐 값으로 인해 처리연산결과가 틀릴 수 있다. 따라서 시험경우는 이러한 인터럽트에 의하여 발생할 수 있는 고장 메카니즘을 고려하여 같은 입력신호를 로직의 수행순서와 시간별로 반복해서 실행될 수 있도록 생성되어야 한다.

4.4 시나리오 기반 시스템 테스트

임베디드 소프트웨어가 참조하는 자료의 무결성 검증, 단위로직에 대한 개별적인 검증, 로직간의 무결성 검증을 통해서 오류들을 검증했다고 해도 모든 오류들을 검증한 것은 아니다. 실제로 임베디드 시스템에 입력되는 요소들은 무수히 많은 조합이었기 때문이다. 그러나 이 모든 것들을 다하기엔 불가능하므로 특정 시나리오를 바탕으로 시험경우를 생성해서 검증하는 것이 타당하다. 시나리오는 일반적인 상황을 표현할 수 있는것 뿐만 아니라 특이한 상황 즉 고장이 발생할 만한 상황들을 표현할 수 있는 것들도 포함되어야 한다.

이러한 점들을 기반으로 열차제어 임베디드 시스템의 소프트웨어에 대한 시나리오 기반 시험경우를 제시하면 ..... 같다.

5. 결 론

본 연구에서는 열차제어 시스템의 임베디드 소프트웨어를 검증하기 위하여 시스템 사양서가 정의하는 가장 작은 단위인 단위로직으로부터 시스템에 이르기까지 점진적인 검증방법을 제시하고 전체적인 고장 메카니즘을 파악하기는 어렵지만 임베디드 소프트웨어를 단위 로직으로 잘게 나누어 각각에서 발생할 수 있는 고장 메카니즘을 찾아 이를 추적할 수 있는 시험경우를 제안하였다.

고장 메카니즘을 추적해 간다는 목적을 지니고 시험을 진행하였기 때문에 오류검증 능력을 높일 수 있었고 오류에 구조적 접근이 가능하여 오류의 원인 파악이 용이하게 되었다. 철도제어 시스템이 거의 대부분 임베디드화 되어 블랙박스로 간주되었지만 내부의 단위로직을 구분하고 이의 시험경우들을 제시하고 고장 메카니즘을 분석하고 오류의 원인을 찾아 신뢰성을 높일 수 있었다. 향후 특성화된 시나리오를 개발하여 시험 시간과 경비를 줄일 수 있는 방법에 관해 연구하고자 한다.

참고문헌

[1] Boyer,S.A. SCADA: Supervisory Control And Data Acquistion,ISA-The Instrumentation , Sytem and Automation society ,2004

[2] Operation theory research organization, Operation theory ,2002

[3] Rail Safety and Standards Board " Engineering Safety Management" (the yellow book)

[4] J.C.Ruiz, Y.Pedro, L.Lenin " On Benchmarking the dependability of Automotive engine control

unit ", Proceedings of DSN'04,2004
[5] 이상용 , 장종순외 " 임베디드소프트웨어의 검증에 관한 연구", 대한 산업 공학회 Proceeding. 2004
[6] Tordai,L. "Common Safety Indicator and Common Safety Targets of European Railways", WCRR 2006, Montreal, Canada,4-8 June 2006