'87 韓國自動制御學術會議論文集 1987. 10.16~17

다중 프로세서를 이용한 제어기에서의 자체고장탐지

°신 영 달* 김 지 홍* 김 병 국** 정 명 진* 변 증 남*

*한국과학기술원 전기및전자공학과 **한국과학기술대학 전자전산학부

Fault Detection of the Controller based on Multiprocessor

°Y. D. Shin* J. H. Kim* B. K. Kim** M. J. Chung* Z. Bien*

*Dept. of EE KAIST, **Dept. of EC KIT

Abstract

The reliability is the critical issue in many computer applications, particulary in process control system. In this paper we describe how to achieve the reliability improvement in controller system based multiprocessor. The proposed method is accomplished by using the techniques of fault detection, fault isolation, safe action, and fault diagnosis.

1. 서론

최근에 이르러 반도체 및 컴퓨터 분야의 발전으로 자동화 시스템 또한 복잡하고 대규모화 되어가고 있으며 그의 안전성과 신뢰도의 증가는 필수적 요소로 등장하고있다. 또한 대규모 시스템을 제어하기 위하여 다중프로세서를 이용한 빠르고 신뢰도가 높은 성능을 내게하는 제어기에 대한 많은 연구가 수행되고 있으며, 이를위한 FT(Fault Tolerant)시스템이 요구되어진다. 지금까지는 원자력 발전소나 비행기의 제어기와 같이 고장으로 인한 피해가 매우 큰 제어 대상에 주로 적용되어 왔으나 최근에는 많은 분야에서 신뢰도를 높이는 문제가 요구되어 지고 있다.

시스템의 신뢰도는 고장의 발생빈도 및 정도에 의해서 결정되며, 신뢰도를 개선하기 위해서는 고장의 원인을 분석하고 규명하여 사전에 이를 예방하는 고장방지 및 고장 발생시 그 영향을 최소로 줄여 시스템이 정상적인 동작을 계속하도록하는 FT방법이 있다. 고장 방지 방법은 경험 및 분석에 의하며 한계가 있어서 고장을 완전히 없앨 수는 없다. 이보다 적극적인 방법으로 FT 시스템을 구성하는 방법이 최근에는 많이 쓰이고 있다.

FT시스템을 구성하기 위해서는 일반적으로 여분의 시스템을 지녀야 하며, 기본적인 과정을 첫째, 고장탐지 과정으로서 고장의 발생을 탐지하는 기능이 있어야 하며 둘째, 고장 진단과정으로 고장이 어느 부위에서 발생되었나를 판단하는 과정 셋째, 고장분리 과정으로서 고장이 다른 영역에 확산되지 않도록 고장부분을 고립시키는 과정 네째, 재 구성과정으로 고장이 난 부분을 분리 시킨후 여분의 하드웨어나 소프트 웨어를 고장이 발생한 부분과 대치 시키는 과정 다섯째, 고장 회복 과정으로 재 구성이 끝나고 나서 고장의 영향을 제거시키고 고장이 발생하기 이전의 상태로 최대한 전환하는 과정 여섯째, 재 실장 과정으로서 고장난 부분을 수리하여 다시 원 상태의 시스템으로 환원 시키는 과정이다.

위의 여섯 단계에 의해서 일반적인 컴퓨터의 FT시스템이 구성되어 지며, 제어기 에서는 고장에 의해서 출력된 데이타는 회복할 수 없기 때문에 고장회복 기능은 소프트웨어적인 고장에서 다음에 출력될 데이타를 수정해 줄 수 있다. 본 논문에서는 기존의 제어기에 간단히 첨가할 수 있도록 주로 하드웨어에 의한 고장을 다루게 되며, 소프트웨어적인 고장은 일반적으로 다루어질 수 있는 범위에서만 다룬다. 또한 소프트웨어 및 하드웨어 고장을 모두 탐지하여 여분의 시스템으로 대치하여 FT시스템을 구성해야 하지만 기존의 제어기가 대부분 여분의 하드웨어가 없으므로 FT시스템을 구성할 수 없기 때문에 여기서는 고장탐지 및 고장진단 그리고, 고장에 인한 위급한 상황에 처리하는 방법을 제안한다.

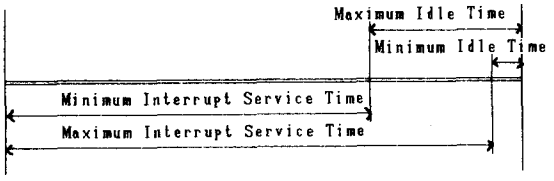

그림 1.1 일반적인 디지탈 제어기의 타이밍

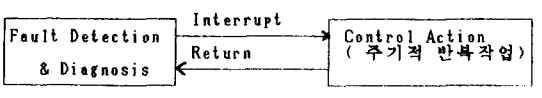

그림 1.2 디지탈 제어의 흐름도

고장탐지를 위해서 여러가지 테스트가 필요하여 시간이 요구되어 진다. 시간을 할당하는 방법에는 기존의 일정한 시간마다 전체 시스템의 동작을 점검하는 방법이 있는데, 이러한 방법은 실시간 제어에 있어서 일정하게 유지되어져야하는 주기가 변화할 우려가 있으며, 또한 이를 위하여 충분한 시간적 여유를 두어서 주기를 결정할 수도 있으나 주기는 빠를수록 좋기 때문에 좋은 방법이라고는 볼 수 없다. 따라서 여기서 제안하는 방법은 한 주기 내에서 제어기의 동작이 끝나고 다음 주기를 기다리는 시간에 자체 고장 탐지 기능을 수행하게 한다.

본 연구는 보일러의 기존 제어기에 여분의 제어기를 첨가하여 콜드 스탠바이(cold standby) 시스템을 구성하는데 있어서 요구되어지는 보일러 모델링 부분, 모델과 실제의 입출력 신호로 부터 고장을 판별하는 고장검출, 고장이 검출된 후 고장부분을 여분 시스템로 대치시키는 백업 스위칭 부분, 그리고 가장 신뢰도가 높아야 하는 여분의 시스템의 자체적인 고장검출 및 진단기능이 있으며 이는 여분의 시스템에 고장검출 및 보일러 모델 등의 고장을 검출할 수 있는 기초적인 데이타를 가지고 있으며, 또한 백업기능을 같이 가지고 있기 때문에 여분 시스템의 자체의 신뢰도를 높이는 방안에 대해서 연구하였다.

2. 고장탐지

(1) 다중 프로세서의 모델링

고장탐지 및 진단을 위해서는 시스템을 여러 부분으로 나눌 필요가 있다. 또한 다중 프로세서이기 때문에 각각의 프로세서가 별개의 ROM 및 RAM 등을 가진다고 할때 그림 2.1 과 같으며 고장탐지를 위해서 각각의 프로세서 모듈이 그의 주변장치들을 테스트하는 자체 테스트와 다른 모듈들의 주변장치들을 테스트한 결과로부터 고장여부를 판별하는 외부 테스트로 구성되어 있으며, 각각의 모듈들은 서로서로 CM(Common Memory)를 거치지 않고는 통신을 할 수 없기 때문에 각 모듈들의 테스트한 결과를 CM에 모두 기록을 하고 그 결과로 부터 외부 테스트 및 최종 진단을 할 수 있도록 한다.

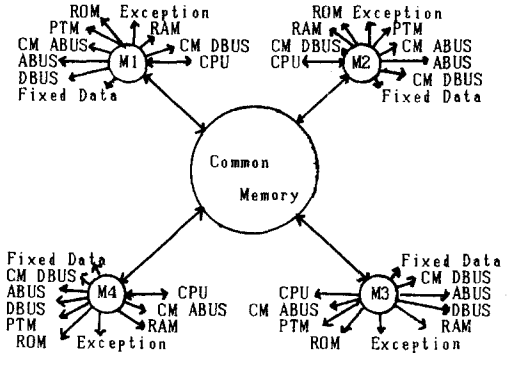

그림 2.1 다중 프로세서의 테스팅 그래프

고장탐지를 할때 가장 주의가 요구되는 것은 제어기의 동작에 영향이 없어야 한다. 즉 테스트를 하면서 제어기에 필요한 데이타를 변화시키거나 주기에 영향을 주어서는 안된다. 대상으로 하는 제어기가 VME버스로 되어 있으면서 각각의 프로세서 모듈이 이 버스에 연결되어져 있으며, 또한 CM도 이 버스에 연결되어 있고 제어기의 입출력은 VME버스 및 별도의 입출력장치에 의하여 이루어 진다.

(2) 자체고장탐지

자체고장탐지는 각 모듈들이 그의 주변장치들을 테스트하는 것으로 앞의 모델에서와 같이 구분하여 각각을 테스트한다. 이때 테스트를 위하여 주변장치들을 다루게 되는데 특히 RAM과 같은 경우 제어기의 동작을 위한 자료가 들어 있을 수 있으므로 항상 먼저 다른 장소에 보관한 후 테스트를 하고, 테스트 도중 인터럽트에 의하여 제어기의 동작을 시작할 수 있기 때문에 데이타가 변화하기 직전에 인터럽트가 걸리지 못하게 한 후 테스트를 하고 데이타를 원상복귀 시킨 후 인터럽트가 동작할 수 있게 하여 항상 올바른 데이타로써 제어기가 동작하게 하여야 한다. 그 각각의 테스트는 다음과 같이 수행되어진다.

CPU는 모듈 내에서 가장 중요한 부분으로서 모든 동작을 관장하고 있으며 주변장치들을 테스트하기 때문에 가장 면밀히 다루어져야 한다. CPU의 구성은 그림 2.2 와 같이 크게 분류할 수 있으며 CPU내부에서 데이타의 보존 및 전달기능을 테스트하는 레지스터 테스트, 그리고 산술 논리동작 테스트, 명령어 해석 동작 테스트, 예외 기능 테스트 등을 하여서 CPU의 동작을 점검한다. 레지스터 테스트는 모든 데이타 및 어드레스 레지스터를 32가지의 데이타로써 그의 보수와 함께 전체를 돌고나서 두 내용을 더하여 "-1"이 되면 다음 산술 논리 동작 테스트를 수행하고 그렇지 못하면 CPU내의 레지스터 고장으로 판별되어서 테스트된 시간 및 모듈번호 그리고 고장번호를 CM에 기록하고 진단에 들어가게 된다. 산술 논리 동작 테스트는 CPU가 수행할 수 있는 모든 산술동작에 특정 데이타를 인가하여 그 결과가 장확한지를 검사한다. 그리고 명령어 해석 동작테스트는 앞의 두 가지 테스트에서 여러가지 명령어의 테스트가 동시에 이루어 졌으므로 여기서는 어드레싱 모드만을 테스트한다. 어드레싱 모드가 여러가지가 있기 때문에 같은 번지를 여러가지 방법으로 지정하여 그 내용이 일치하는 가를 점검한다. 그리고 예외 테스트는 MC68000 프로세서가 가지고 있는 여러가지 예외 동작을 소프트웨어적으로 발생시킨 후 그 예외 동작이 잘 수행되었는 지를 점검한다. 단 인터럽트는 PTM테스트에서 다룬다.

데이타 버스 테스트는 버스선 중에서 단락 또는 개방되어 있는 것이 있는 지를 점검한다. 이는 특정한 번지에 32가지의 데이타를 한 비트의 차이가 나게하여 각각 쓰고나서 다시 읽어 보았을 때 쓴 값과 읽은 값의 차이를

가지고 어느비트가 고장인가를 판단할 수 있다. 또한 주변장치의 연결 부위에서 고장이 발생할 수도 있으므로 주변장치에 맞게 여러곳을 점검하여야 하며 여러가지로 쓰고 읽을 수 없는 주변장치는 별도로 그 주변장치에 맞는 테스트를 하여야 한다. 어드레스 버스 테스트는 위의 데이타 버스 테스트와 아주 유사하며 "FFFF"번지를 기초로 하여 한 비트씩 "0"으로 바꾼 번지 중 한쪽에 먼저 데이타를 쓰고 다른쪽에 다시 데이타를 쓴 후 먼저 쓴 데이타를 읽어보면 그 번지가 잘 지정되었는 지를 알수 있다. 이 또한 주변장치에 따라 여러번 수행을 해야한다.

CSUM(Check Sum)테스트는 기존의 데이타가 운전도중 잘못되어 진 데이타는 없는 지를 알아보기 위해서 사용되며 테스트하고자 하는 모든 데이타를 워드 또는 바이트 단위로 모두 더하여 그 값이 "0"이 되게 하여서, 후에 다시 CSUM하였을 때 "0"이 아니면 변화가 발생되었으며 가장 변화를 적게 탐지할 수 있는 MSB는 4번 이하의 고장을 찾을 수 있으며 LSB쪽으로 갈수록 2배씩의 에러까지도 검사할 수가 있다. 다른 방법으로는 XOR(Exclusive OR)를 써서 모든 데이타를 바이트 또는 워드 단위로 XOR하여 "0"이 아닌가를 판단하면 되나 이 때는 같은 비트의 고장이 두번 일어나면 고장으로 탐지되지 않기 때문에 CSUM방식을 많이 쓰지만, XOR방식 보다 테스트하는데 소요되는 시간이 많은 단점이 있다. 그러나 여기서는 CSUM방식을 쓰기로 한다. 주로 ROM데이타 및 중요한 데이타 불럭에 데이타를 쓸때 한 비트를 별도로 두어 CSUM데이타가 "0"이 되도록 항상 만들어 주면 테스트할 수 있다.

PTM(Programmable Timer)은 CPU에 인터럽트를 걸게하여 매 주기의 제어기 동작을 하도록 인터럽트를 발생시키는 장치로서 이 장치의 테스트를 위하여, 인터럽트 서비스 루틴을 테스트를 위한 것으로 바꾼 후 한 주기 정도의 충분한 시간을 기다린 후 인터럽트 루틴이 수행되어 졌는 지를 검사한다. 이때 테스트를 위한 인터럽트 루틴 내에서는 아주 짧은 시간안에 수행되었음을 표시하고 제어기의 동작을 할 수 있도록 인터럽트가 계속 유지되게 한다. 그리고 이것이 끝나면 이 PTM테스트가 제어기 동작의 시간을 아주 적게 할애하여 인터럽트가 잘 동작하는 가를 판별할 수 있다. 또한 초기 테스트를 별도로 두어 제어기가 처음 동작을 시작하기 전에 인터럽트에 관련된 모든 장치들을 테스트 한 후에 정상적인 동작으로 들어가게 한다.

RAM은 비교적 크기 때문에 CPU의 레지스터 테스트와 같은 방법을 사용하기에는 많은 시간을 요구하기에 RAM자체를 여러 부분으로 나누어서 테스트하며 시간 절약을 위해서 4바이트를 동시에 테스트한다. 먼저 "1010...10"의 형태로 쓴 후 읽어와서 검사하고 또한 "0101...01"형태의 데이타로서 다시 테스트 한다. 이때 stuck-at-1 과 stuck-at-0 를 테스트 할 수 있으며 인접한 데이타와의 단락 개방 또한 테스트가 되어진다. 이때 테스트하는 동안에는 인터럽트가 동작하지 못하게 한 후 테스트하게 되는데 아무리 작은 불럭으로 나누었다고 해도 테스트하는 시간이 제어기에 영향을 줄 수 있기 때문에 4바이트의 테스트가 끝나면 인터럽트가 걸릴 수 있게 하였다가 없게하였을 때 외부에서 인터럽트가 걸린 상태였다면 바로 제어기의 동작을 할 수 있을 것이다.

CM은 모듈내의 RAM과는 차이가 있으며, 여러 모듈이 동시에 읽고 쓰기 때문에 제어기가 사용하는 영역을 테스트하기 위해서는 인터럽트 및 상호 표시를 잘 이용해야 하며 또한 복잡한 알고리즘이 요구되며 제어기의 동작에 상당한 영향을 주게된다. 따라서 여기서는 CM부분을 제어 동작에서만 쓰게 한 후 이 영역을 RAM테스트와 같은 방법으로 테스트한다. 이때 제어동작에 사용되어 지는 영역을 여러 부분으로 나눈후 그 사이사이를 CM테스트 하면 더 효과적인 고장탐지가 된다.

대상을 VME버스와 CPU모듈들로 구성하였기에 CPU모듈은 이미 몇 가지 고장탐지 기능이 부가되어 있는데 이는 예외 처리가 거의이 대부분이 고장 또는 소프트웨어의 오류로서 발생된다.이는 CPU자체가 고장으로 하여주기 때문에 이 고장을 처리하는 루틴만 첨가하면 된다. 실제로 동작에 있어서 상당부분의 에러가 이 예외 처리로 검출되어 진다.

(3) 외부 테스트

외부 테스트는 각 모듈들의 동작 상태를 테스트하는 것으로 각 모듈들은 모든 테스트 결과를 CM에 적어놓고 다른 모듈들의 테스트 결과를 가지고 그 모듈이 전체적으로 정상적인 동작을 하는 지를 테스트한다. 외부 테스트는 두가지로 구성되며 시간초과 고장과 전갈 형태 고장이 있다.

시간 초과는 각 모듈들이 내부에 다른 모듈들의 현재 시간과 그 시간으로부터의 경과를 데이타로 가지고 있으면서 다른 모듈의 동작하지 않은 시간을 검사하여 일정 시간을 초과하면 시간초과 고장으로 처리한다. 각 모듈은 자체 테스트를 수행하며 매 테스트가 끝나면 그 결과를 CM에 기록하는데 이때 자신의 시간을 같이 기록하여 주게 된다. 따라서 한 모듈이 다른 모듈의 시간을 읽어서 그 값의 변화가 어느 일정시간 이상 없으면 고장이다. 이는 모듈이 전혀 동작을 하지못할 때도 발생하고, 또한 한 주기 내에서 남는 시간이 없을때 자체 테스트를 거의 수행할 수 없기 때문이다.

전갈 형태고장은 CM에 기록된 테스트 결과가 주어진 형태에 맞게 구성되어져 있는 지를 검사하게 된다. 즉 데이타 형태를 여분어 많게 특수한 몇 가지만을 존재하게 하여 그 나머지는 전갈 형태 고장으로 한다. 이 전갈 형태 고장은 재 테스트에 의하여 바로 고쳐지게 된다.

3. 고장 진단

CM에 기록된 테스트의 결과는 각 모듈별로 되어

있으며 또한 각각의 테스트는 고유한 테스트 번호를 가지게 된다. 모든 테스트를 차례로 모두 테스트 한 후 다음을 반복하는 것이 고장의 최대 잠복 시간이 짧아서 좋으나 고장이 자주 발생하는 것은 잠복 시간이 길면 문제가 있으므로 중요한 부분은 매번 테스트하며 ROM데이타와 같은 거의 고장이 나지 않는 것은 어느정도 시간이 경과 한 후 한번씩 테스트 한다. 그림 3.1과 같이 하였을 경우 한번 수행 하는데 0.1[sec] 정도 소요 되었으며 RAM 및 ROM 그리고 프로그램 CSUM테스트에서 많은 시간이 소요되었다.

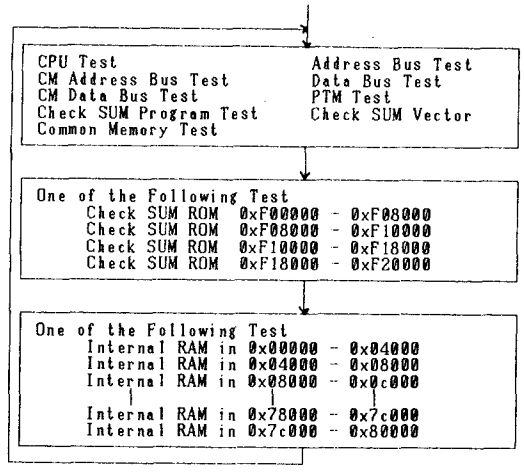

그림 3.1 자체 테스트의 흐름도

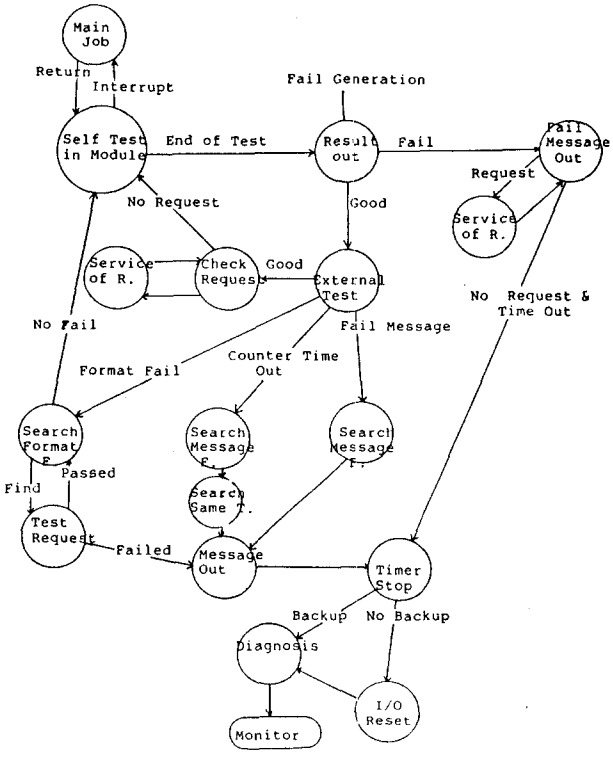

그림 3.2 전체 상태도

그림 3.2 와 같이 각 사태 변화가 발생하며, 매 자체 테스트가 끝나면 그 결과를 CM에 기록하고 고장이 탐지되면 재 테스트를 하여 정상이면 외부 테스트를 수행하고, 재 테스트에서 다시 고장이면 CM에 고장이라고 쓰고 다시 외부에서 테스트 요구가 있는 지를 확인 한다. 이는 만약 고장이면 CM에 쓰는 결과가 정확하지 못할 경우도 있기 때문에 테스트 요구에 의해서 확인을 하게 된다. 또한 외부테스트 에서 전갈 형태 및 시간 초과를 검사한 후 다른 모듈의 고장이 있는지를 검사하여 분명한 고장이 있으면 이 고장이 전체 동작에 크게 영향을 미치는 것인지를 판별하여 계속 운전할 수 없으면 제어기를 멈추게 하여야 하므로 인터럽트가 동작하지 못하게 한 후 적절한 동작에 의해서 위급한 상태가 발생하지 않게 처리한후 진단에 들어가게 된다. 이때는 시간적 여유가 충분하기 때문에 모든 데이타를 검색 할 수 있다. 진단을 위해서는 3개 이상의 모듈이 있어야 하며 모듈수가 많을수록 진단은 정확하게 할 수 있다.모든 모듈은 자기 모듈의 테스트 결과를 검색하여 2가지 종류 이상의 고장이 찾아지면 진단을 포기하게 되며, 이는 자기 모듈이 고장이라고 하여 다른 모듈이 이 모듈을 고장으로 처리하게 될 것이다.

한 모듈이 자체 테스트 또는 외부 테스트에서 고장이 검출되면 그 모듈 및 테스트 번호를 모든 모듈들에게 알리게 되는데 이 전갈에 의하여 다른 모든 모듈 또한 외부 테스트에서 고장이 검출되기에 한 모듈이 고장을 검출하면 모든 모듈이 그 고장을 알수 있으며 각 모듈은 또다시 모든 테스트 결과를 검색해 본 후 고장으로 된 테스트를 찾아서 다시 모든 모듈에게 알리게 된다. 그리고 가장 신뢰도가 높은 모듈은 다른 모듈을 검사하여 고장을 알리지 않은 모듈이 있으면 동작을 하지않은 모듈로서 별도로 처리하고, 그 나머지 모듈에서 서로 같은 정보를 지닌 것을 검사한다. 이때 같은 정보가 가장 많은 것의 모듈 및 테스트 번호로부터 어떤 모듈의 어떤 테스트가 고장인지를 안다. 만약 동수인 경우는 미리 선정된 순서에 따라 신뢰도가 높은 순서에 따라 신뢰도가 높은 모듈의 결과를 따른다.

이러한 방법은 고장이 탐지되면 바로 처리가 되어 지기 때문에 2개 이상의 고장이 발생할 가능성은 아주 희박하며 발생하였다면 주로 이 모듈에 고장이 발생하여 오동작에 의해서 다른 고장을 발생시킨 경우 이므로 이는 전체 테스트 결과에서 2개 이상의 고장이면 진단을 포기하게 하여 피한다.

4. 고장처리

위와 같은 동작 전체를 다루기 위하여 여러가지 모드가 필요되어 진다. 즉 항시 모드를 확인하여 그 모드에 의하여 동작이 되어진다. 처음 운전을 시작하기 전에 테스트 모드에 의해서 세부적으로 테스트를 수행하여 그 결과를

알려주게 하여 전체 시스템이 정상적인 동작을 할 수 있는지를 판별하고 고장이 있으면 정밀히 알아보기 위하여 모니터 모드에 의해서 여러가지를 제공 받으며 시스템 전체가 정상이면 운전 모드에 의해서 테스트를 수행하면서 인터럽트에 의한 제어 동작이 동시에 수행되게 된다. 고장이 발견되면 출력을 그대로 유지 할 것인지 아니면 초기화 시킬 것인 지를 판별하여 동작을 시킨 후 고장진단에 의하여 어느 고장인지를 알려주고 나서 다시 명령어 모드로 와서 사용자가 모드를 선택하여 다음 동작을 할 수 있게 한다. 또한 고장이 아닐때라도 명령어는 읽어들일 수 있도록 외부 테스트를 할 때마다 항상 명령어를 점검하여 다음 동작을 결정한다.

5. 결과 및 결론

실험을 위해서 VME시스템에 CPU모듈로서 MVME110-1을 2개와 MVME-117 1 개 그리고 CM과 시스템 제어기로서 MVME050을 사용하였다. 이 3개의 모듈에서 각각이 고유한 제어 동작을 수행하면서 자체 및 외부 테스트를 수행하게 하였다. 그리고 또한 SYS121이 전체를 관장 하도록 명령어를 내리게 하였다.

실험을 위하여 몇가지 고장을 발생시켜 보았는데 1가지씩의 고장을 발생시켰을 때는 정확히 찾아 내었다. 그러나 동시에 5가지의 전갈 형태 고장을 발생시켰을 때는 모든 결과를 재 테스트에 의하여 회복 시킬수 있었으나 그 이상은 회복도중 시간초과 고장으로 되었다. 또한 여러개의 고장을 테스트의 결과를 바꾸어서 가하였을 때는 모두 고장이라고 나타낸 후 그 중 신뢰도가 가장 높은 모듈이 선택한 고장을 추측하여 나타내었다. 또한 CM의 IC한개를 뽑았을때 CM고장, 또는 CM ABUS고장, CM DBUS고장의 3가지 중에 한 가지로 나왔다.

본 연구에서는 다중프로세서의 신뢰도를 개선하기 위하여 디지탈 제어의 특징을 이용, 주된 제어기의 동작을 모두 인터럽트 서비스 루틴 으로 처리하고 그 밖에 모든 시간을 고장 탐지에 할애하였으며 하드웨어 및 일부 소프트웨어의 고장을 탐지하게 하였으며 진단에 의하여 고장 부위를 찾았다. 여분의 시스템이 있으면 고장 탐지에 의하여 재 구성을 하여 모든 고장에 대하여 운전을 계속할 수 있는 FT시스템을 구성할 수 있을 것이다.

앞으로 더 연구가 필요한 부분으로서는 고장 탐지에서 시스템에 적합한 테스트를 더 첨가하고, 또한 여분의 CM및 버스를 두어서 완전한 FT시스템을 구성할 수 있으며, 실시간으로 하여 고장발생 시간 또한 명확히 판별할 수 있을 것이다.

그러나 다중프로세서를 이용한 기존 제어기에 쉽게 적용하기 위하여 FT시스템으로 하기는 어려우며 , 단지 여러가지 고장 탐지 기능 및 각 시스템에 적합한 고장 대처 기능을 첨가하여 안정되고 또한 고장시에 안전을 보장하는 시스템으로 만들 수 있을 것이다.

* 참고문헌

1] David A. Rennels, "Fault-Tolerant Computing -- Concept and Examples," IEEE Trans. on Computer c-33, pp. 1116-1129, 1984.

2] Barry W. Johnson, "Fault-Tolerant Microprocessor-based Systems," IEEE Micro December, pp. 6-21, 1984.

3] Ormi Serlin, "Fault-Tolerant System in Commercial Application," IEEE Computer August, pp. 19-30, 1984.

4] 이 헌, "Fault-Tolerant Computing System," 전자교환기술, 제 1 권, 제 1호, pp. 46-61, 1985.

5] Jon G. Kuhl, S. M. Reddy, "Fault Tolerant Consideration in Large, Multiple-processor Systems," IEEE Computer, March, pp. 56-67, 1986.

6] E. Schrodi, "Fault-Tolerant and Fail-Safe Microcomputer Systems," IFAC, pp. 183-188, 1984.

7] L. Y. Lu, "Fault-Tolerant Programs in Process Control Systems," IECON, pp. 877-882, 1984.

8] Patric P. Fasang, "A Fault Detection and Isolation Technique for Microcomputers," IEEE Test Conference, pp. 214-219, 1982.

9] Dhanajay Brahme, Jacob A. Abraham, "Functional Testing of Microprocessors," IEEE Trans. on Computer, c-33, pp. 475-485, 1984.

10] S. H. Hosseini, Jon G. Kuhl Sudhakar M. Reddy, "A Diagnosis Algorithm for Distributed computing Systems with Dynamic Failure and Repair," IEEE Trans. on Computers, c-33, pp. 223-233, 1984.

11] 조 현 용, "화력발전소를 대상으로 한 백업 콘트롤 시스템의 연구," KAIST, 석사학위 논문, 1986.

12] 신 영 달, "Boiler Backup Control을 위한 Multiprocessor 방식에서의 신뢰도 개선에 관한 연구," KAIST, 석사학위 논문, 1987.