내장형 시스템의 상호작용 오류 감지를 위한 테스트 데이타 선정 기법

(Test Data Selection Technique to Detect Interaction Faults in Embedded System)

성 아 영 † 최 병 주 ††
(Ahyoung Sung) (Byoungju Choi)

요 약 하드웨어와 소프트웨어가 조합된 내장형 시스템이 복잡해지면서, 내장형 시스템에 탑재되는 내장형 소프트웨어 테스트가 중요하게 인식되고 있다. 특히, 원자력 발전소와 같이 안전 등급이 높은 시스템에 들어가는 소프트웨어 테스트는 필수적이다. 내장형 시스템 테스트의 경우 하드웨어와 소프트웨어의 상호작용에 의해 발생하는 오류를 발견하기 위한 효과적인 테스트 기법이 필요하다. 본 논문에서는, 하드웨어와 소프트웨어 사이의 상호작용에 의해 생성되는 오류를 발견하기 위하여, 오류 삽입 기법을 이용한 테스트 데이타 선정 기법을 제안하고, 이 기법을 Digital Plant Protection System에 적용하였으며, 실험을 통해 제안한 기법의 우수성을 분석한다.

키워드 : 내장형 시스템, 하드웨어 오류, 소프트웨어 오류, 소프트웨어 테스트, DPPS

Abstract As an Embedded system combining hardware and software gets more complicated, the importance of the embedded software test increases. Especially, it is mandatory to test the embedded software in the system which has high safety level. In embedded system, it is necessary to develop a test technique to detect faults in interaction between hardware and software. In this paper, we propose a test data selection technique using a fault injection technique for the faults in interaction between hardware and software in embedded system and we apply our technique to the Digital Plant Protection System and analyze effectiveness of the proposed technique through experiments.

Key words : Embedded System, Hardware Fault, Software Fault, Software Test, DPPS

1. 서 론

하드웨어와 소프트웨어 조합된 내장형 시스템[1,2]은 안전에 대한 치명도가 높은 의료 기기, 발전소 시스템 및 가전 제품과 같이 실생활에서 사용되고 있는 대부분의 시스템이 내장형 시스템이다. 최근 내장형 시스템의 기능이 복잡해지면서 내장형 시스템에 대한 테스트의 중요성이 인식되고 있으나, 아직은 내장형 시스템 테스트에 대한 연구가 미흡한 실정이다.

내장형 시스템은 하드웨어와 소프트웨어가 조합된 시스템으로, 하드웨어와 소프트웨어 사이의 상호작용에서 예상하지 못한 상황이 발생한다[3]. 이러한 상황은 하드웨어와 소프트웨어의 상호작용에 의한 오류로 인해 발생하기 때문에, 하드웨어와 소프트웨어의 상호작용에 의한 오류들을 발견할 수 있는 테스트 데이타 선정 기법은 필수적이다. 이미 만들어진 내장형 시스템에 오류가 생기면 수정하는 것이 쉽지 않고, 이를 테스트를 하게 되면 비용도 많이 든다. 특히 발전소 시스템이나 의료기기처럼 안전에 대한 치명도가 높아 실제 시스템에 테스트를 하기 힘든 시스템일수록, 하드웨어와 소프트웨어의 상호작용에 의한 오류를 효과적으로 발견할 수 있는 테스트 데이타 선정 기법이 필요하다.

본 논문에서는 내장형 시스템의 하드웨어와 소프트웨어의 상호작용을 테스트하기 위하여 오류 삽입 기법[4-15]을 이용한 테스트 데이타 선정 기법을 제안한다. 오류 삽입 기법은 하드웨어나 소프트웨어의 특정 위치에 오류를 삽입하여 대상 시스템이 어떻게 동작하는지를 관찰하는 기법이다. 내장형 시스템에서는 하드웨어와

· 본 연구는 "원자력 연구개발 중장기사업인 원전계측제어 사업단"에 의해 지원되었음
† 학생회원 : 이화여대 컴퓨터학과
aysung@ewha.ac.kr
†† 종신회원 : 이화여대 컴퓨터학과 교수
bjchoi@ewha.ac.kr
논문접수 : 2002년 12월 13일
심사완료 : 2003년 8월 21일

소프트웨어의 상호작용에 의한 오류 때문에 예상하지 못한 상황이 발생하며, 내장형 시스템의 기능이 복잡해짐에 따라 이러한 오류를 발견하는 것이 점점 어려워지기 때문에, 하드웨어와 소프트웨어의 상호작용에 의한 오류를 발견하기 위하여 대상 시스템에 인위적으로 오류를 삽입하여 시스템의 행동을 관찰하는 오류 삽입 기법은 매우 적합하다.

제안하는 테스트 데이타 선정 기법은 요구 명세 단계로부터 내장형 시스템의 동작을 소프트웨어 프로그램으로 시뮬레이트하고, 하드웨어 오류를 소프트웨어 오류화 하여 시뮬레이트 된 프로그램이 삽입함으로써, 하드웨어와 소프트웨어의 상호작용에 의한 오류를 발견할 가능성이 높은 테스트 데이타를 선정한다.

본 논문의 구성은 다음과 같다. 2절에서는 관련 연구를 기술하고, 3절에서는 원자력 보호계통 시스템인 DPPS(Digital Plant Protection System)에 기술하고, 4절에서는 본 논문에서 제안한 기법을 DPPS에 적용한 예를 기술하고, 5절에서는 실험을 통해 제안한 기법의 우수성을 분석하고 6절에서는 결론 및 향후 과제에 대하여 기술한다.

2. 관련 연구

내장형 시스템은 하드웨어와 소프트웨어가 조합된 시스템으로, 하드웨어와 소프트웨어의 상호작용에 의해 발생하는 오류를 발견하는 것이 중요하다.

본 논문에서는 내장형 시스템의 상호작용 오류 발견을 위하여, 다음과 같이 오류 삽입 기법을 활용한다. 대상 시스템에 존재하는 하드웨어 오류를 파악하고, 이 하드웨어 오류를 소프트웨어 오류로 전환한 후, 대상 시스템의 동작을 시뮬레이트 한 프로그램에 소프트웨어 오류로 전환된 하드웨어 오류를 삽입하는 방식의 오류 삽입 기법을 활용한다. 즉, 내장형 시스템의 상호작용 오류 발견을 위하여, 하드웨어 오류 삽입 기법에서 사용되는 하드웨어 오류를 파악하고, 파악한 하드웨어 오류를 소프트웨어 오류로 전환한 후 대상 프로그램에 삽입함으로써, 대상 프로그램의 동작을 관찰하고 테스트 데이타를 선정하는 방식의 소프트웨어 오류 삽입 기법을 활용한다.

하드웨어 오류 삽입 기법[4-15]에 사용되는 하드웨어 오류들을 파악함으로써, 대상 시스템에 존재 할 수 있는 하드웨어 오류를 파악하였다. 하드웨어 오류 삽입 기법은 이미 만들어진 하드웨어가 제대로 동작하는지를 보기 위해 대상 하드웨어에 인위적으로 오류를 삽입함으로써 대상 하드웨어의 행동을 관찰하는 기법이다. 이 때 사용되는 하드웨어 오류는 하드웨어에 물리적인 접촉에 의해 오류를 삽입하는 경우와 대상 하드웨어의 물리적인 접촉이 아닌 방법에 의해 오류를 삽입하는 경우로 구분한다. 물리적인 접촉에 의해 삽입하는 오류는 하드웨어 핀(Pin) 레벨의 프로브(Probe)나 소켓(Socket)을 사용하여 오류를 대상 하드웨어에 주입하며, 이 경우에 해당하는 하드웨어 오류의 예는 Open 오류, Bridging 오류, Bit-flip 오류, Stuck-at 오류, Power surge 오류등이 있다. 물리적 접촉이 없는 경우에는 대상 하드웨어와 직접적으로 물리적인 접촉은 없지만 중이온(Heavy ion)이나 전자기의 간섭(Electromagnetic interference)을 사용하여 대상 하드웨어에 영향을 주며, 이 경우에 속하는 하드웨어 오류의 예는 Spurious current 오류이다.

소프트웨어 오류 삽입 기법[22,23]은 다음과 같은 두 가지 목적에서 사용된다. 첫번 째 목적은 하드웨어 오류 삽입 기법과 마찬가지로 대상 소프트웨어에 인위적으로 오류를 삽입함으로써 대상 소프트웨어의 동작을 관찰하기 위함이다. 두번 째 목적은 대상 프로그램에 오류를 삽입함으로써,원래의 프로그램과 오류가 삽입 된 프로그램의 결과 값을 차별화 하는 입력 데이타를 테스트 데이타로 선정 하는 방식을 통해, 대상 프로그램에 존재하는 오류를 발견할 수 있는 테스트 데이타를 선정하기 위함이다.

3. Digital Plant Protection System

원자력 발전소의 보호 계통 시스템은 원전에 문제가 발생했을 때, 원전 시스템을 안전하게 정지시키기 위한 시스템이다. 하드웨어와 소프트웨어가 조합된 DPPS는 원전 보호계통 시스템으로 4개의 독립적인 채널로 구성되어져 있다. 각 채널은 PPC(Plant Protection Calculator)와 CPC(Core Protection Calculator)로 구성되어져 있다.

각 PPC는 그림 1과 같이 바이스테이블 프로세서(Bistable Processor), 동시논리 프로세서(Coincidence Processor), 트립개시 프로세서(Initiation Processor)로 구성되어져 있다. 바이스테이블 프로세서는 원전의 상태를 모니터링 하는 센서로부터 9개의 아날로그 신호와 CPC로부터 2개의 디지털 신호를 입력받아, 이미 설정되어 있는 설정치와 비교하여 트립(Trip) 상태를 결정한다. 입력 값이 정해진 설정치보다 높아지거나 낮아지게 되면 동시논리 프로세서로 트립 신호를 보낸다. 이때, 바이스테이블 프로세서가 입력을 받을 때, 운전 우회신호를 확인하여 우회 신호가 걸려 있는 경우에는 계속 입력 신호를 받아들인다. 우회 신호는 원자로 제어에 관련하여 관리자가 인위적으로 값들을 조작하는 경우나, 시스템을 처음 켰을 경우에 세팅된다. 동시논리 프로세서는 바이스테이블 프로세서의 출력을 확인하여 4개의 채널 중 2개 이상의 채널에서 트립 신호를 가질 때 트립개시

프로세서로 트립개시 신호를 보낸다. 트립개시 프로세서는 트립개시 신호를 받으면 TCB 신호(제어봉 낙하신호)를 보내서 원자로를 중지하라는 명령을 내린다. 원자로의 정지는 제어봉 구동장치의 코일을 차단함으로써 제어봉을 노심 내로 자중 낙하함으로써 달성된다.

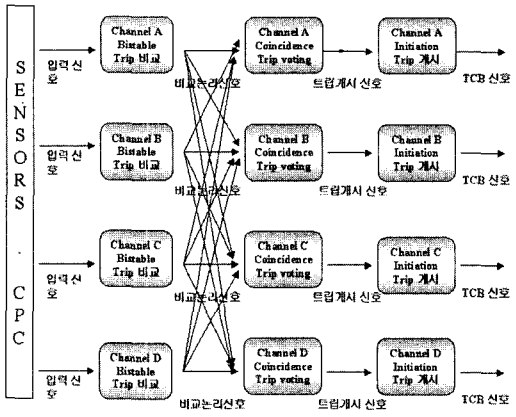

그림 1 DPPS의 구조와 동작

4. DPPS의 테스트 데이타 선정

DPPS는 시스템에 문제가 발생 할 때, 시스템을 안전하게 정지시키기 위한 보호계통 시스템이다. 시스템이 이상 상태에 있을 때 시스템을 정지하는 것을 트립(Trip)이라고 한다. DPPS 내장형 시스템 테스트는 트립이 제대로 일어나는지 검사하는 트립 시험을 의미한다. DPPS는 바이스테이블 프로세서, 동시논리 프로세서, 트립개시 프로세서와 같은 하드웨어와 소프트웨어로 표현된 각 프로세서의 기능이 조합된 내장형 시스템이기 때문에, 하드웨어와 해당 하드웨어의 기능을 표현한 소프트웨어의 상호작용에서 발생하는 오류가 중요하며, 이러한 오류를 발견하기 위하여, 본 논문에서 제안한 테스트 데이타 선정 기법과 기법의 적용 예를 4장에서 기술한다.

4.1 테스트 데이타 선정 기법

내장형 시스템 테스트를 위한 테스트 데이타 선정 단계는 크게 두 단계로 구성한다. 첫 번째 단계는 내장형 시스템을 분석하는 단계로 하드웨어 모듈 및 소프트웨어 모듈을 분석하여, 하드웨어 모듈과 그 모듈의 입, 출력 정보를 표현하기 위한 HBD(Hardware Block Diagram)를 작성하고, HBD를 구성하는 하드웨어 모듈에 기능을 나타내는 소프트웨어 모듈을 표현한 HSBD (Hardware Software Block Diagram)을 작성한다. 두 번째 단계는 내장형 시스템 테스트를 위한 테스트 데이타 선정 단계로 HSBD의 동작을 시뮬레이트 하는 내장형 소프트웨어 프로그램 S를 작성하고, 하드웨어 오류 f들을 소프트웨어 오류로 전환하여 프로그램 S에 내장한 프로그램 P들을 활용하여 테스트 데이타를 생성한다.

4.2절에서 제안한 기법의 각 단계에 대한 구체적인 설명과, DPPS의 적용한 예를 기술한다.

4.2 DPPS에의 적용 예

4.2.1 DPPS 분석

내장형 시스템의 분석단계는 그림 2와 같이 내장형 시스템의 요구사항을 분석하여 하드웨어 블록 다이어그램(Hardware Block Diagram: HBD)을 작성하는 HBD 작성 단계와 HBD에 소프트에어 모듈을 표현하는 하드웨어 소프트웨어 블록 다이어그램(Hardware Software Block Diagram: HSBD)을 작성하는 단계로 구분한다.

HBD는 대상 시스템의 요구 명세로부터 하드웨어 모듈과 모듈의 입, 출력 시그널(Signal)을 표현한 다이어그램으로 블록으로 표현하는 하드웨어 모듈과 선으로 표현하는 입, 출력 시그널로 구성하며, HSBD는 HBD를 구성하는 각 하드웨어 블록의 기능을 표현하기 위하여, 각 하드웨어 블록의 기능을 소프트웨어 모듈로 표현한 다이어그램으로 그림 3과 같이 소프트웨어 모듈을 타원으로 표현하고, 소프트웨어 모듈 사이에 발생하는 호출 관계는 방향성이 있는 화살표로 표현한다.

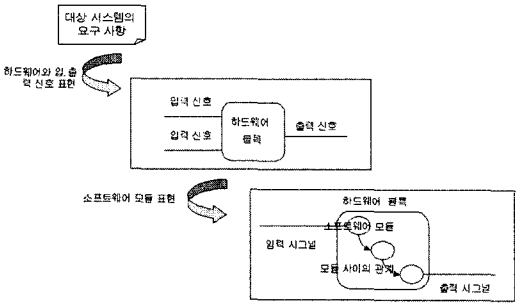

그림 2 내장형 시스템 분석 단계

DPPS의 요구 사항을 분석하여 채널 A에 대한 DPPS의 HSBD를 작성하면 그림 3과 같다. 그림 3에서는 DPPS에 들어가는 하드웨어는 사각형으로 표현하였으며, 각 하드웨어의 동작을 나타내는 소프트웨어 모듈은 타원으로 표현하였으며, 하드웨어 사이의 라인은 하드웨어 사이의 입력과 출력에 해당하는 신호를 의미한다.

그림 3에서 Sensor와 CPC로부터 입력받은 9개의 아날로그 시그널 AI와 2개의 디지털 시그널 DI를 나타내는 입력 변수의 값이 입력이며, 이 입력 변수의 값은 바이스테이블 프로세서인 A_BP로 가서 A_BP가 트립 신호인지 아닌지를 분석한다. 동시 논리 프로세서인 A_CP는 각 채널의 바이스테이블 프로세서의 출력 신호

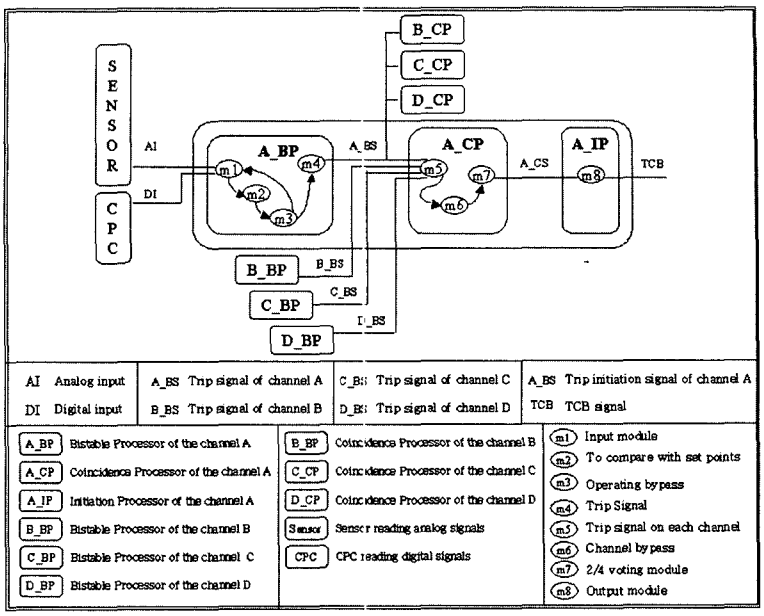

그림 3 채널 A에 대한 HSBD

즉 A_BS, B_BS, C_BS, D_DS를 입력으로 받아 채널의 상태를 점검하여 2개 이상의 채널에서 트립 신호가 발생되면, 트립개시 프로세서인 A_IP로 트립개시 신호인 A_CS를 보내주며, A_IP는 이 신호를 받아 TCB 신호로 내보낸다. 이때 출력 변수의 값으로 표현되는 TCB신호는 DPPS의 출력 신호로 입력받은 11개의 신호가 대상 시스템에서 트립을 일으키는지를 나타낸다.

4.2.2 DPPS의 테스트 데이타 선정

(1) 프로그램 S 작성

HSBD의 동작을 시뮬레이트 하는 소프트웨어 프로그램 S를 작성한다. S를 C 프로그래밍 언어로 작성한다면, 그림 4와 같이 HSBD의 각 하드웨어 모듈 Mi에 속하는 소프트웨어 모듈 mij를 함수로 구성하여, main()에서 연결하도록 한다.

본 논문에서는 그림 3의 HSBD로부터 DPPS의 동작

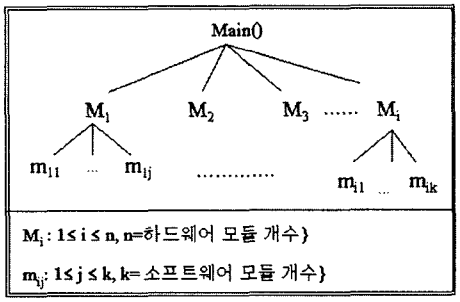

그림 4 S의 구성

을 시뮬레이트 하는 프로그램 S를 C 프로그래밍 언어로 작성한다. DPPS의 각 채널별 프로그램 S는 동일하며, 채널 A의 경우를 예를 들면, 그림 3의 채널 A의 하드웨어 모듈 A_BP, A_CP, A_IP의 각 소프트웨어 모듈은 프로그램 S에서 총 8개의 function으로 된다.

(2) 프로그램 Pf 작성

내장형 시스템의 하드웨어와 소프트웨어의 상호작용에 의한 오류 발견을 위하여 하드웨어 오류를 소프트웨어 오류로 변환하여 프로그램 S에 삽입하여 프로그램 Pf를 생성하도록 한다. 이를 위하여 먼저 하드웨어 오류의 종류를 분류할 필요가 있다. 내장형 시스템에 존재하는 하드웨어 오류를 기존의 연구[4-15]를 바탕으로 분류하면 Open 오류, Bridging 오류, Bit-flip 오류, Stuck-at 오류, Spurious current 오류, Power surge 오류로 분류되며, Stuck-at 오류의 경우 Stuck-at 0과 Stuck-at 1로 세분화한다. 표 1에서는 분류된 각 하드웨어 오류와 각 하드웨어 오류에 대한 설명과, 하드웨어 오류가 미치는 영향을 나타낸다.

하드웨어 오류에 의해 시스템이 영향을 받는다는 것은 대상 시스템의 동작을 시뮬레이트 한 소프트웨어 프로그램 S의 어딘가에 오류가 존재한다고 해석할 수 있다. 내장형 시스템에 존재하는 하드웨어 오류를 소프트웨어 오류로 바꾸기 위해서는 하드웨어 오류가 프로그램 S의 어느 부분에 영향을 미치는지를 파악하여 오류를 인위적으로 삽입하는 위치를 결정하는 과정과, 그 부분을 어떻게 소프트웨어 오류로 바꾸는 것에 대한 두

표 1 하드웨어 오류 분류

종류		하드웨어 오류 설명 오류의 영향
Open		연결 불량 때문에 라인이나 블록에 저항이 발생한다. 라인이나 블록에 해당하는 값이 다른 값으로 바뀐다.
Bridging		2개 이상의 라인이 교차할 때, 교차하는 라인의 값이 바뀐다. 교차하는 라인의 값이 바뀐다.
Bit-flip		비트가 바뀐다. 바뀐 비트에 해당하는 변수 값을 바꾼다.
Stuck-at	Stuck-at 0	결과 값이 0으로 고정된다. 결과에 해당하는 변수 값이 항상 0으로 나온다.
	Stuck-at 1	결과 값이 1로 고정된다. 결과에 해당하는 변수 값이 항상 1로 나온다.
Spurious current		중이온(Heavy ion)에 노출된다. 하드웨어에 흐르는 전류가 변하여, 특정 위치의 값이 변하는 것이 아니라 시스템 전반에 걸쳐 이상 값이 발생한다.
Power surge		일정하지 않은 전원이 공급된다. 하드웨어에 흐르는 전류가 변하여, 특정 위치의 값이 변하는 것이 아니라 시스템 전반에 걸쳐 이상 값이 발생한다.

개의 과정이 필요하다.

1) 하드웨어 오류를 삽입하는 위치(Fault Injection Target) 결정

하드웨어 오류가 소프트웨어 프로그램 S에 영향을 미치는 부분은 하드웨어 오류에 영향을 받는 S의 마지막 위치의 변수라고 정의한다. 왜냐하면 S에 사용된 변수를 통해 하드웨어의 작동이 운영되며, 하드웨어 오류가 있다면 S에서의 해당 변수의 마지막 위치에서 예상하지 않은 결과를 낼 확률이 높기 때문이다.

본 논문에서는 내장형 시스템의 하드웨어와 소프트웨어의 상호작용 부분에 오류를 임의로 삽입하는 방식에 따라 테스트 데이타를 선정하는 것을 제안하며, 이때 하드웨어 오류에 영향을 받는 S의 마지막 위치의 변수가 오류 삽입 위치(FIT)가 된다. 예를 들면, 표 1에서 보는 바와 같이 Open 오류, Bridging 오류, Bit-flip 오류, Stuck-at 0 오류, Stuck-at 1 오류의 경우 오류가 어느 위치에 어떻게 영향을 미치는지 알 수 있기 때문에, 이러한 오류들이 S에 미치는 영향을 예상할 수 있다. 따라서 Open 오류, Bridging 오류, Bit-flip 오류, Stuck-at 0 오류, Stuck-at 1 오류에 의해 영향을 받는 S의 마지막 위치 변수를 FIT로 선정한다. Spurious current 오류, Power surge 오류의 경우에는 전체 시스템에 영향을 미치는 오류이기 때문에, S의 최종 출력 변수를 FIT로 선정한다.

표 2 DPPS 채널 A의 FIT의 일부

하드웨어 오류	하드웨어 오류 위치	FIT
	설명	
Stuck-at 0	A_BS, A_BP	*f*1:analog[i].trip_set *f*2:digital[i].trip_set *f*3:analog[i].trip_bistable[A] *f*4:digital[i].trip_bistable[A]
	A_BS, A_BP에 Stuck-at 0가 있을 경우, S에서 A_BS와 A_BP에 해당하는 프로그램의 마지막 위치의 변수 *f*1-*f*4가 FIT가 된다.	
	A_CS, A_CP	*f*5:analog[i].trip_coincidence *f*6:digital[i].trip_coincidence
	A_CS, A_CP에 Stuck-at 0가 있을 경우, S에서 A_CS와 A_CP에 해당하는 프로그램의 마지막 위치의 변수 *f*5, *f*6이 FIT가 된다.	
	TCB, A_IP	*f*7:analog[i].trip_initiation *f*8:digital[i].trip_initiation
	TCB, A_IP에 Stuck-at 0가 있을 경우, S에서 TCB와 A_IP에 해당하는 프로그램의 마지막 위치의 변수 *f*7, *f*8이 FIT가 된다.	
...	...	...
Spurious current	A_IP	*f*47:analog[i].trip_initiation *f*48:digital[i].trip_initiation
	A_IP에 Spurious current faults가 있을 경우, S에서 A_IP에 해당하는 프로그램의 마지막 위치의 변수 *f*47, *f*48이 FIT가 된다.	

표 2는 표 1의 하드웨어 오류 분류 체계에 따라 DPPS 채널 A에서 발생 가능한 하드웨어 오류를 식별한 것으로, Open 오류, Bridging 오류, Bit-flip 오류, Stuck-at 0 오류, Stuck-at 1 오류, Spurious current 오류, Power surge가 있으며, 각 오류가 발생할 하드웨어의 위치와, 이 오류들이 소프트웨어로 삽입될 때, 삽입되는 오류 삽입 위치를 파악한 표로, Stuck-at 0와 Spurious current 오류 대한 예를 보여주며, DPPS 채널 A의 경우 총 48개의 FIT가 파악되었다.

2) 하드웨어 오류를 소프트웨어 오류로 변환하여 프로그램 P*f* 작성

표 1에서 파악한 하드웨어 오류들은 코드 패치(Code Patch)를 통해 소프트웨어 오류로 바꾼다. 코드 패치란 하드웨어 오류에 영향을 받는 S의 마지막 위치의 변수인 FIT의 변수 값을 바꾸기 위하여 프로그램 코드를 추가하는 것이다. 하드웨어 오류를 소프트웨어 오류로 변환하기 위한 코드 패치 방법은 다음과 같다.

< 코트 패치 방법 >

① FIT의 변수 값이 디지털 신호 값 0/1을 가지는 경우에는 변수 값을 부정함으로써 변수 값을 바꾼다.

② FIT의 변수 값이 아날로그 신호와 같이 여러 가지 값을 가지는 경우에는 변수 값을 rand()%n을 통해 임의의 값으로 바꾼다.

프로그램 S에 표 2에서 파악한 48개의 FIT를 위에서 설명한 코드 패치 방법을 이용하여 소프트웨어 오류로 변환하여 프로그램 48개의 P_f들을 생성한다. 그림 5는 HSBD의 동작을 시뮬레이트 한 S의 일부와, Spurious current 오류를 S에 삽입한 생성한 P_{f48}의 일부를 나타내며, $f48$에 해당하는"digital[i].trip_initiation"은 0 또는 1의 값을 가지는 신호로, 그림 5의 P_{f48}의 이탤릭체로 표시한 "digital[i].trip_initiation!=digital[i].trip_initiation" 처럼 코드 패치 한다.

따라서 DPPS 채널 A의 경우, 48개의 FIT 부분에 코드 패치를 이용하여 소프트웨어 오류로 변환함으로써, 48개의 서로 다른 프로그램 $P_{f1}...P_{f48}$을 생성한다. 그림 5에 P_{f48}의 일부분을 나타내었듯이 프로그램 S와 표 2의 48번째 f_{48} 오류가 삽입된 프로그램 P_{f48}은 FIT 부분에 코드 패치된 일부분을 제외하면 나머지 코드는 프로그램 S와 동일하다.

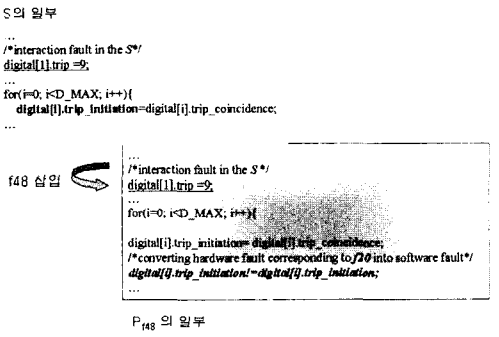

그림 5 프로그램 S와 프로그램 P_{f48} 생성 예

(3) 테스트 데이타 선정

본 논문에서는 내장형 시스템의 하드웨어와 소프트웨어 인터렉션으로 인해 생길 수 있는 오류를 발견할 수 있는 테스트 데이타를 선정함을 목적으로 한다. 프로그램 S의 테스트데이타 집합 T는 다음과 같이 프로그램 S의 출력과 P_f의 출력을 다르게 하는 입력 데이타로 정의한다.

$T = \{t \mid S(t) \neq P_{fi}(t),\ 1 \le i \le n,\ n=\text{FIT의 개수}\}$

DPPS 채널 A 프로그램 S는 9개의 아날로그 변수와 2개의 디지털 변수인 총 11개의 입력변수(Input variables)를 갖으며, 입력 변수 값에 따라 11개의 trip 신호를 출력한다. DPPS 프로그램 S의 테스트 데이타는 프로그램 S의 출력과 P_f의 출력을 다르게 하는 입력 변수의 데이타 값으로 선정한다. T 중에서 하나의 테스트 데이타 t와 t의 예상 출력 O(t), t를 S에 적용한 결과 S(t), t를 P_{f48}에 적용한 결과 $P_{f48}(t)$을 표 3에 표시하였다.

표 3 테스트 데이타 t∈T와 O(t), S(t), $P_{f48}(t)$

신호	A0	A1	A2	A3	A4	A5
t	1200	40	2300	300	200	1000
출력 변수	Trip_ A0	Trip_ A1	Trip_ A2	Trip_ A3	Trip_ A4	Trip_ A5
O(t)	y	y	y	n	y	y
S(t)	n	n	n	n	n	n
$P_{f48}(t)$	y	y	y	y	y	y
신호	A6	A7	A8	D0	D1	
t	800	400	40	7	8	
출력 변수	Trip_ A6	Trip_ A7	Trip_ A8	Trip_ D0	Trip_ D1	
O(t)	y	y	y	y	y	
S(t)	n	n	n	n	n	
$P_{f48}(t)$	y	y	y	y	n	

예를 들면, 테스트 데이타 t = (1200,40,2300,300,200, 1000,800,400,40,7,8)를 S와 P_{f48}에 적용하였을 때, 다음과 같이 다른 출력 값을 갖는다.

S(t) = (n,n,n,n,n,n,n,n,n,n,n)이며 11개의 출력에서 모두 trip이 발생하지 않음

$P_{f48}(t)$ = (y,y,y,y,y,y,y,y,y,y,n)이며 10개의 출력에서 trip이 발생

따라서 t는 S와 와 P_{f48}을 차별화할 수 있는 입력 데이타로서 제안한 기법에 의해 선정될 수 있는 테스트 데이타 중 하나이다.

표 3에서 기술하였듯이 t에 대한 예상 출력 O(t) = (y,y,y,n,y,y,y,y,y,y,y)이다. 그런데 S(t) = (n,n,n,n,n,n,n, n,n,n,n)로써 O(t)와 다르다. 다시 말하면, t는 DPPS에서 하드웨어와 소프트웨어 사이의 상호작용 오류를 발견할 수 있는 데이타가 된다는 의미이다.

올바른 프로그램의 경우에는 "digital[1].trip=9"가 없어야 하지만, 그림 5의 프로그램 S에는 밑줄로 표시한 부분에 하드웨어와 소프트웨어의 상호작용 오류가 있다. 프로그램 S의 밑줄로 표시한 "digital[1].trip=9"는 CPC로부터 읽어 들인 하드웨어 신호를 소프트웨어 신호로 바꿔야 하기 때문에, 하드웨어와 소프트웨어의 상호작용이 일어나는 부분이다. 따라서 "digital[1].trip=9"는 하드웨어 소프트웨어 사이의 상호작용 오류이다. 그러므로 선정한 테스트 데이타 t는 S의 밑줄 친 상호작용 오류를 발견할 수 있도록 하는 테스트 데이타임을 알 수 있다.

본 기법은 내장형 시스템의 하드웨어와 소프트웨어 사이의 상호작용에 의한 오류를 발견할 수 있는 테스트 데이타를 선정함을 목적으로 하며, 5절에서 제안한 기법의 우수성을 실험 결과를 토대로 기술한다.

5. 실 험

본 논문에서 제안한 기법은 내장형 시스템의 하드웨

어와 소프트웨어의 상호작용에 의한 오류를 발견하기 위한 테스트 데이타를 선정하는 것으로, 기존의 테스트 기법과 비교하여 제안한 기법의 우수성을 보이기 위하여 기법을 100% 커버(cover)하는 테스트 데이타 개수와 오류 발견 비율을 비교하였다.

실험 대상은 4절에서 예를 들어 기술한 DPPS 채널 A를 각각 다른 5 사람이 시뮬레이트 한 C 프로그램 S_1, S_2, S_3, S_4, S_5을 대상으로 5개의 실험, Exp_1, Exp_2, Exp_3, Exp_4, Exp_5를 수행하였다. 오류가 없는 DPPS 채널 A에 대한 C 프로그램을 S라고 할 때, 각 프로그램 S_i는 표 4에서처럼 각각 다른 상호작용 오류를 갖도록 하였으며, 각 실험 $Exp_i(1 \le i \le 5)$에 대하여 수행한 실험 절차는 아래의 세 단계를 거친다.

① 프로그램 S_i에 대하여 일반적인 테스트 기법을 적용하여 테스트 데이타를 구한다. 본 실험에서는 화이트 박스 테스트 데이타 생성 도구인 ATAC(Automatic Test Analysis for C)[18,19,20,21]을 이용하여, all-node, decision, c-use, p-use, all-use 기준(Criteria)을 만족하는 테스트 데이타 집합 T^g를 생성하였다.

② 프로그램 S_i에 표 2에서 식별한 FIT에 대하여 소프트웨어 오류로 변환한 하드웨어 오류를 삽입하여 프로그램 P_f들을 생성하였다.

③ T로부터 프로그램 S_i와 프로그램 P_f들을 차별화하는 테스트 데이타의 집합 T를 ①에서 구한 집합 T^g로부터 선정하였다.

표 4 기존 기법과 제안한 기법의 실험 데이타

Exp_i	S_i에 있는 오류	비교 항목	실험 데이타(오류 발견 비율)					
			기존	제안				
Exp_1	**오류** (라인 번호 = 187) digital[i].trip=9; **올바른 코드:** 스테이트먼트가 없어야 함	테스트 데이타 수	23	2	3	4	5	6
				12	56	97	36	2
		오류를 발견한 테스트 데이타 수	6	12	44	75	24	0
		평균 테스트 데이타 수	23 (0.26)	3.74 (0.76)				
Exp_2	**오류** (라인 번호 = 151) analog[3].opby_ set=400; **올바른 코드:** analog[3].opby_ set=40;	테스트 데이타 수	24	2	3	4	5	6
				10	68	105	42	0
		오류를 발견한 테스트 데이타 수	5	10	53	88	30	0
		평균 테스트 데이타 수	24 (0.21)	3.80 (0.80)				
Exp_3	**오류** (라인 번호 = 158) digital[1].opby_ set=10; **올바른 코드:** digital[1].opby_ set=0;	테스트 데이타 수	22	2	3	4	5	6
				4	34	63	26	0
		오류를 발견한 테스트 데이타 수	6	4	17	46	20	0
		평균 테스트 데이타 수	22 (0.27)	3.87 (0.69)				
Exp_4	**오류** (라인 번호 = 147) analog[1].trip_ set=500; **올바른 코드:** analog[1].trip_ set=300;	테스트 데이타 수	23	2	3	4	5	6
				5	54	81	33	0
		오류를 발견한 테스트 데이타 수	7	5	42	66	27	0
		평균 테스트 데이타 수	23 (0.30)	3.82 (0.81)				
Exp_5	**오류** (라인 번호 = 292) analog[8].trip_ bistable[A]=1; **올바른 코드:** analog[8].trip_ bistable[A]=0;	테스트 데이타 수	22	2	3	4	5	6
				0	33	57	26	0
		오류를 발견한 테스트 데이타 수	5	0	15	45	22	0
		평균 테스트 데이타 수	22 (0.23)	3.94 (0.71)				

5.1 테스트 데이타 개수의 비교

기존 기법에 의한 테스트 데이타의 개수와 제안한 기법에 의한 테스트 데이타의 개수를 표 4와 그림 6의 막대그래프를 통해 나타내었다. 표 4에서 보는 바와 같이 Exp_1의 경우, 기존 기법을 100% 커버하는 테스트 데이타 T^g의 개수는 23개이며, S_1과 S_1로부터 생성한 모든 P_f들을 차별화하는 데이타 즉, 제안한 기법을 100% 커버하는 데이타를 분석하면 다음과 같다. 2개의 테스트 데이타로 제안한 기법을 100% 커버하는 경우는 10가지, 3개의 테스트 데이타로 제안한 기법을 100% 커버하는 경우는 56가지, 4개의 테스트 데이타로 제안한 기법을 100% 커버하는 경우는 97 가지, 5개의 테스트 데이타로 제안한 기법을 100% 커버하는 경우는 36 가지, 6개의 테스트 데이타로 제안한 기법을 100% 커버하는 경우는

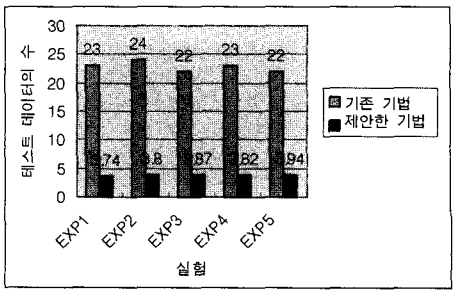

그림 6 테스트 데이타의 수 비교

2가지로, 제안한 기법을 100% 커버하는 평균 테스트 데이타 수는 3.74 개다.

그러므로 Exp_1의 경우 기존의 방법에 의해 구한 테스트 데이타는 23개이나, 제안한 기법에 의해 생성된 테스트 데이타는 평균 개로, 제안한 기법이 기존의 기법에 비해 적은 테스트 데이타가 소요됨을 알 수 있다. Exp_1과 마찬가지로 Exp_2, Exp_3, Exp_4, Exp_5에 대한 평균 테스트 데이타의 개수는 3.80, 3.87, 3.82, 3.94로, 제안한 기법에 의해 생성되는 테스트 데이타가 기존의 기법에 비해 상당히 적음을 알 수 있으며, 각 실험 별로, 기존 기법과 제안한 기법의 평균 테스트 데이타의 수를 그림 6의 막대 그래프를 통해 나타내었다.

5.2 오류 발견 비율 비교

오류가 없는 프로그램 S에 상호 작용 오류를 인위적으로 삽입하여 작성한, S_1, S_2, S_3, S_4, S_5를 대상으로 오류 발견 비율을 측정하였으며, 기존 기법에 의한 오류 발견 비율과 제안한 기법의 오류 발견 비율을 표 4의 소괄호와 그림 7의 막대그래프를 통해 나타내었다. 오류 발견 비율을 다음과 같이 정의한다.

오류 발견 비율 = 오류를 발견한 테스트 데이타의 수 / 총 생성된 테스트 데이타의 수

Exp_1의 경우, 기존의 기법은 23개의 테스트 데이타 중 6개의 테스트 데이타가 오류를 발견하였으므로, 이때의 오류 발견 비율은 0.26이다. 제안한 기법의 경우, 생성된 총 테스트 데이타 203 개 중 155 개의 데이타가 오류를 발견하였으므로, 이때의 오류 발견 비율은 0.76이다. Exp_1과 마찬가지로 Exp_2, Exp_3, Exp_4, Exp_5에 대한 오류 감지율도 0.80, 0.69, 0.81, 0.71로, 각 실험별 오류 발견 비율은 그림 7의 막대 그래프로 나타내었으며, 이를 통해 제안한 기법이 기존 기법에 비해 높은 오류 발견 비율이 높음을 알 수 있었다.

5.1절의 테스트 데이타의 개수 비교와 5.2절의 오류 발견 비율 비교를 통해, 본 논문에서 제안하는 기법이 기존의 테스트 기법에 비해 적은 수의 테스트 데이타를 생성하였으며, 이 적은 수의 테스트 데이타로도, 기존 기법에 비해 오류 발견 비율이 높음을 알 수 있었으며,

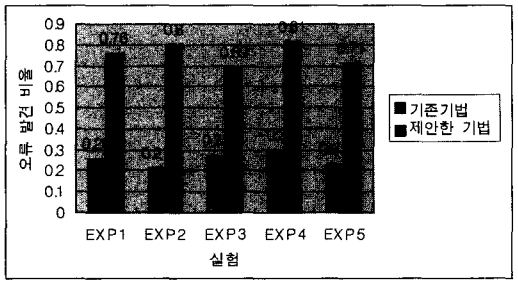

그림 7 오류 발견 비율 비교

따라서 제안한 기법이 하드웨어와 소프트웨어의 상호작용에 의한 오류를 발견할 수 있는 효과적인 테스트 기법임을 분석하였다.

6. 결론 및 향후 과제

내장형 시스템의 기능이 복잡해지면서 내장형 시스템에 대한 테스트의 중요성이 인식되고 있으나, 아직은 내장형 시스템 테스트에 대한 연구가 미흡한 실정이다. 본 논문에서 사례로 든 원전 보호계통 시스템인 DPPS와 같이 위험도가 높은 시스템을 테스트를 한다는 것은 비용도 많이 들고, 위험도 높기 때문에 내장형 시스템의 하드웨어와 소프트웨어의 상호작용 의해 발생되는 오류를 발견하기 위한 효과적인 테스트 기법이 필요하였다.

본 논문에서는 DPPS 채널 A의 하드웨어와 소프트웨어 모듈을 분석하여 HSBD를 작성하고 이를 토대로 DPPS 채널 A의 동작을 C 프로그래밍 언어로 시뮬레이트 한 코드 S를 작성하였으며, 하드웨어 오류를 소프트웨어 오류로 전환하여 프로그램 P_f들을 생성하고, S와 P_f들의 출력을 다르게 하는 입력 데이타를 DPPS 채널 A의 하드웨어와 소프트웨어의 상호작용에 의한 오류를 발견하는 테스트 데이타로 정의하였다.

제안한 테스트 기법의 효율성을 분석하기 위하여 DPPS 채널 A의 동작을 시뮬레이트 한 프로그램들을 실험 대상으로 하여, 기존의 화이트박스 테스트 기법에 비해 테스트 데이타의 개수와 오류 발견 비율에 대한 분석을 하였다. 제안한 기법에 의해 생성되는 테스트 데이타의 개수는 기존 테스트 기법에 의한 테스트 데이타의 개수보다 적은 수의 테스트 데이타 개수가 필요하였으며, 이 적은 수의 테스트 데이타로도 기존 테스트 기법보다 높은 오류 발견 비율을 얻을 수 있었다.

향후에는 본 논문에서 제시한 기법을 도구로 구현하고, 실제 내장형 시스템의 하드웨어 소프트웨어의 상호작용 의한 오류를 발견하기 위한 테스트 데이타를 자동 생성할 수 있도록 할 예정이다. 또한 본 논문에서는 제안한 기법을 DPPS 채널 A에만 적용하여 사례 연구 및 실험을 수행하였으나, 완성된 도구를 다양한 내장형 시스템에 적용하고, 다양한 실험을 수행하여 본 기법의 우수성을 나타낼 예정이다.

참 고 문 헌

[1] E. A. Lee, What's Ahead for Embedded Software?, *IEEE Computer*, pp 18-26, September, 2000.

[2] E. A. Lee, "Computing for embedded systems," proceeding of IEEE Instrumentation and Measurement Technology Conference, Budapest, Hungary, May, 2001.

[3] Mattias O'Nils and Axel Jantsch, "Communication in Hardware/Software Embedded Systems-A Taxonomy and Problem Formulation," in Proceeding of 15th NORCHIP Conference, November, 1997.
[4] M.Hsueh, Fault Injection Techniques and Tools, *Computer*, Apr, 1997, pp75-82.
[5] J.F. Clark, Fault Injection, *Computer*, June, 1995, pp47-56.
[6] Charlse R. Yount 외, A Methodology for the Rapid Injection of Transient Hardware Errors, *IEEE Transactions on Computers*, VOL. 45. No.8, Aug, 1996.
[7] Gwan S. Choi, Simulated Fault Injection : A Methodology to Evaluate Fault Tolerant Microprocessor Archtecture, *IEEE Transactions on Reliability*, VOL. 39, NO. 4, Oct, 1990.
[8] Dimiter Avresky, Fault Injection for formal testing of fault tolerance, *IEEE Transactions on Reliability*, VOL. 45, NO.3, Sept, 1996.
[9] J.V. Carreira, Fault Injection spot-checks computer system dependability, *IEEE Spectrum*, Aug, 1999.
[10] J.V. Carreira, Xeption : A technique for the Experimental Evaluation of Dependability in modern computers, *IEEE Transaction on Software Engineering*. VOL 23, No.2, Feb, 1998.
[11] Johan Karsson, Using Heavy-ion Radiation to validate fault-handling mechanisms, *IEEE micro*, 1994, pp8-23.
[12] Marcello Dalpasso, Fault Simulation of parametric Bridging Faults in CMOS ICs, *IEEE Transaction on computer aided design of integrated circuits and systems*, vol 12. no. 9, Sept, 1993.
[13] Jean Alert 외, Validation Based Development of dependable systems, *IEEE micro*, Jul, 1999, pp 66-79.
[14] J. Voas, Certifying software for high assurance environments, *IEEE Software*, Jul/Aug, 1999, pp48-54.
[15] T. A. Delong, A Fault Injection Technique for VHDL Behavioral-Level Models, *IEEE Design and Test of Computers*, 1996, pp24-33.
[16] Paul C. Jorgensen, *Software Testing - A Craftsman's Approach*, CRC Press, 1995.
[17] UCN 3&4 Final Safety Analysis Report, Volume 11, Korea Electric Power Corporations.
[18] J.R. Horgan and S. London, "ATAC : A data flow coverage testing tool for C," in Proceedings of Symposium on Assessment of Quality Software Development Tools, pp2-10, New Orleans, LA, May, 1992.
[19] M.R Lyu, J.R. Horgan and S. London, "A coverage analysis tool for the effectiveness of software testing," in Proceeding of International Symposium on Software Reliability Engineering, 1993.
[20] Li, N., Malaiya, Y. K., Denton J, "Estimating the Number of Defects: A Simple and Intuitive Approach," in Proceeding of 7th International Symposium on Software Reliability Engineering, 1998.
[21] Burr, Kevin and Young, William, "Combinatorial Test Techniques: Table-Based Automation, Test Generation, and Test Coverage," in Proceeding of International Conference on Software Testing, Analysis, and Review. San Diego, CA, October, 1998.
[22] Richard A. DeMillo, Richard J. Lipton, and Frederick G. Sayward, *Hints on test data selection: Help for the practicing programmer*. IEEE Computer, 11(4):34-41, April 1978.
[23] M. E. Delamaro, J. C. Maldonado, and A. P. Mathur., "Integration Testing Using Interface Mutation," In Proceedings of International Symposium on Software Reliability Engineering (ISSRE '96), pages 112--121, April 1996.

최 병 주
정보과학회논문지 : 소프트웨어 및 응용
제 30 권 제 10 호 참조

성 아 영
1996년~2000년 이화여대 컴퓨터학과 학사. 2000년~2002년 이화여대 컴퓨터학과 석사. 2002년~ 이화여대 컴퓨터학과 박사과정. 관심분야는 소프트웨어 공학, 소프트웨어 테스트 및 테스트 프로세스