컴퓨터를 활용한 철도제어 시스템에서의 안전성 및 신뢰성 확보방안



이종우 신덕호

1. 서 론

컴퓨터를 활용한 제어 시스템은 우리들의 생활 속에 깊숙이 관여하고 있다. 컴퓨터를 사용한 제어는 가전제품, 휴대폰 항공기, 철도 등 다양한 산업분야에서 응용되고 있으며, 그 적용분야를 넓혀가고 있고, 성능향상이 급속히 진행되고 있으며, 인간생활의 의존도가 점점 커지고 있다. 따라서 컴퓨터의 안전성과 신뢰성은 현재 사회에서 가장 중요한 문제로 부각되고 있다.

최근에 전국적으로 인터넷 망에 바이러스가 침투를 하여, 인터넷을 마비시키는 상태가 발생하여, 상당히 큰 피해를 경험하였다. 이와 같이 컴퓨터 제어 시스템의 기능이 정지되거나 오작동을 할 경우에는 사용자에게 상당한 피해를 줄 수 있다. 따라서 컴퓨터 시스템의 신뢰성과 안전성은 사회전반에 영향을 미칠 수 있으므로 반드시 확보해야할 목표 중에 하나이다.

시스템이 안전하다는 것은 사람에게 상해를 주지 않고, 시스템에 손상을 입히지 않는 것으로 정의할 수 있다. 시스템에서 결함이 발생하여, 시스템 고장상태가 발생하는 경우에 사람에게 상해를 주지 않고, 시스템에 손상을 입히지 않는다면 이 시스템은 안전성이 높다고 이야기 할 수 있다. 신뢰성은 기기와 시스템이 목표하는 임무를 올바르게 수행할 수 있는 능력으로 정의한다. 안전성과 신뢰성은 깊은 관계가 있지만 다른 개념이다. 신뢰성을 향상시키는 것으로 위험 측 고장의 발생확률 자체를 작게 할 수 있고, 고장 시의 기능 저하에 기인하는 취급(처리) 오류 등으로 인한 사고 발생확률을 작게 할 수 있다. 그러나 아무리 신뢰성을 높이더라도 고장은 피할 수 없으므로, 대책을 염두에 둘 필요가 있다. 때로는 신뢰성을 희생하더라도 안전성을 확보하지 않으면 안 되는 상황도 있다.

본문에서는 컴퓨터를 이용한 제어 시스템에서 신뢰성과 안전성 확보방안에 대하여 방법적인 접근을 하였다. 첫 번째로는 컴퓨터제어 시스템에서 결함이 발생하였을 때 검지 방법에 대해서 논하였으며, 두 번째와 세 번째는 하드웨어와 소프트웨어 신뢰성과 안전성 확보

를 위해 현재 사용되고 있는 방법에 대해서 논하였다.

2. 안전성과 신뢰성

안전성(safety)이란 Webster사전에 의하면 "the quality or condition of being safe" 이고, 안전(safe)은 "free from damage, danger or injury ; secure" 라고 말하고 있다. 즉, 품질에 관한 하나의 척도로서 안전성을 생각할 수 있지만 이 안전성이라고 하는 결과에 대한 원인은 대부분의 경우 제품에 있어서 신뢰성 특성의 결여로 일어난다.

안전성이란 "인간의 사상 혹은 자재에 손실 혹은 손해를 주는 상태가 없는 것"이라고 말한다. 또 신뢰성에서는 임무수행을 위한 기능상의 고장을 대상으로 하지만, 안전성에서는 인간 · 자재에 손실 · 손해를 주는 위험한 상태를 대상으로 한다. 극단적인 예로서, 제품의 신뢰성이 떨어져서 고장이 자주 발생한다 할지라도 인간의 사상 혹은 자재에 손실이나 손해를 주는 상태가 일어나지 않는 다면 안전성은 높다고 할 수 있다.

신뢰성(도)은 시스템이 정상적인 작동을 하는 확률이며, 안전성(도)은 인간의 사상 혹은 자재에 손실 혹은 손해를 주는 상태가 없을 확률로 정의 할 수 있다.

2.1 안전성의 요구수준

미국의 스탈은 자발행위와 외부로부터의 재난(피재)으로 나누고, 어느 정도 위험에 노출되고 있는지, 혹은 수용할 수 있는지를 나타냈다. 그 일부를 표1에 나타냈다. 교통사고와 자연사를 제외하고, 자발행위에 의한 사망률은 피재보다 100배 정도 높다. 흡연처럼 사람은 자기의 즐거움을 위해서는 상당히 큰 위험을 무릅쓰고 있지만, 교통사고 등의 피재는 리스크가 높다고 중요시한다. 결론적으로 자발행위는 스스로 제어가 가능하다고 생각을 하며, 피재는 제어가 불가능하다고 생각을 하기 때문이다.

교통사고 등을 생각했을 경우, 1일 24시간 위험에 노출되고 있을 리는 없다. 영국의 크레소는 일정의 업무와 행위에 직접 노출되고 있는 시간당의 사고사(事故死)를 만날 확율「FATR」(Fatality Accident Frequency Rate)를 정의하고, 각 업계를 비교하였다.

안전성의 요구수준으로서 제안되고 있는 수치 예를 표2에 나타내었다. 이제부터 사망을 동반하는 위험율은「」(1시스템당 10 ~ 100만년에 1회)으로 사회적인 기준에서 보아 우선 타당한 값이라고 말할 수 있다. 따라서 철도 신호시스템에서 요구되는 수준도 이와 같은 수준에서 요구된다.

〈표 1. 자발행위와 피재의 사고사망비율〉

자발행위($\times 10^{-5}/h$)		재난($\times 10^{-7}/h$)	
흡연	500	교통사고	500
자동자운전	17	홍수	7
이륜원동기운전	2000	지진	7
축구	4	토네이도	22
		비행기추락	1
		원자력발전사고	1
		낙뢰	7
		자연사	125,000

〈표 2. 각종의 안전성 요구수준〉

	허용된 위험율
미국	자발행위 : /년 인 재해피해 : /년인
영국	프로세스제어 : /년 시스템 (90% 99%의 신뢰도 달성)
오스트리아	화학공업 FAFR : 3.5 시스템내의 개개의 위험도 : 0.35
	무인화 수송시스템 : $10^{-12}/h$ 인간관여 시스템 : $10^{-9}/h$
원자로	주변의 방사능 확산 : /년/로 (노심용융사고: 격납용기:)
NASA 항공기 제어용 컴퓨터	위험측 고장율 : $10^{-9}/h$
점보여객기	이착륙회수 : 150 만회

2.2 철도신호 시스템의 신뢰성 요구수준

신뢰성은 주어진 시간 동안에 기기가 정상적으로 작동하는 시간을 말한다. 신뢰성을 측정하는 척도로서

MTBF(Mean Time Between Failure)가 사용되며, MTBF는 첫 번째 고장이 발생하기까지 시스템이 정상 동작하기로 기대되는 시간을 말하는 것으로 일반적인 의미는 시스템 고장이 발생할 때까지의 평균수명을 말한다. 대부분의 기업에서는 MTBF를 시스템 신뢰도의 평가척도로 활용하며, MTBF를 얻기 위해 MIL-HDBK-217이나, Bellcore 등의 국제규격을 근거로 고장율을 예측하거나 가속수명 테스트를 이용하여 시스템의 고장률을 추정하는 방법으로 데이터를 얻는다.

〈표 3. 주요 신호시스템의 MTBF〉

기기명	내용
CTC	$2.5\times 10^{5}(H)$(MTBF)
AF 궤도회로	$2.5\times 10^{5}(H)$(MTBF)
전자연동장치	$1\times 10^{5}(H)$(MTBF)
건널목 제어유니트	$1.4\times 10^{5}(H)$(MTBF)
전자폐색유니트	$1.2\times 10^{5}(H)$(MTBF)

3. 결함허용

fault tolerant - 결함이 발생하더라도 자동적으로 상쇄하여 시스템고장을 막고, 지속적인 서비스를 제공하는 디바이스 결함은 그림1과 같이 발전된다. 물리적인 결함 즉, 소자의 파괴 또는 외부의 잡음으로부터 발생된 결함은 시스템이 동작하는 동안의 정보를 변화시켜 올바르지 않은 연산을 수행하게 되는 오류로 발전하고, 이 오류에 의해 외부로의 잘못된 출력이 발생되는 것을 고장으로 표현한다.

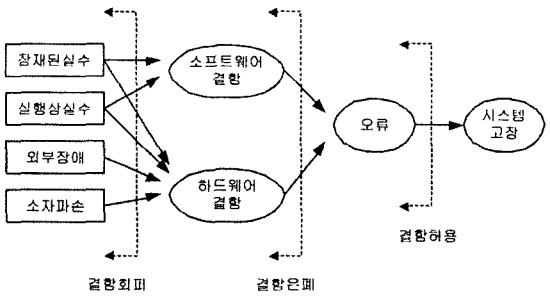
결함허용은 시스템 내부에 발생한 결함을 극복하여 시스템이 목표하는 동작을 계속 수행하는 것을 목적으로 하기 위해 그림2에서와 같이 결함회피와 은폐 그리고 허용을 하도록 설계한다.

〈그림 1. 결함의 발전단계〉

설계단계부터 잠재하는 실수 또는 운영중의 오조작, 외부로부터의 장애, 소자의 파손 등이 모두 결함발생 원인이 된다. 이러한 결함발생 원인의 최소화를 위해 설계단계에서부터 결함발생원인을 예측하여 예방하는 활동을 결함회피라고 한다. 소자의 노후나 복잡한 소프트웨어에 잠재하는 결함으로 인하여 소프트웨어 또는 하드웨어에서 결함이 발생되는 경우에는 발생된 결함이 오류로 발전하지 못하게 하기 위해 결함을 격리시켜서 더 이상의 확산을 방지하는 결함은폐가 사용된다. 결함의 은폐는 결함검출회로 또는 결함검출소프트웨어를 사용하여 결함발생즉시 검출하고, 결함이 검출된 부분을 완전히 격리하는 과정을 이야기한다. 마지막으로 결함허용은 오류로 발전된 결함을 여분을 사용한 다중계방법 등으로 극복하는 단계로 이러한 일련의 결함회피, 은폐, 허용의 단계를 수행하여 시스템의 고장을 최소화 시켜야 한다.

3.1 신뢰성, 안전성 및 이중화 시스템과의 관계

신뢰성, 안전성은 시스템에 대한 요구가 고도화함에 따라 그 구조는 복잡화하며 그와 더불어 신뢰도와 안전도가 저하되는 것은 불가피해지고 있다. 더욱이 시스템 고장이 가져오는 영향도 증대함으로서 시스템의 신뢰도, 안전도에 대하는 요구도 또한 엄격하게 된다. 이러한 배경으로 인해 고신뢰성 및 고안전성 기법 적용의 필요성이 높아지고 있다.



〈그림 2. 결함의 발생과 극복의 단계〉

오래 전부터 신호보안 시스템은 목표동작의 중요성으로부터 고신뢰화 및 고안전성이 적극적으로 도입되어 다양한 기법을 다른 산업분야에 비해 앞장서서 수용하고 있다.

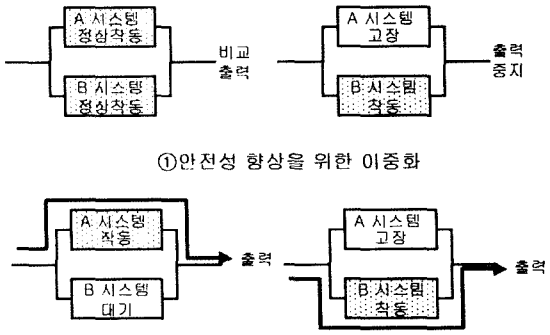
고신뢰성으로의 접근에는 2가지 방법이 있다. 첫 번째는 신뢰도가 높은 부품의 채용이다. 실용실적으로 높은 신뢰도가 인증된 것을 채용하여 고유의 신뢰도를 높이는 것이다. 두 번째의 방법은 다중계의 채용이다. 구성품의 결함발생은 물리적인 한계가 존재하므로 결함을 허용하되 그것이 시스템의 고장으로 발전하지 않도록 하는 것이다. 단일 시스템에서 고장이 발생하면 다른 시스템으로 절체를 함으로서 시스템의 기능을 유지하는 방법이다.

고안전성을 확보하기 위해서는 신뢰성과 동일하게 2가지 방법이 제안되고 있다. 첫 번째는 선천적으로 안전 특성을 갖는 부품을 이용하여 안전을 확보하는 방법이 사용되며 예를 들어 과전류 보호시스템에서 퓨즈와 같이 정격이상의 과전류가 도통을 하면 퓨즈가 융단 되어 자동적으로 전류를 차단하여 시스템을 안전측으로 작동하는 방법이다. 또 다른 방법의 하나는 다중계를 구성하여, 서로의 결과를 비교하고, 비교 값이 일치하지 않는 경우에는 안전측으로 작동하여, 시스템을 정지하는 등으로 안전측 동작을 수행하게 한다.

4. 하드웨어 구성법

4.1 안전성 확보를 위한 하드웨어 구성법

신호시스템에 마이크로 컴퓨터를 도입할 때는 시스템이 페일 세이프 특성을 포함하도록 하기 위한 몇 가지 구성방법이 사용된다. 일반적으로는 하드웨어 이중화와 소프트웨어 이중화를 사용하여 구성된다. 하드웨어 이중화는 동일한 소프트웨어를 실장한 복수의 마이크로 컴퓨터가, 데이터를 비교하여, 비교값이 상이한



〈그림 3. 이중계에서 안전성 및 신뢰성 구현방법의 예〉

경우에는 출력을 안전측에 고정하는 것이다.

소프트웨어 이중화에서는 1대의 하드웨어에 복수의 소프트웨어를 실장하기도 하고, 처리는 데이터 등에 체크 부호를 부가하여 실시하여, 처리결과와 출력의 정당성을 보증하는 것이다. 페일 세이프와 폴트 시큐어는 출력의 정확함에 착안하는 점에서는 같지만, 페일 세이프는 고장 시에 적극적으로 안전상태로 전이시켜 그 상태를 유지하는 것에 비해, 폴트 시큐어는 고장나더라도 정확한 출력이 되는 동안은 결함을 허용하고, 잘못된 출력의 경우만 검출한다.

신호장치에 사용되고 있는 대표적인 방식을 다음에서 언급하였다.

(1) 버스동기식

복수의 프로세서가 공통의 클록으로 동기화 되어 동작하고, 동일한 처리를 실시한다. 내부버스상의 데이터를 머신사이클마다 페일세이프로 설계된 비교회로로 비교한다. 이상검지 시에는 시스템을 안전측으로 제어하는 회로가 포함되어 있다. 범용의 CPU와 LSI를 이용하여 비교적 쉽게 구성할 수 있기 때문에, 널리 이용되고 있다. 일본에서는 연동장치, ATC(자동열차제어장치), 건널목 등에 널리 채용되고 있다. 독일의 연동장치도 같은 구성이다.

(2) 시간차동기방식

2대의 프로세서가 시간차를 가지고 동작하고, 처리결과를 비교회로로 출력하는 구조로 비교회로를 주기적으로 일치/불일치상태로 하게 하고, 다이너믹 동작에 의해 비교회로를 포함한 진단을 실행한다. 비교회로를 페일세이프로 할 필요가 없는 이점이 있다.

(3) 프로그램 동기식

복수의 프로세서가 프로그램 레벨로 동기하여 동작하고, 데이터링을 통하여 처리결과를 비교한다. 상기 (1), (2)만큼 비교빈도는 높지 않고, 또 비교를 위한 오버헤드도 크지만, 외부에 전용의 비교회로를 가질 필요가 없는 이점이 있다. 영국의 연동장치 등에서 이용되고 있다. 또 최근에는 일본의 연동장치에서도 이 방식이 채용되고 있다.

(4) 소프트웨어 다양화

독립된 팀이 설계, 작성한 복수의 소프트웨어를 삽입하고, 그것들의 연산결과를 비교하는 것으로 페일세이프를 확보하는 방식이다. 스웨덴의 전자 연동장치에 채용되었다. 프로그램은 2개이고, 출력의 비교는 외부의 페일세이프 특성을 갖는 오류 검지부에서 행해진다. 하드웨어의 이중화가 필요 없지만, 작성한 복수의 소프트웨어가 독립성을 어떻게 유지하는가, 소프트웨어의 제조, 보수의 비용의 증대를 어떻게 억제하는가가 과제이다.

(5) 부호화 프로세서의 채용

프로그램과 데이터에 잉여부호를 부가하고, 소프트웨어에 의해 하드웨어의 고장도 검지하는 방식으로서, 미국의 전자 연동장치(VPI)와 프랑스의 열차 제어시스템에 채용되고 있다. 단 프로세서로 실현할 수 있고, 그 안전성은 부가된 부가부호의 길이와 비례한다.

위에서 제시된 방법 이외에도 여러 가지 구성방법이 있지만, 이것들을 기본구성으로 하여 페일세이프를 확보하고 있다. 시스템의 가용도를 향상시키기 위해서는 앞에서 언급한 구성을 기본으로 핫 스탠바이와 3중계 다수결 구성이 채용된다.

4.2 신뢰성을 위한 하드웨어 구성법

신뢰성을 확보하기 위한 하드웨어 구성법은 여러 가지 방법이 존재하지만 기본적인 원리는 다중계 시스템을 구성하여 시스템 결함이 발생하였을 때 시스템을 절체하는 방법이다. 신뢰성하드웨어 구성법으로 대기 2중계, 동적대기 2중계, 비동기 병렬 2중계, 동기 병렬 2중계 및 2 out of 3등의 방식 등이 있다. 그림 4에서는 다중계 구성을 통한 신뢰성 향상방법을 제시하고 있다.

(1) 대기 2중계

대기 2중계 방식에서 대기계는 Warm Standby 상태로 존재하여 동작계의 차단시에만 임무를 수행한다.

(2) 동적 대기 2중계

동적 대기 2중계에서는 대기계의 프로세서가 동작계의 정상동작 지원 및 고장감시를 분담하며, 동작계의 결함발생을 감시한다.

(3) 비동기 병렬 2중계

비동기 병렬 2중계에서는 프로세서가 각각 임무를 수행하며 상호비교를 통하여 서로의 결함발생을 감시한다. 비교값 불일치시 다른 시스템으로 절체하며, 병렬 2중계 구성으로만 동작시 2회까지 비교값을 재검사하여 불일치시 2중계의 출력을 모두 차단한다.

(4) 동기 병렬 2중계

동기 병렬 2중계에서는 프로그램 레벨에서 동기를 하며, 양계의 출력은 별도의 회로인 이중계 제어장치

에 의해 감시되어 비교값이 상이한 경우에 위의 비동기 병렬 2중계와 동일하게 처리된다.

(5) 2 out of 3

2 out of 3는 3계중의 구성에서 2계로 병렬 이중 동기운전을 실시하며, 나머지 1계는 오프라인으로 대기한다. 2중계 비교값이 불일치 하는 경우 자동 재구성하며, 결함이 발생한 계를 제외하고 동작한다. 이때 결함이 발생된 계를 찾아내기 위해 자기검사 특성을 갖아야 한다.

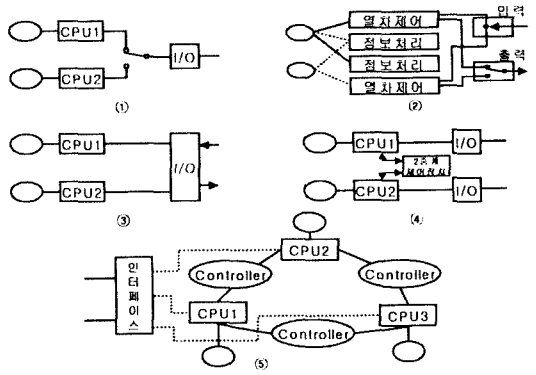
5. 컴퓨터 제어신호시스템 소프트웨어

5.1 소프트웨어의 안전성 · 신뢰성 향상

일반적으로 고안전성 및 고신뢰성 시스템 실현방법으로서, 구성요소의 품질을 높이고, 결함이 일어나지 않도록 하는 결함회피와 결함의 발생은 불가피하다 라는 전제에서 시스템 적으로 대처하는 결함허용의 두 가지 방법이 있다. 시스템 소프트웨어 분야에서도 안전성과 신뢰성 향상을 위한 기술을 다음과 같이 분류할 수 있다. 또 소프트웨어의 개발에서 운용 · 보수까지의 라이프싸이클을 대상으로 각 공정에서의 관리체제도 안전성과 신뢰성을 확보하는데 중요하다.

5.2 소프트웨어 결함회피(Software Fault Avoidance)

결함회피는 소프트웨어에 버그, 즉 잠재결함이 없는 것을 지향하는 것이다. 시스템을 사용하기 전에 결함을 찾아서 제거해주는 비용보다 시스템을 사용하는 도중에 결함을 찾아서 제거하는 경우가 비용이 더 적게 소요되는 경우도 있다. 그러나 안전과 직접적 관련이 있는 소프트웨어의 경우에는 시스템 운영이전에 소프트웨어의 결함을 찾아내는 것이 매우 중요하다. 결함회피를 위한 방법으로는 구조화 프로그래밍과 데이터형이 있다.



〈그림 4. 다중계 구성법 (① 대기 2중계, ② 동적대기 2중계, ③ 비동기 병렬2중계, ④ 동기병렬 2중계, ⑤ 2 out of 3)〉

(1) 구조화 프로그래밍

while루프와 if명령만으로 이루어지는 프로그래밍 혹은 톱다운 방법에 의한 설계 등의 총칭이며, 실행순서가 순차적으로 진행되는 프로그램이다. goto 문을 사용할 경우에는 프로그램의 진행순서가 조건에 따라 뒤바뀌어지기 때문에 쉽게 프로그램의 절차가 눈에 쉽게 띄지 않는다. 구조화된 프로그래밍 기법은 결함을 회피할 수 있는 방법의 하나로서 사용되고 있다. 구조화 기법을 사용한다 할지라도 결함을 피하기 위해서 다음 사항을 유의해야 한다.

① 부동소수점형의 수치

정밀도 보증이 충분하지 않고, 수치 끼리를 단순 비교하는 방식은 문제를 일으킬 수 있다. 따라서 고정소수점형의 수치는 필요한 행수가 일치된 상태에서 정확한 비교를 수행해야 한다.

② 포인터

기억장치상의 특정 영역을 참조하기 위한 것이며 동일한 데이터를 다른 이름으로 사용하는 것을 용인한

다. 처리상은 편리하지만, 프로그램의 분석이 용이하지 않다.

③ 병렬처리

병렬처리 되는 프로세스 상호간의 미묘한 타이밍 문제가 발생하기 쉽고, 프로세스간의 의존관계를 최소한으로 억제하여 신중하게 처리되어야 한다.

④ 재귀 호출

어느 서브루틴이 자기 자신을 호출하기도 하고, 서로 다른 서브루틴이 상호 호출하는 구조로 설계된 경우를 말한다. 재 호출에서는 간결한 프로그램으로 되지만, 그 로직을 추적하는 것이 상당히 어려워 짐 과 동시에 에러가 발생했을 경우에는 스택변수가 얽히기 때문에 시스템 메모리 전체의 점유를 야기할 수 있다.

⑤ 인터럽트

현재 실행중인 프로그램과는 다른 프로그램으로 처리를 강제적으로 옮기는 수단이다. 인터럽트는 중요한 프로그램의 실행을 도중에 중단시켜 버리게 되므로, 신중히 고려되어야 한다.

(2) 형식화기법

첫째, 데이터가 대상을 프로그램 상에서 어떻게 모델화 · 표현한 것인지를 명시적으로 나타내기 위한 도입과 둘째로는 프로그래밍 언어에 의한 표현 · 기술이라는 입장으로부터 프로그램 중에 나타나는 형식이 취할 수 있는 값에 관해 제약하는 형식이 도입되었다.

프로그램 언어를 대상으로 한 이 같은 형식화의 개념을 도입하는 것에 의해, 어떤 제약을 가진 대상을 다루고 있는가를 명확히 의식하는데 도움을 주고, 다른 형의 데이터를 대응시켜 버리는 것 같은 오류를 적게 할 수 있다. 게다가 프로그램 가운데의 각 식이 어떤 형을 가지는가를 주는 엄밀한 규칙을 정하고, 그 규칙을 프로그래밍 언어의 일부로 정하는 것에 의해 컴파일러 자동검출이 가능하게 되고 프로그램의 오류를 감소시킬 수 있다.

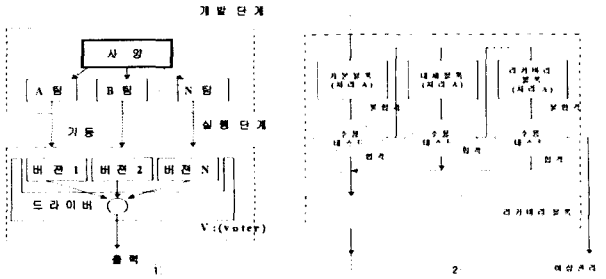
보다 강력하게 구조화된 프로그래밍 언어(Ada 등 추상 데이터형 정의기구를 가지는 언어)로는 구조화의 오류가 없고, 신뢰성이 높은 프로그램의 기술이 가능하게 된다. 그 반면, 프로그램의 설계, 코딩, 디버깅의 단계에서 구조의 일관성을 유지하도록 하지 않으면 안되고, 프로그램이 이중화로 될 경향이 있음과 함께 도중의 규정변경, 기능의 추가 등에 대해서 유연히 대처하는 것이 어렵다. 따라서 이해와 수정이 용이하도록 형식화가 추진되고 있다.

5.3 소프트웨어 결함허용(Software Fault Tolerance)

결함허용은 소프트웨어에 결함이 잠재하는 것은 불가피하다고 전제하고, 결함의 발생 방법과 외부로부터의 영향에 착안하여, 대처하려고 하는 것이다. 소프트웨어에 있어서도 하드웨어와 시스템의 분야에서의 경우와 동일하게 이중화의 개념을 도입하여 결함허용을 실현하는 기술이 존재한다. 구체적으로는 소정의 서비스를 제공하는 소프트웨어를 복수구성으로 하고, 에러가 검출되었을 때 다수결 논리로 정정하기도 하고, 대체 소프트웨어로 교체하는(교체기법)것에 의해, 외부에는 결함이 존재하지 않는 것처럼 보이는 것이다. 또, 결함에 의한 에러가 발생했을 때, 그 영향을 국소화하기 위해 시스템의 일부 기능을 정지하기도 하고, 분리하는 페일소프트 기술과 강제적으로 시스템을 안전측으로 천이시키는 페일세이프 기술을 포함할 수 있다.

(1) N버전 프로그래밍

동일규정에 대해서 다른 N개 버전의 프로그램을 작성하고, 동시에 처리하여, 각각의 결과를 다수결 논리에 의거하여 최종출력을 결정하는 것이다(그림5). 각 프로그램의 작성이 독립적으로 시행되고, 소프트웨어 결함에 상관이 없다면, 상호 출력비교에 의해 에러 검



〈그림 5. N 버전 프로그래밍의 개념과 리커버리블럭〉

출, 정정이 가능하다.

각 버전의 기동과 동기, 다수결 처리는 드라이버라고 불리는 시스템 프로그램이 시행된다. 드라이버의 에러는 치명적이지만, 처리는 단순하기 때문에 철저한 검사에 의해 잠재하는 결함을 배제할 수 있다.

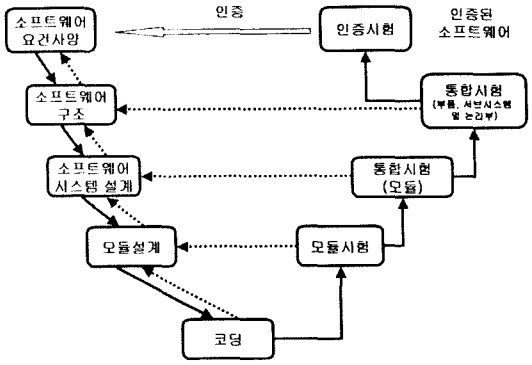
애매함을 배제한 규정과 함께 동일 알고리즘의 사용 방지의 고안에 의해 버전간의 독립성이 확보된다면 결함허용화에 유효하고 또한 적용하기 쉬운 기술이다.

⑵ 리커버리 블록

모듈의 처리가 종료될 때마다 연산결과를 체크하고, 에러가 검출되었을 경우에는 대체모듈로 다시 연산하여결함허용을 구현하는 방법이다. 상시 N개 프로그램 모듈을 실행시켜야 하는 N버전 프로그래밍과 다르고, 테스트 합격 시에는 다른 모듈을 기동시킬 필요는 없다.

그림5의 ②에서 리커버리 블록의 개념을 표현하였다. 각 모듈은 기본블록, 받아들일 테스트, 대체블록을 구성요소로서 가진다. 기본 블록에 의한 연산결과는 받아들일 테스트에서 체크된다. 테스트결과가 만족하면 다음모듈의 처리를 실행하지만, 불합격 시에는 초기치를 재 입력하여 다른 버전인 대체블록이 처리를 수행한다.

리커버리 블록 방법의 중심으로 되는 것은 대체블록에 의한 재처리 시에 필요한 데이터의 보존과 설정, 대체블록이 불합격으로 되었을 때의 시스템 적인 대응



〈그림 .6 소프트웨어 개발 라이프사이클(V 모델)〉

및 실시할 테스트이다.

5.4 소프트웨어의 라이프싸이클 관리

소프트웨어의 고안전성 · 신뢰성을 확보하기 위해서는 요구사항에서 운용 · 보수까지의 소프트웨어의 라이프사이클을 통한 관리가 필요하다. 각 공정에서는 각각의 내용과 완료기준의 이외, 필요한 문서화의 내용과 양식, 관리체제에 관해서도 명확히 결정되어야 한다. 소프트웨어 개발 라이프사이클 중 V모델이 그림 6에 나타내었다.

각 공정에서는 품질보증을 위한 기술이 적용되고 있다. 규정화된 공정에서는 소프트웨어에 대한 요구분석의 단계와, 그것을 구체적으로 정의하는 시스템 정의의 단계로 구별된다. 시스템 정의단계에서는 기능, 데이터, 제어의 흐름을 강조하는 부분인 상태 천이도와 부분적인 시행에 의한 사전평가인 프로토타이핑 등의 방법이 이용되고 있다. 또 설계공정에서는 계층화, 구조화 및 추상화 등에 의한 설계 방법론이 있다. 코딩공정에서는 구조화 프로그래밍, 자료흐름도 등이 채용되고 있는 외, 테스트 공정에서는 모듈서브시스템, 결합시험, 전수시험 등이 시행되고있다. 테스트의 방식에는 톱다운 시험법, 보텀업 시험법 등 이외, 버그의 검출누적성장곡선 등에 의한 에러의 생기빈도의 추정 등

소프트웨어 신뢰성 모델에 의한 정량화의 방법이 적용되고 있다.

또, 안전성이 요구되는 시스템에서는 FTA를 통하여 소프트웨어의 오류의 원인을 밝혀내고, FMEA를 이용하여 고장영향을 분석하는 리스크 분석도 필요하게 되고 있다. 이와 같은 소프트웨어의 관리 공정 모델에 관해서는 더욱더 연구과제로 되고 있고, 여러 가지 모델이 제안되고 있다.

6. 신호시스템 소프트웨어의 고안전성 · 신뢰성화 기술

마이크로 일렉트로닉스화된 철도신호에 있어서도, 동일한 방법으로 결함에 대한 안전성과 신뢰성향상을 위한 접근방법이 취해지고 있다. 실제로는 결함회피의 접근방법으로서, 설계 · 제작의 단계에서 하드웨어, 소프트웨어 모두 고 신뢰화 방법이 적용됨과 동시에 충분한 현장 시험이 시행되고 있다. 결함허용에 관해서는 장치의 목적과 신뢰성 요구기준에 따라 처리의 계속을 목적으로 한 여러 가지의 방법이 취해지고 있지만, 에러가 검출된 후에는 정지가 안전하다라는 철도의 특성을 이용하여, 하드웨어, 소프트웨어 모두 출력을 안전측으로 고정하는 안전 정지를 기본으로 하고 있다. 장치의 고장, 운용실수 등을 포함하는 입력의 에러 등에 대해서도 위험 측의 출력을 하지 않는 것이 요구되는 철도신호에서는, 에러검출이 아주 중요하고, 많은 방법이 취해지고 있다.

또 라이프싸이클을 통한 안전성 관리에 관해서도, 하드웨어의 경우와 같은 방법에 따라 시행되고 있다.

6.1 신호시스템의 결함허용 소프트웨어 기술

결함허용 철도신호 소프트웨어의 기본적인 논리는 다음과 같다. 일반시스템에 있어서의 소프트웨어의 결함허용 기술에 프로그램에 제약을 가지게 하여 버그를 회피하는 방법이 취해지고 있다.

(1) 안전성이 요구되는 기능과 그 외의 기능을 명확히 분리하고, 안전성이 요구되는 기능의 프로그램의 구조는 단순화하여 논리미스와 제작미스를 배제한다.

① 모듈화 설계를 하고, 각 모듈이 크게 되지 않도록 150스텝 정도로 제약한다. 또 프로그램의 구조화를 실시한다.

② 멀티태스크 처리를 취하지 않고, 싱글스레드 처리로 하고 정주기 타이머방식으로 한다. 또한, 인터럽트 처리는 피한다.

③ 안전측과 위험 측에서 프로그램상의 정보를 명확히 구분한다.

(2) 설계미스, 제작미스에 기인하는 버그를 제거하기 위해 철저한 이상시험, 안전성 시험, 현재 사용 주의 신호설비와의 동작비교를 위한 모니터 랜 시험, 현지시험을 실시한다.

① 전(全) 경로검사, 인터페이스와 타이밍체크, 이상처리 · 이상취급처리에 대한 시험을 실시한다.

② 소프트웨어를 개수했을 경우, 재차 기능 · 성능시험을 다시 한다.

〈표 4. 안전측과 위험측의 정보의 구분(전자연동)〉

	정보종별	논리 값 0	이론 값 1
입력	진로설정	복위	설정
	궤도회로	열차있음	열차없음
	선로전환기 상태	전환중	전환완료
내부 상태 출력	선로전환기 쇄정	쇄정	해정
	접근 · 진로 쇄정	쇄정	해정
	신호기	정지	진행
	선로전환기	현상유지	전환
	고장검출	고장	정상

안전측과 위험 측의 정보의 구분이란, 프로그램에서 논리 값의 사용이 통일되어 있지 않으면 설계 제작 시에 혼란을 초래하고 위험한 오 제어를 유발할 우려가

있기 때문에 논리 값 "0"을 안전측에 "1"을 위험 측에 할당하여 사용하는 것이다. 전자연동 시스템에서는 표 4처럼 배당하고 있다.

7. 결론

컴퓨터 제어 시스템의 안전성과 신뢰성의 확보방안에 대해서 살펴보았다. 위에 언급한 방법들은 안전성과 신뢰성을 확보하기 위한 방법들을 제시한 것이다. 안전성과 신뢰성을 확보하기 위해서는 먼저 요구되는 안전도과 신뢰도의 요구에 따라 구현방법이 달라진다. 또한 운용환경에 따라서 시스템의 안전도와 신뢰도가 변할 수 있으므로 운용환경을 고려해야한다. 객관적으로 안전성과 신뢰성이 확보되었다는 것을 입증하기 위해서는 인증체계를 갖추어야 하며, 그 해당하는 적절한 시험을 거쳐서 안전성과 신뢰성이 입증될 수 있다.

참고문헌

1. "Design and Analysis of Fault-Tolerant Digital Systems" written by Barry W. Johnson Edited by Addison-Wesley.1989.
2. "Fault-Tolerant and Fault Testable Hardware Design" written by parag K. Lala. 1985.
3. "Fail-Safe Inteface for VLSI : Theoretical Foundations and Implementation" Michael Nicolaidis, Member, IEEE Computer Society. Vol. 47. No. 1 JAN. 1998.
4. "Optimal Self-Testing Embedded Parity Checkers" Dmitris Nikolos, Member, IEEE Vol. 47. No. 3. MARCH 1998.
5. "Error Secure/Propagating Comcept and its Application to the Design of Strongly fault-Secure Processors" TAKASHI NANYA, Member, IEEE and TOSHIAKI KAWAMURA. Vol.37. No.1. JAN. 1988
6. "On-Line Detection of Bridging and Delay Faults in Functional Blocks of CMOS Self-Checking Circuits" Cecilia Metra, Michele Favalli, Piero Olivo, and Bruno IEEE. Computer-Aided Design JULY 1997 Vol 16. No. 7.
7. "Design of Totally Self-Checking Circuits for m-out-of-n Codes" IEEE Trans. Computers. Vol. 22.No. 3. pp. 263-169. Mar. 1973
8. "SAFETY-CRITICAL COMPUTER SYSTEMS" Written by Neil Storey 1996.