최신 프로세서 탑재 비행제어 컴퓨터의 통합시험을 위한 프로세서 모니터링 연구

A Study on Processor Monitoring for Integration Test of Flight Control Computer equipped with A Modern Processor

이 철*, 김 재 철, 조 인 제
(Cheol Lee, Jae-Cheol Kim, and In-Jae Cho)

Abstract : This paper describes limitations and solutions of the existing processor-monitoring concept for a military supersonics aircraft Flight Control Computer (FLCC) equipped with modern architecture processor to perform the system integration test. Safe-critical FLCC integration test, which requires automatic test for thousands of test cases and real-time input/output test condition generation, depends on the processor-monitoring device called Processor Interface (PI). The PI, which relies upon on the FLCC processor's external address and data-bus data, has some limitations due to multi-fetching capability of the modern sophisticated military processors, like C6000's VLIW (Very-Long Instruction Word) architecture and PowerPC's Superscalar architecture. Several techniques for limitations were developed and proper monitoring approach was presented for modern processor-adopted FLCC system integration test.

Keywords : Flight Control Computer(FLCC), processor interface, integrated test, debug agent, VLIW(Very Long Instruction Word)

I. 서론

현대의 전투기 개발은 기동성능을 향상시키기 위하여 정적으로 항공기를 불안정하게 설계하는 정안정성 완화개념(RSS: Relaxed Static Stability) 적용을 보편화하고 있다. 불안정하게 설계된 항공기의 안정성 및 조종성을 보장하기 위해서는 센서와 조종면 액추에이터 및 디지털 제어기술에 의한 전자식 비행제어컴퓨터(digital fly-by-wire flight control computer)가 필수적이 되어야 한다. 그러나 전자식 장비는 초기 항공기에서 사용된 기계적 링크(mechanical linkage) 시스템보다 많은 복잡성으로 인해 결함 가능성을 내재하고 있다[1]. 이에 대하여 디지털 비행제어 시스템은 다중화(redundancy)를 가지도록 개발되어 PLOC(Probability of Loss of Control)를 향상시키고 있다. 최근에 개발된 초음속 전투기급 훈련기의 경우, 세 채널의 독립된 비행제어 컴퓨터(Flight Control Computer, FLCC)를 가지고 있다. 각 채널의 컴퓨터는 각각의 독립된 전원 및 입출력을 가지며, 다른 채널의 입출력 데이터를 서로 공유하기 위해 Hardwired CCDL(Cross-Channel Data Link)를 가진다. 각각의 컴퓨터는 인접한 두개의 컴퓨터로부터 받은 데이터와 비교하여 신호 선택을 수행하다. 신호별 허용 범위를 넘어서 일정지속(persistence limit)을 가질 경우 결함분석(failure analysis) 및 결함 격리와 재형상(reconfiguration)을 수행하게 된다. 또한, 임의 채널 컴퓨터 전체에 대한 격리는 프로세서와 독립된 하드웨어 로직(channel valid logic)이 수행하며, 외부통신 및 CCDL과 모든 엑추에이터 출력을 끊어지도록 하여, 결함에 대한 전파를 보호하고 있다.

그림 1은 비행제어 소프트웨어에 대한 안전인증(flight safety certificate) 절차를 도시한 것이다. 소프트웨어 개발그룹으로부터 배포된 비행제어 소프트웨어는 개발그룹과 독립된 테스트 그룹에서 검증을 수행하게 된다. 그림에서 Stand Alone V&V(Verification and Validation)는 Open Loop Static 상황에서 소프트웨어 레벨의 요구도 검증 절차를 나타내고 있으며, 이후 MILS(Man In the Loop System)조건에서의 Integrated V&V는 시스템 레벨의 요구도가 검증된다. 이후 FMET (Failure Mode Effectiveness Test)에서는 비행제어컴퓨터의 다중화 기능에 있어서 하드웨어 및 소프트웨어 대한 결함 강제주입을 통해, 결함 인가 조건에서도 안전(fail safe) 비행이 가능함을 실제 조종사를 통해 검증 수행한다.

안전 중시의 비행제어 개발 프로세스에서는 비행제어 소프트웨어가 배포 될 때마다, 검증 프로세스 처음부터 재 수행하게 된다. 비행제어 소프트웨어의 검증은 변경된 부분을 검증할 뿐만이 아니라, 변경된 부분이 다른 모든 부분에 영향이 없음 또한 증명해야 하기 때문이다. 특별히 소프트웨어

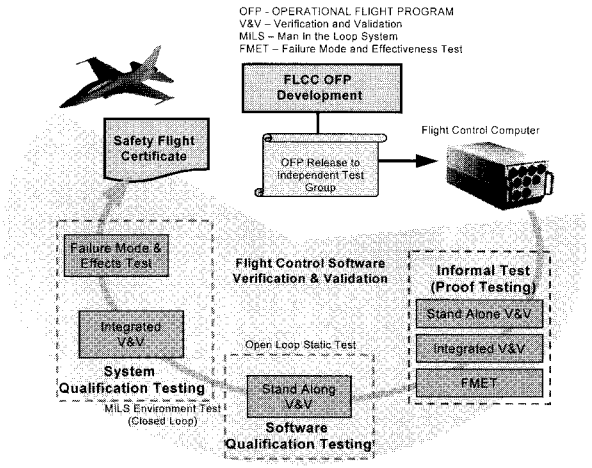


그림 1. 비행제어 소프트웨어 검증 절차.
Fig. 1. Flight control software V&V process.

* 책임저자(Corresponding Author)
논문접수 : 2008. 2. 21., 채택확정 : 2008. 5. 28.
이 철, 김재철, 조인제 : 한국항공우주산업㈜ 비행역학팀
(newsong@koreaaero.com/jckimT50@koreaaero.com/fogchoij@koreaaero.com)

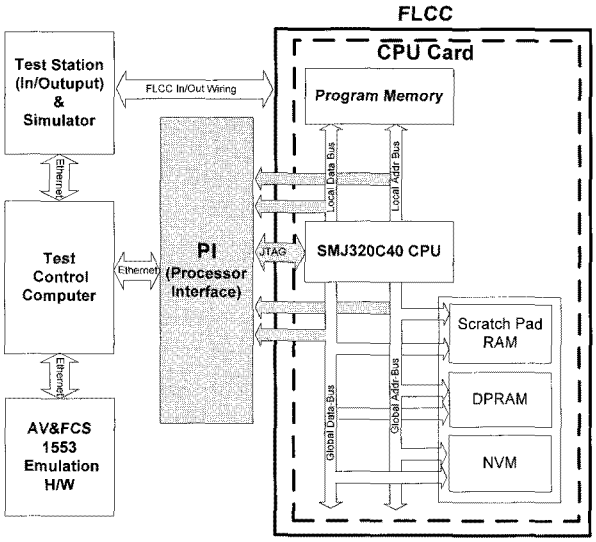


그림 2. PI(Processor Interface)와 FLCC 인터페이스.
Fig. 2. FLCC Interface with PI.

요구도 검증을 위해 오천가지 이상의 테스트케이스들이 생성된다. 안전중시 비행제어 소프트웨어 개발 프로세스상 작은 부분의 소프트웨어 수정이 되더라도 매번 전체 검증 프로세스를 다시 거쳐야 하므로, 소프트웨어 개발 및 검증에 많은 시간이 소요와 및 휴먼에러(human error) 가능성에 대하여, 자동화된 시험환경이 반드시 필요로 하게 된다.

그림 2에서와 같이 시험 통제컴퓨터는 전체시험을 자동화하여 수행하며, 비행제어 인터페이스가 각 시험 조건별 만들어 질 수 있도록 입출력을 관장하는 테스트 스테이션 컴퓨터와 항공전자 및 비행제어 1553 인터페이스 장비와 연동된다. 이와 더불어, 요구도 검증을 위한 자동화 된 시험 조건을 위해 FLCC 실시간 처리에 영향을 주지 않으며, 각종 시험조건의 이벤트가 발생되어 스냅샷 등의 대량의 데이터 획득 또는 강제 변경을 수행하는 장비가 필요하게 된다. 이와 같이, 자동화된 테스트 수행을 위해 프로세서 동작제어, 메모리와 레지스터의 읽기와 쓰기, 메모리 스냅샷 및 메모리 강제변경의 사이클 등의 기능을 3개의 독립된 채널에 대해 수행하는 장비를 여기서는 PI 장비(Processor Interface)라 부르며, 그림 1의 가운데서 표현되어 있다. 일반 상용 에뮬레이터 등에서도 스크립트를 통해서는 자동화 시험수행을 어느 정도 구현할 수도 있지만, 앞서 말한 바와 같이 비행제어 컴퓨터 외에 비행제어 모든 입출력 장비와 실시간 연동되도록 자동화 해야 하므로 상용 에뮬레이터 환경으로는 불가능하며 별도로 개발된 모든 FLCC 입출력(Avionic 1553, FCS 1553, RS-422, Discrete Input/Output, Analog Input/Output)을 3채널 컴퓨터에 모두 관장할 수 있는 시험 통제 소프트웨어와 PI 장비를 통해 시험수행을 하고 있다. 프로세서와 직접 하드웨어적으로 맞물리는 PI 장비는 프로세서 외부버스 공유를 통해 이루어지므로, 비행제어 컴퓨터의 프로그램 메모리와 데이터 메모리는 프로세서 외부에 위치되어야만 하는 단점이 있다.

본 논문에서는 비행제어 컴퓨터의 최신 고속 프로세서 업그레이드 과정에서 발생된 프로세서 모니터링 문제점들에 대하여 분석을 수행하였다.

II. 본론

1. 시스템 통합시험과 Processor Interface장비

비행제어 통합시험 환경은 그림 2 또는 3과 같이 구성이 된다. 먼저 가운데 위치한 테스트 스테이션은 비행제어 컴퓨터와 항공기와의 모든 입출력 신호를 연결을 담당하며, 비행제어 컴퓨터의 전원 공급과 각종 항공기내 탑재 장비들과 인터페이스 되도록 한다. 유압 엑추에이터 및 센서의 아날로그 상사모델이 내장되어 있으며, 각종 신호의 통제 및 모니터링 및 자동 보정 기능을 가지고 있다. 그림 3 상단의 조종석에서는 조종간과 러더브레이크 페달 및 각종 계기판 등 실제 하드웨어가 장착되며 그림과 같이 테스트 스테이션과 연결되어 비행제어 컴퓨터로 이어지게 된다. 조종석 앞의 지형영상은 공력/중량/힌지모멘트/ 추진력 정보가 포함된 시뮬레이션 데이터베이스와 신호연동장비를 통해 화면에 뿌려지게 되며, 모든 신호는 테스트 스테이션과 연동된다. 테스트 스테이션에서는 실제 조종면 엑추에이터와 연결될 수 있다. 좌측 상단의 PI 장비는 비행제어컴퓨터의 프로세서 외부 어드레스 및 데이터 버스 및 JTAG 라인과 커넥터를 통해 연결된다. 비행제어 컴퓨터는 PI를 통해 모니터링 또는 제어되며 프로세서 동작제어(run/halt), 브레이크포인트(breakpoint), 레지스터 읽기/쓰기, 메모리 읽기/쓰기, 메모리 스냅샷, 메모리 사이클 및 조합된 이벤트 트리거링(combined event triggering) 기능을 통해 각종 통합시험수행을 지원하게 된다. 실시간 자동화된 시험수행을 위해 중요한 부분인 메모리 읽기 및 메모리 스냅샷 기능은 FLCC 실시간 동작에 전혀 영향을 주지 않도록 PI 내에 쉐도우 메모리를 통해 1:1 쉐도우가 수행하며, 부가적으로 프로세서를 임시 홀딩 시킨 후 프로세서 외부버스를 직접 엑세스하여 메모리 값의 읽기 또는 강제 변경을 수행할 수 있다. 이러한 PI 장비를 통하여 비행제어컴퓨터 Open Loop Static 상의 소프트웨어의 요구도 검증 및 MILS(Man-In-the-Loop System) 상의 시스템 요구도 검증과 결함모드시험을 지원하게 된다.

이러한 비행제어 컴퓨터 통합시험을 위한 PI 장비에서는 프로세서 모니터링을 위해 프로세서 외부버스를 사용하므로

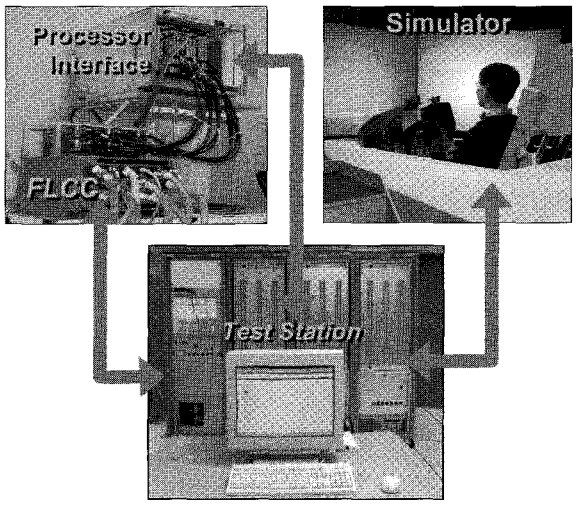


그림 3. 비행제어 통합시험 환경.
Fig. 3. Integration test environment of FLCC.

프로그램과 데이터 메모리를 프로세서 외부버스에 물려야 하는 제한을 가지고 있다. 다음절에서부터 PI 장비와 함께 최근의 고속 프로세서를 적용 시 발생되는 문제들을 보기로 한다.

2. 프로세서의 고속화에 따른 Processor Interface장비 문제점

2.1 기존 프로세서 아키텍쳐

현재의 프로세서들은 고속화를 위해 내부메모리 또는 캐시사용을 기본전제로 프로세서 아키텍쳐가 설계가 되고 있다. 최근에 개발된 초음속 훈련기의 비행제어 컴퓨터는 SMJ320C40 프로세서와 디지털 및 아날로그 입출력을 담당하는 입출력 프로세서(IO processor)로서 SMJ320C50 프로세서를 사용하였다. 프로그램 메모리로 플레쉬 롬(flash ROM)에 연결된 지역버스(ocal bus)와 데이터메모리로 scratch-pad RAM (SRAM), DPRAM 및 NVM에 연결된 광역버스(global bus)를 가지도록 되어, 외부메모리까지 하버드(harvard) 아키텍쳐가 구성될수 있다. 프로그램과 데이터 메모리가 거의 Zero-Wait State을 가지도록 운용되고있어 외부메모리가 CPU 1 Clock에 접근이 가능하다. 그림 4은 SMJ320C40 프로세서에서 분기명령(branch instruction)이 발생될 때 어드레스 버스의 데이터를 logic analyzer(NI PXI-6542)를 통해 획득한 결과 및 해당 실행코드의 disassembly code이다. 그림 4 상단 그림은 분기명령 발생시 그 이후 어드레스 인스트럭션에 대해 "pre-fetch"가 단 1개만 발생한다는 것을 보여주고 있다. 즉, 프로세서 외부버스에 지나가는 데이터에 의존하는 기존 PI에서는 C40과 같은 단순구조의 프로세서에서 브레이크 포인트 이벤트를 잡는데 있어 효과적일 수 있음을 보여주고 있다[2]. 하단 그림에서는 이러한 1개의 pre-fetch만 발생하는 원인에 대해 설명을 해주고 있다. 즉, 분기명령이 외부 프로그램 메모 리에서 프로세서 코어 안으로 들어와서 파이프라인 디코드(decode)

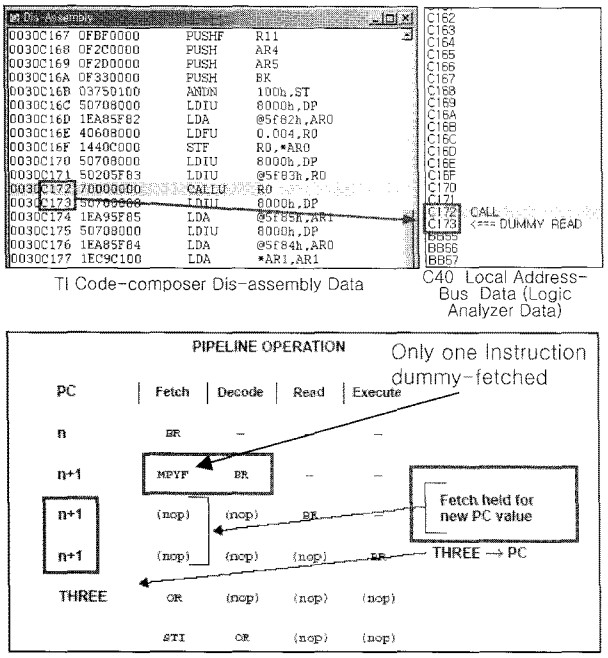


그림 4. Branch instruction에 따른 C40 파이프라인 작동.

Fig. 4. C40 pipeline operation about branch instruction.

시점에 오게 되면, 파이프라인 컨트롤러는 더 이상 fetch를 수행하지 않도록 fetch 홀딩을 하며 분기 이후 파이프라인 플러싱(pipeline flushing)을 대비한다[3]. 이때, 명령어는 디코드 시점에 와야 해독이 되므로 fetch 단계에 하나의 명령어가 미리 와 있게 되므로 pre-fetch가 1개 발생한다. 참고로, 각종 분기 조건에 따른 여러 가지 분기 명령어가 있을 수 있다. SMJ320C40과 같이 단순구조 프로세서는 PI의 브레이크포인트 로직에서 잘못된 이벤트를 발생시킬 가능성이 상당히 작으며, 멀티패칭(multi-fetching)에 의한 중복 브레이크포인트 문제가 발생되지 않는다. 세부내용은 다음절에서 설명한다.

2.2 최신 프로세서 업그레이드 및 이에 따른 PI 문제점

군용항공기를 개발하는 선진사 사례를 볼 때, 군용 비행제어 컴퓨터의 프로세서 선정을 위한 trade-off 스터디 수행 시 프로세서의 단종 또는 생산기간은 중요한 선정 지표 중의 하나이다[4,5]. 선진사의 경우 정부차원에서 운용되는 프로그램을 통해 군용 장비의 오래된 프로세서를 최신화를 수행하기도 한다. 또한 계속적으로 증대되는 실시간 처리량(throughput)의 요구 및 프로세서 단종 또는 생산기간, 그리고 프로세서 및 관련 개발장비에 대한 유지보수 지원을 고려할 때, 프로세서 선정은 어느 정도 최신의 프로세서가 채택될 수 밖에 없다[6]. 상기와 같은 이유로 최신의 프로세서 선정하되, 최근의 국내외 군용 장비에서 많이 사용되는 TI 프로세서 및 PowerPC 계열이 많이 사용되고 있으며, 모두 많은 명령어들이 프로그램 메모리로부터 멀티패칭되어 프로세서 코어 안으로 들어가는 방식이 사용되고 있다. 이들 프로세서에서는 VLIW(Very-Long Instruction Word)와 super-scalar 방식을 각각 사용하고 있으며, VLIW 아키텍쳐는 컴파일 단계에서 8세트 명령어(8 Set Instructions)를 패치패킷(fetch packet) 형식으로 구분을 어셈블리 코드를 구성하는 반면, super-scalar 아키텍쳐는 컴파일 단계가 아닌 런타임(run-time) 상황 즉, 프로세서 자원가용에 따라 유동적으로 다수의 명령어가 멀티패칭된다[7]. 두 아키텍쳐 모두에서, 프로그램 메모리를 외부에 둘 경우 PI 장비 사용에 있어 몇 가지 문제점이 발생된다.

첫째로 패치패킷 방식과 깊은 파이프라인(deep pipeline)구조로 인하여 분기 명령어(branch instruction)이 디코드(decode) 단계에 왔을 때 이미 execute 시점에 해당하는 어드레스 이후 많은 명령어들이 프로세서안으로 pre-fetch 된 상태이므로 프로세서 외부버스를 모니터링하는 PI는 분기명령어 이하 어드레스에 설정된 브레이크포인트 로직이 읽어 잘못된 브레이크 포인트(false breakpoint) 이벤트가 발생할 수가 있다. 둘째는 TI C6000 VLIW 프로세서에만 해당되는 것으로서 8세트 패치패킷 방식 사용에 따른 문제점으로 8세트 패킷 내의 분기명령어가 동일한 패치패킷 내로 다시 분기해 올 경우, 상기와 같은 잘못된 브레이크 포인트(false breakpoint) 이벤트가 발생에 따른 문제이다. 셋째는 PI 장비의 브레이크포인트 로직에서 인지한 브레이크포인트 이벤트 어드레스는 실제 프로세서 파이프라인의 execute 시점과는 시간적 차이로 인한 문제이다. 이는 pre-fetch된 많은 명령어들이 파이프라인에 적재되어 있어 발생된다.

이러한 프로세서의 고속 고효율의 방식으로의 진화에 따라서, 비행제어 시스템 통합시험을 위한 PI 장비의 모니터링

방식이 바뀌어야 됨이 요구되고 있다. 다음 절에서 세부 문제점 및 해결방안을 논의하였다.

2.3 문제 1(pre-fetch에 의한 false breakpoint) 및 해결방법

앞서 간략히 설명한 바와 같이, 실제 설정된 브레이크 포인트 어드레스에서 트리거 이벤트가 발생하지 않고 다른 잘못된 어드레스에서 발생되는 현상이다. Pre-fetch된 명령어들은 이미 프로세서 외부 버스를 지나서 프로세서 코어 안으로 들어와 있으나, 이 명령어들은 실제 수행되는 코드가 아니며 분기명령어 아래 어드레스들에 위치한 코드들이다. 분기 명령어 실행(execute)시 파이프라인 내에 적재된 상기 명령어들은 플러싱(flushing) 된다. 외부버스를 모니터링하고 있는 PI에서는 분기명령어 이후 인접한 번지에 브레이크포인트가 설정될 경우, 잘못된 브레이크포인트(false breakpoint) 이벤트를 발생하게 된다. 개발된 비행제어 컴퓨터 소프트웨어의 경우 전체 유닛이 600여 개이며, 함수 호출 및 복귀와 각종 조건 및 순환문에서 발생되는 많은 양의 분기명령어로 인하여 통합시험 수행에 큰 지장을 주게 되었다.

이전 장에서, SMJ320C40의 경우 분기 명령어에 의한 pre-fetch는 1개가 발생되는 것이 실험적으로 확인되었다. 그러나 VLIW 프로세서인 SMJ320C6000 계열의 경우, 컴파일 시 명령어 조건에 따라 8세트 명령어가 한꺼번에 끝가지 내려가지 않을 수 있으므로, 분기명령어 하위 어드레스의 명령어들이 대략 20여 개가 pre-fetch 되는 것으로 실험적으로 확인되었다.

이러한 발생원인은 고속 프로세서가 갖는 VLIW의 넓고 깊은 파이프라인(large width and deep pipeline)으로 인해 발생이 되는 것이다. 그림 5에서 분기 명령어가 디코드(decode) 시점에 내려오기 전에 많은 양의 명령어들이 프로세서 외부버스를 통해 fetch되어 파이프라인 안으로 들어온 상태가 되며, PI는 분기명령어 이후 외부버스를 추가로 지나가는 원치 않는 20여 개의 명령어를 모니터링 하게 되므로 잘못된 브레이크포인트(false breakpoint) 이벤트가 발생하게 된다.

VLIW 프로세서인 C6000의 경우 프로그램 코드(.text)가 프로세서 내부메모리에 위치할 경우, 8세트 패치 패킷이 CPU 1 Clock에 패치 되어 파이프라인을 이동을 할 수 있다. 그러나 프로그램 코드(.text)가 프로세서 외부에 있게 되면, 하나의 패치 패킷(8세트 명령어)을 구성하기 위해 외부 프로그램 메모리를 8회 액세스해야 하며 실시간 처리량(throughput) 감소 효과를 가져온다[8,9]. 개발초기 단계에서 asynchronous 타입인 flash ROM을 프로그램 메모리를 사용하므로 초기 개발당시 실시간 처리량의 부족으로 인하여 프로세서-메모리 아키텍쳐를 변경하여 실시간 처리량을 향상시켰다. 비행제어 컴퓨터에서는 안전성의 이유로 프로그램 메모리가 Flash ROM에서 바로 수행되도록 설계되어 왔다. 실시간 처리량 향상을 위해 synchronous SRAM에 복사하여 수행하도록 금번에 설계변경이 되었으며 synchronous SRAM 사용에 따른 안전성 분석이 수행되었다[2].

분기 발생에 따른 pre-fetch로 잘못된 브레이크 포인트 이벤트를 해결하기 위해 여러 방법이 검토되었다. 비행제어 컴퓨터 통합시험에서는 대부분의 테스트 케이스들이 OFP 유닛

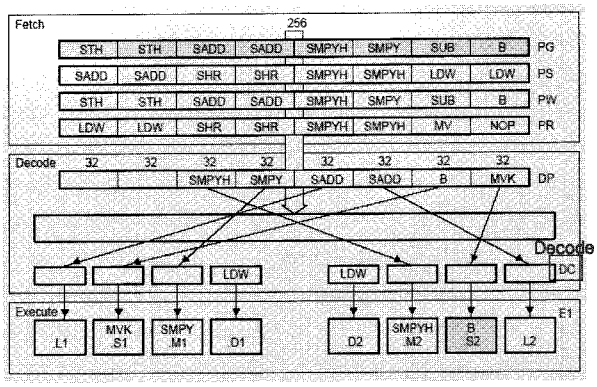


그림 5. SMJ320C6000 파이프라인 구조.

Fig. 5. SMJ320C6000 pipeline structure.

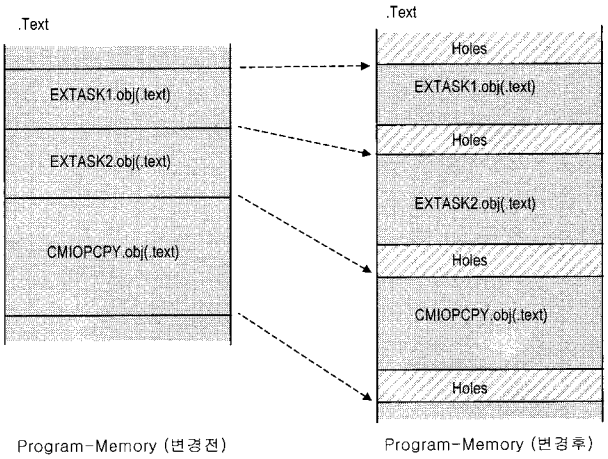


그림 6. Program memory상 code unit별 hole 삽입.

Fig. 6. Hole insertion in between of units on program-memory.

```
MEMORY
{
    INT_VECT_Table          o=0x20000000,  l=0x600,   f=0x00380362
    M_LG_ROLLYAW            o=0x20006600,  l=0x7700,  f=0x0
    M_LG_PITCH              o=0x2007D00,   l=0x40C0,  f=0x0
    M_HPCR_ROLL             o=0x200BDC0,   l=0x31E0,  f=0x0
    M_HP_STANDBYGAINS       o=0x200EFA0,   l=0x25E0,  f=0x0
    M_HPC_PITCH             o=0x2011590,   l=0x1AA0,  f=0x0
    M_FMINIT                o=0x2013020,   l=0x18C0,  f=0x0
    M_HP_COMMANDS           o=0x20148E0,   l=0x1820,  f=0x0
    M_CM1553AVSTXPRE15      o=0x2016100,   l=0x17A0,  f=0x0
    M_HPCR_CMDS             o=0x20179A0,   l=0x1EA0,  f=0x0
    M_CLFCTPPARAMCALC       o=0x2018F40,   l=0x1300,  f=0x0
    M_H_CONTROL_SURFACE_MIXER o=0x201A240, l=0x11E0,  f=0x0
    M_FMFRFSWTYPE1          o=0x201B420,   l=0x1100,  f=0x0
    ..............
}
SECTIONS
{
    .LG_ROLLYAW:            load=M_LG_ROLLYAW          {Debug\\LG_ROLLYAW.obj(.text)}
    .LG_PITCH:              load=M_LG_PITCH            {Debug\\LG_PITCH.obj(.text)}
    .HPCR_ROLL:             load=M_HPCR_ROLL           {Debug\\HPCR_ROLL.obj(.text)}
    .HP_STANDBYGAINS:       load=M_HP_STANDBYGAINS     {Debug\\HP_STANDBYGAINS.obj(.text)}
    .HPC_PITCH:             load=M_HPC_PITCH           {Debug\\HPC_PITCH.obj(.text)}
    .FMINIT:                load=M_FMINIT              {Debug\\FMINIT.obj(.text)}
    .HP_COMMANDS:           load=M_HP_COMMANDS         {Debug\\HP_COMMANDS.obj(.text)}
    .CM1553AVSTXPRE15:      load=M_CM1553AVSTXPRE15    {Debug\\CM1553AVSTXPRE15.obj(.text)}
    .HPCR_CMDS:             load=M_HPCR_CMDS           {Debug\\HPCR_CMDS.obj(.text)}
    .H_CONTROL_SURFACE:     load=M_H_CONTROL_SURFACE   {Debug\\H_CONTROL_SURFACE.obj(.text)}
    .CLFCTPPARAMCALC:       load=M_CLFCTPPARAMCALC     {Debug\\CLFCTPPARAMCALC.obj(.text)}
    .FMFRFSWTYPE1:          load=M_FMFRFSWTYPE1        {Debug\\FMFRFSWTYPE1.obj(.text)}
    ..............
}
```

그림 7. 수정된 link script.

Fig. 7. Modified link script.

시작 어드레스에 설정되며, 일부 유닛 끝 어드레스에 설정이 된다. 따라서, 유닛 시작 어드레스에 설정되는 브레이크 포인트 경우, 프로그램 메모리 상에 유닛별 코드영영(.text)에 hole을 강제로 내어주어 해결할 수 있었다[10]. 각 유닛의 시작 어드레스는 유닛 이름에 대한 심볼 어드레스가되므로, 링크 후 생성되는 맵 파일을 통해 해당 정보들을 추출하여, 그림

6과 그림 7에서와 같이 링크 스크립(link script)의 MEMORY와 SECTION부에 유닛 별 코드 위치를 강제 할당 되도록 하였다. 즉, 유닛 별 코드 위치를 링커(linker)에 맡기지 않고 유닛 내에는 프라그마(pragma)을 이용하여 심볼 이름과 SECTION 내 유닛 영역 이름을 매칭시킴으로 유닛 별 강제배치를 수행하되, 600여 개나 되는 소프트웨어의 유닛 배치에 있어 휴먼 에러를 없애기 위해 자동화하여 구현하였다. 유닛 끝 어드레스에 브레이크포인트 설정의 경우 그림 6 우측에서와 같이 20여개 이상의 NOP(No Operation) 명령어를 코드 내에 인라인(in-line) 어셈블리어로 삽입하여 잘못된 브레이크 포인트 이벤트가 발생하지 않도록 하였다.

2.4 문제 2(packet fetching에 따른 중복 breakpoint) 및 해결 방법

중복 브레이크 포인트 문제는 TI C6000 VLIW 프로세서에서 컴파일 시 결정되는 8세트로 맞추어지게 되는 패치패킷을 가져옴으로 발생되는 현상이다. 그림 8 상단과 같이 패치 패킷 내에 함수나 반복문의 분기 명령어가 있을 경우 분기 수행 후 복귀 시, 다시 8세트 패치 패킷을 처음부터 다시패치하기 때문에 발생한다. OFP 요구도 검증을 위한 테스트 케이스들은 대부분 유닛 시작 어드레스에 이벤트를 필요로 하므로 for 또는 while등의 반복문이나 함수 분기가 유닛 시작 어드레스에서 8개의 명령어 안에 있을 경우 발생하게 된다. 그림 8 하단에서와 같이 실제 브레이크포인트는 CLCONTROL 어드레스에 설정을 했으나 바로 아래 CLINPROC 함수 분기 후 되돌아올 때 CLCONTROL 어드레스가 프로세서 외부버스를 모니터링하는 PI장비에 중복으로 잡히기 때문이다. 이로 인하여 임의 곳에 브레이크포인트를 설정한 어드레스에서 8세트 패치 패킷 범위 안에 분기명령어가 있을 경우, 이러한 현상이 발생된다[11]. 해결 방법으로는 NOP(No Operation) 명령어를 코드 내에 인라인(in-line) 어셈블리어로 삽입하여 8세트 명령어 이상의 어드레스 차이가 나도록 하면 된다. 이러한 문제점은 함수 분기의 경우 1회 중복 이벤트가 발생되며 기타 반복문이 있는 경우 반복 횟수만큼 중복 브레이크포인트 이벤트가 발생된다.

```
#include "CL.H"
#include "SM.H"

void CLCONTROL(void)
{
    asm("NOP");
    asm("NOP");
    asm("NOP");
    asm("NOP");
    asm("NOP");
    asm("NOP");
    asm("NOP");
    asm("NOP");

    CLINPPROC();

    if ((FrameInSuperframe==1) && (Gains_Ready == TRUE))
    {
        LOWTOHIGH_INTERFACE();

        istrs_h = istrs_l;

    }
```

그림 8. Packet fetch방식과 중복 breakpoint 발생.
Fig. 8. VLIW packet fetch and plural breakpoint read.

2.5 문제 3(PI breakpoint의 시간차이) 및 해결방법

세 번째 문제점으로 앞서 설명한 기존 PI 장비의 브레이크포인트 하드웨어 로직에서 인지한 브레이크포인트 이벤트 어드레스는 실제 프로세서 파이프라인의 Execute 시점과 시간적 차이로 발생하는 문제이다. 그림 4에서와 같이, 기존 PI 장비와 pipeline depth가 짧고 간단한 프로세서 (예, SMJ320C40)의 경우 실제 패칭 시점과 파이프라인 실행시점과의 차이는 거의 발생되지 않는다고 볼 수 있다. 그 이유는CPU 1 clock에 외부 프로그램 메모리가 엑세스 되어 실제 파이프라인 이동속도와 외부 프로그램 메모리 패칭 속도가 같이 갈 수가 있다. 그림 4에서 보여주는 바와 같이, 패칭 시점과 파이프라인 실행(execute)시점과의 차이가 없을 수는 없다. 실제 데이터 메모리 변경까지를 고려할 때, 명령어별로 상기의 차이는 CPU 몇 클럭의 차이가 있을 수 있다. 따라서 기존 PI 장비에서는 정확한 브레이크 포인트를 걸어야 할 때에는 CPU 시뮬레이터를 이용하여 원하는 브레이크포인트 시점 대비 외부 프로그램 메모리 패칭 되는 시점을 고려하여 브레이크 포인트를 설정하였다.

그러나, 최근의 TI C6000 계열 또는 PowerPC의 멀티패칭 프로세서가 사용하게 되면 상기의 파아프라인 실행시점과 예상 패칭 시점을 추정하는 것이 상당히 어렵게 된다. 더욱이 프로세서 외부메모리를 사용하게 되면 문제는 더욱 복잡하게 된다. 한 개의 프로세서 외부버스를 통해 데이터와 프로그램을 런타임으로 콘트롤 되기 때문이다. 본 문제에 대하여 근원적으로 문제를 해결하는 방법은 CPU 코어에서 브레이크 포인틀 잡는 방법 외에는 없다. 즉, 기존 에뮬레이터나 OS 디버그 모듈을 사용하는 것이다. 그러나 비행제어 통합시험을 위한 IO 인터페이스 및 자동화시험을 구현하기 위해서는 인터페이스가 공개되지 않은 상용 장비를 사용하기가 어렵게 되며, 특히 OS의 경우 OS 자체에 대한 비행안전 인증(safety-critical qualification)의 문제로 검증에 대한 추가의 고려가 필요하다.

이와같이, OS의 디버그 모듈의 기능을 갖는 에이전트(debug agent) 도입을 하여 기존 비행제어 소프트웨어 executive와 연동되도록 하여 근원적인 해결을 수행할 수 있었다.

디버그 에이전트는 원격 호스트에서 명령을 받아 이를 해석하고 실행하며 그 결과를 호스트로 전송하는 역할을 하는 간이 소프트웨어이다[12,13]. 서론에서 언급한 바와 같이, 안전중시의 비행제어 소프트웨어 검증 절차에서 디버그 에이전트의 자체의 검증은 쉽지않은 문제일수 있다. 이 경우, 별도로 검증하지 않고 비행제어 소프트웨어 요구도 검증 시, 디버그 에이전트를 함께 검증을 할 수가 있다. 즉, 디버그 에이전트의 요구도 및 설계정보를 비행제어 소프트웨어 요구도 및 설계문서에 함께 반영하여, 설계 및 요구도 검증시험을 함께 수행한다.

최근의 고속프로세서의 경우, 앞서 언급한 두 가지 이유들로 인하여 프로세서 외부버스 데이터에 의존하는 PI 장비에서는정확한 프로그램 이벤트 트리거링이 어려웠다. 더욱이 세 번째 문제로서 많은 양의 명령어들이 pre-fetch되어 파이프라인에 적재되어 있으므로 파이프라인 execute시점과 외부버스를 지나왔을 시점의 시간 차이가 있으므로 정확한 디버

깅이 되지 않았다. 디버그 에이전트를 이용할 경우 정확한시점의 브레이크포인트 이벤트 발생을 통해 세가지 모든 문제를 해결할 수가 있다.

디버그 에이전트를 사용할 경우, 브레이크포인트 이벤트의 구현은 트랩(trap) 명령어의 기계어를 해당 설정 요구되는 브레이크 포인트 어드레스에 있는 임의 명령어와 교환(swap)하는 것이다. 이 방법은 일반적으로 상용 에뮬레이터의 소프트웨어 브레이크포인트(software breakpoint)와 소프트웨어 디버거(debugger)에서 사용되는 방법이다[14]. 상용 에뮬레이터의 하드웨어 브레이크(hardware breakpoint)의 경우 제한된 적은 개수가 지원되지만 소프트웨어 브레이크포인트의 경우 제한없이 사용 가능하다. 소프트웨어 인터럽트라고도 불리는 트랩기능은 SMJ320C40에서는 하드웨어적으로 지원이 되지만 SMJ320C6000계열은 지원이 안되며 소프트웨어적으로 구현을 해야만 했다. 트랩 기능이 지원될 경우, 어셈블리어 한 개 라인으로 구현이 가능하다. 이 경우 트랩은 인터럽트 발생시와 같이, 동일하게 PC(Program Counter)를 스택(stack)에 저장하고 인터럽트를 디스에이블(disable)하는 일련의 과정을 프

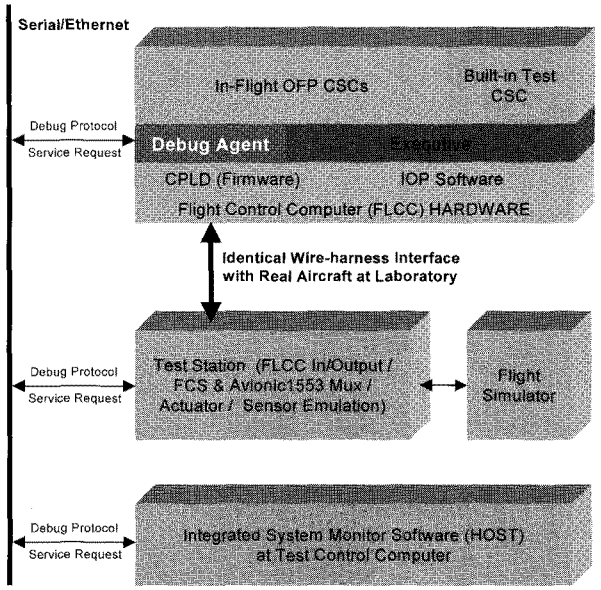


그림 9. 디버그 에이전트를 이용한 통합시험환경.
Fig. 9. Integrated test environment using debug agent.

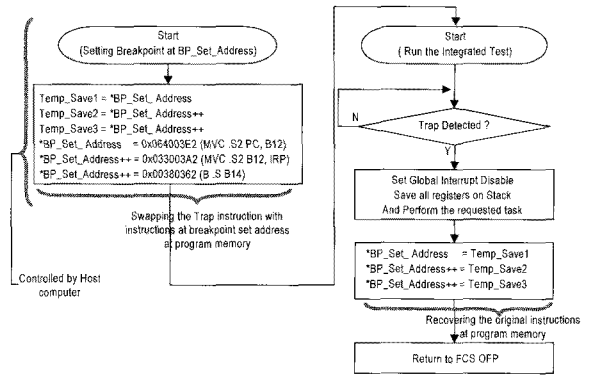


그림 10. 디버그 에이전트를 이용한 breakpoint 동작.
Fig. 10. Breakpoint mechanism in debug agent.

로세서 하드웨어에서 자동적으로 해주는 것이다. 그러나 C6000과 같이 소프트웨어적으로 구현해야 하는 경우는 연속적인 최소 3개의 어셈블리 명령어가 필요하다. 그림 9는 SMJ320C6000용 디버그 에이전트를 이용한 브레이크포인트 동작을 설명한 그림이다. 일반 레지스터인 B14 레지스터는 컴파일 시 메모리 모델 중의 하나인 코드영역이 64Kbyte이하인 near model 설정 경우 data-page pointer 레지스터로 사용이 된다. 그러나, 비행제어컴퓨터 소프트웨어에서는 far model을 사용하게 되어 컴파일러에서 사용하지 않으면서 임의 목적으로 B14 레지스터를 사용할 수 있었다. 이 레지스터에 브레이크포인트 이벤트 핸들러(breakpoint event handler)의 어드레스를 설정하되 소프트웨어의 부팅(power-up) 초기에 설정하도록 하였다.

상기에서와 같이, 디버그 에이전트는 외부 통합시험 통제 컴퓨터로부터 프로그램 메모리 이벤트 트리거 설정은 그림 10과 같이 수행된다. 즉, 이벤트 발생시킬 어드레스를 입력받아 3개의 소프트웨어 트랩 명령어 세트와의 스왑을 수행한다. 여기서 3개의 트랩 명령어 세트는 스왑된 기존 프로그램 메모리 값은 임의 지정된 곳에 저장되며, 트리거 이벤트가 발생되었을 때 해당 핸들러는 변경된 프로그램 메모리 값을 다시 본래의 값으로 바꾸어 놓는다. 이러한 이벤트 핸들러는 외부 통합시험 통제 컴퓨터로부터 메모리 및 레지스터 읽기/쓰기와 메모리 스냅샷 등의 작업을 수행하게 된다. 또한 통합시험 관리 컴퓨터의 명령을 통해 이벤트 트리거링 없이 언제든지 레지스터 및 메모리 읽기/쓰기를 비행제어 컴퓨터의 실시간 동작에 거의 영향을 주지않고 할 수 있다. TI 6000 계열 프로세서로는 데이터 메모리에 대한 이벤트 트리거링은 불가능하며 PowerPC 계열에서는 가능하다. PowerPC 계열에서는 하드웨어적으로 프로그램 메모리와 데이터 메모리에 대해 컴패레이터(comparator) 레지스터를 가지고 있다. 그러나, 시스템 통합시험의 테스트 케이스들은 복수의 조합 이벤트 트리거링(combined event triggering)이 필요한 경우도 있으므로 여전히 디버그 에이전트에서 소프트웨어적으로 구현해야 한다.

상기와 같은 디버그 에이전트 기능이 간이로 구현이 되었으나, 실제 비행제어 컴퓨터 통합시험단계에 까지는 사용되지는 않았으며, 적용 가능성만 데모로 확인하였다. PI장비를 사용하지 않고, 디버그 에이전트 만을 사용해서 세 채널 독립의 비행제어 컴퓨터 시스템 통합시험을 수행하려면 시험방법론적인 변경이 필요하다. 독립된 세 채널에 대해 sync-run/halt 기능 그리고 master/slave 브레이크포인트기능과 기존 PI의 쉐도우 램을 통한 FLCC 실시간 처리에 영향 없이 데이터 전송에 대하여 시험 방법론적 변경이 필요하다. Sync-run/halt와 master/slave 브레이크포인트의 경우 Cross-Channel로 연결된 hardwired 신호와 memory-memory discrete신호를 통해 구현이 가능하다. 실시간 처리에 영향 없이 데이터전송을 위해서는 해당 이벤트 발생시 내부메모리 저장 후 IDLE task 수행시 data pumping을 통해 구현될 수 있다.

III. 결론

본 논문에서는 기존 비행제어 컴퓨터 통합시험을 위해 사용되어 왔던 프로세서 모니터링 장비에 대하여 비행제어컴

퓨터의 최근의 프로세서 채택에 따른 문제점을 분석하였으며 이에 대한 해결 방법들을 보였다. 비행제어 컴퓨터 통합시험에서 FLCC 프로세서와 직접 하드웨어 적으로 물리는 기존 방식의 PI장비가 최근의 고속 프로세서에 적용될 경우, pre-fetch에 의한 잘못된 브레이크 포인트 이벤트와 8-set fetch packet 방식에 의한 중복 브레이크 포인트 이벤트, 그리고 브레이크 포인트 이벤트 시점과 실제 파이프라인 Execute시점의 차이의 문제들이 야기되었다. 각각의 분석 및 해결 방안이 제시되었으며 일부는 실제 초음속 항공기 비행제어 시스템통합시험에 적용되었다.

본 문제들의 근본 원인은 최근의 프로세서에 적용되는 멀티패칭(multi-fetching) 방식의 프로세서 사용에 따른 기존 PI 장비의 프로세서 외부버스 모니터링 방식에 있다. 계속적인 프로세서의 고속화 및 증대된 실시간 처리량의 증대의 요구에 따라, 기존 PI를 이용한 시스템통합시험은 어려울 수 밖에 없게 되었으며 새로운 프로세서 모니터링 방법이 필요하게 되었다. 세 번째 문제점에서 제시된 디버그 에이전트를 통한 프로세서 모니터링 방법을 통해 상기 언급된 전체 문제들이 해결될 수 있으며, 향후 비행제어 시스템 개발에서 적용 검토 중이다.

참고문헌

[1] J. E. Tomayk, "A history of NASA's pioneering digital fly-by-wire project," NASA SP-2000-4224, 2000.

[2] 이철, 서준호, 함흥빈, 조인제, 윤형식 "비행제어컴퓨터의 throughput 향상 및 power-interruption 대처 설계" 한국항공우주학회 논문지, 제 35 권, 제 10 호, pp. 940-947.

[3] "TMS320C4x user's guide," SPRU063C, Texas Instruments, May 1999.

[4] J. Blosser, "*Processor Selection Criteria*" BAE Off-Set Training Material, 2007.

[5] I. Bate, P. Conmy, T. Kelly, J. McDermid, "Use of modern processors in safety-critical applications," *Computer Journal*, no. 6, pp. 531-543, 2001.

[6] I. Bate, P. Conmy, J. McDermid, "Generating evidence for certification of modern processors for use," *in 5th International High Assurance Systems Engineering Symposium*, 2000.

[7] "Diverging Architectures for Digital Signal Processing," BDTI Articles and Presentations, Berkeley Design Technology, Inc. (BDTI), September, 1998.

[8] "Choosing The Right DSP For Real-Time Embedded Systems," *Electronic Design*, November 2000.

[9] "External Memory Types for C6000 Systems," HUNT ENGINEERING Technical Paper, October 2004.

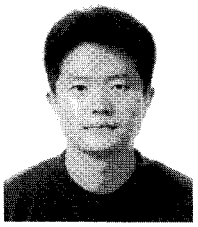
[10] "TMS320C6000 Assembly Language Tools User's Guide," SPRU186P, Texas Instruments, July 2005.

[11] "TMS320C67x/C67x+ DSP CPU and Instruction Set Reference Guide," SPRU733A, Texas Instruments, November 2006.

[12] K. S. Kong, etc., "A Design and implementation of debug agent of debug agent for remote debugging of embedded real-time software," *The 8th IASTED International Conference on Applied Informatics-AI2000*, February 2000.

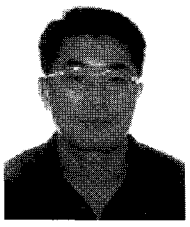
[13] J. G. Ganssle, "Debuggers for modern embedded systems," *Embedded Systems Programming*, November 1998.

[14] "Code Composer Studio v3.0 Getting Started Guide," SPRU509E, Texas Instruments, September 2004.



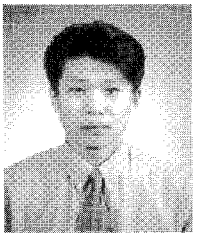
이 철

1998년 금오공과대학교 대학원 기전공학과 석사. 현재 한국항공우주산업㈜ 비행역학팀 선임연구원. 관심분야는 신호처리, 실시간 제어시스템, 진동/소음 제어 및 Embedded System.



조 인 제

1984년 경북대 전자과 졸업. 1986년 동 대학원 석사. 현재 한국항공우주산업㈜ 비행역학팀 수석연구원. 관심분야는 안전중시 소프트웨어 설계/검증.



김 재 철

2001년 부산대 전기공학과 대학원 석사. 현재 한국항공우주산업㈜ 비행역학팀 선임연구원. 관심분야는 Embedded System & Software, AI Control, Robotics.