논문 2005-42SD-12-12

루프 검출 및 예측 방법을 적용한 비용 효율적인 실시간 분기 흐름 검사 기법

(A Cost-effective Control Flow Checking using Loop Detection and Prediction)

김 근 배*, 안 진 호*, 강 성 호*

(Gunbae Kim, Jin-Ho Ahn, and Sungho Kang)

요 약

최근의 저 전력 컴퓨터 시스템은 내장 프로세서의 성능 향상과 공정 기술의 발전을 통한 디바이스 크기 감소로 인해 전압 변동, 커플링 효과 등으로 인한 SEU(single event upset)로 모델링 되는 천이고장으로 인한 예기치 못한 동작 중 에러 발생 가능성이 매우 높아지고 있다. 제안하는 방식은 프로세서가 처리하는 프로그램 분기 흐름상에서 에러를 검출하는 효과적인 watchdog 프로세서 구조로서, 기존 방식이 가지는 오버헤드를 줄이면서 프로그램 내부에서 빈번히 발생되는 루프를 매번 검사할 때, 동일한 동작을 watchdog 프로세서가 반복함으로써 생기는 비효율적인 메모리 접근, 버스 점유 경쟁등과 같은 추가적인 시스템 수준의 오버헤드를 줄이는 새로운 방법을 제안하였다. 본 논문은 기존의 실시간 분기 및 제어 흐름 연구에서는 다루지 않았던 루프 검출 및 예측 기능을 추가함으로써 실제 시스템 적용에 보다 적합한 비용 효율적인 구조를 제안하고 있다.

Abstract

Recently, concurrent error detection for the processor becomes important. But it imposes too much overhead to adopt concurrent error detection capability on the system. In this paper, a new approach to resolve the problems of concurrent error detection is proposed. A loop detection scheme is introduced to reduce the repetitive loop iteration and memory access. To reduce the memory overhead, an offset to calculate the target address of branching node is proposed. Performance evaluation shows that the new architecture has lower memory overhead and frequency of memory access than previous works. In addition, the new architecture provides the same error coverage and requires nearly constant memory size regardless of the size of the application program. Consequently, the proposed architecture can be used as an cost effective method to detect control flow errors in the commercial off the shelf products.

Keywords: On-line test, Watchdog processor, Transient fault, Loop detection, Control flow error

I. 서 론

최근의 마이크로프로세서 디자인의 경향은 빠른 동작 속도와 보다 작은 면적과 적은 전력을 소비하는 방향으로 나아가고 있다. 특히 HHP(hand-held product)의 경우, 내장되는 프로세서의 성능 향상과 공정 기술의 발전을 통한 디바이스 크기 감소를 요구하고 있고 저 전력을 위한 낮은 동작 전압과 성능 유지를 위한 높은 동작 속도를 유지하여야 하므로 전압 변동, 커플링 효과 등의 문제가 보다 심각하게 되었으며, 그로 인하여 SEU(single event upset)로 모델링 되는 천이고장으로 인한 예기치 못한 동작 중 에러 발생 가능성이 매우 높아지게 되었다[1].

천이 고장(transient fault)은 짧은 시간동안 회로 내부의 전류, 전압을 급격히 증가시킨 뒤 사라짐으로써 순간적으로 프로세서의 명령어 주소, 제어 신호, 상태

* 정회원, 연세대학교 전기전자공학과
(Yonsei University, Department of Electrical and Electronic Engineering)
접수일자: 2005년5월16일, 수정완료일: 2005년11월28일

플래그 등의 값들을 변동시키거나 데이터의 값을 잘못된 값으로 변화시킨다[1][2]. 천이 고장이 프로세서에 미치는 영향에 대해서 크게 프로세서에서 발생 가능한 에러를 데이터 에러와 잘못된 분기 에러 두 가지로 분류되고, 실제 천이 고장을 유발하는 대이온 (heavy ion) 삽입 시 분기 에러와 데이터 에러가 발생하는 빈도를 조사한 결과, 실제 데이터 에러보다는 분기 흐름 및 제어 신호에서 발생되는 에러가 전체 동작 중 발생되는 에러에서 무려 96%를 차지한다는 연구 결과를 보였으며, 순수하게 데이터만의 에러가 발생되는 경우는 불과 4%에 불과하다는 결과를 보이고 있다[2]. 따라서 신뢰도 대비 오버헤드의 관계로 볼 때, 분기 흐름 에러 검사에 초점을 맞추는 것이 합리적인 방법이라 할 수 있다.

이런 분기 흐름 에러를 검출하는 방안들은 정상 동작 시의 정보인 기준 정보(reference information)를 저장하고 이를 사용하여 하드웨어적 방법으로 검사를 수행하거나, 응용 프로그램 내에 실시간 분기 흐름 검사를 위한 코드를 삽입하여 소프트웨어적인 방법으로 나뉘어진다[3-13]. 그러나 이러한 기존 연구들은 분기 흐름 에러 검사에 사용되는 오버헤드만을 고려하고 있다. SPEC benchmark를 이용하여 응용 프로그램 루프 특성(loop characteristics)을 조사한 결과, 루프 회수는 대략 80개, 사용된 루프의 중첩 깊이는 대략 평균 2의 값을 가지며 프로그램 내부에서 발생하는 루프의 평균적인 반복 회수는 대략 200회 이하에서 88%정도가 발생하고 있음에서 보이고 있다[14][15][16]. 이처럼, 응용 프로그램 검사 시 루프와 같은 반복적인 동작에 대한 고려가 매우 중요함에도, 이에 대한 고려가 전혀 되어있지 않기 때문에, 실시간 검사 시 시스템 모듈과의 버스 점유율 경쟁으로 인한 추가적인 오버헤드를 유발하게 되어 실제 시스템 적용 시 큰 문제를 야기하게 된다.

따라서 본 논문은 실시간 분기 흐름 에러를 검사하면서 하드웨어 오버헤드를 최소화하고, 루프를 검출 및 예측하는 새로운 방법을 제안, 시스템 모듈 간 버스 점유율 경쟁을 최소화하여 실제 시스템에 효과적으로 적용이 가능한 방법을 제안하고자 한다.

II. 제안하는 분기 흐름 검사 기법

본 논문에서는 WDP[13] 방식을 기반으로 한 새로운 루프 검출 기반 watchdog 프로세서(WLD, Watchdog processor with Loop Detection)를 제안하고자 한다. WLD는 주 프로세서를 실시간 검사하는 보조 프로세서로서 프로그램의 노드 종류에 따라 정의된 OP-code에 따라 동작하며, 루프 검출 및 예측을 위한 추가 하드웨어가 들어가게 된다. OP-code와 노드 타입별 동작, 그리고 루프 검출 및 예측 관련 동작은 다음과 같다.

1. 노드 타입과 동작

본 논문에서 정의하는 각 노드 타입에서 수행하도록 기대되는 동작을 설명하기로 한다. 응용 프로그램 내에 존재 가능한 노드들의 종류와 OPCODE를 다음과 같이 각각 8가지 타입과 3 bit로 정의할 수가 있다.

가. 시작 노드(000, starting node)

그림 1에서와 같이 응용 프로그램이 처음 시작되는 부분에 해당한다. 이 노드와 연관된 주소가 프로세서로부터 도착하고 기준 주소 값과 일치하면 프로그램의 시작에 해당하고, 바로 다음 노드로 진행하기 위해서 현재 메모리 주소에 1 증가한 주소를 접근하여 다음 검사 동작을 준비한다. 그렇지 않을 경우 에러 신호를 프로세서로 보낸다.

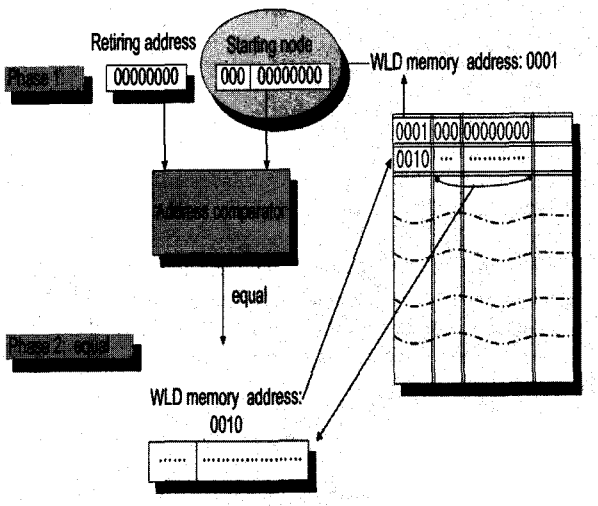


그림 1. 시작 노드의 예상 동작
Fig. 1. Operation of starting node.

나. 진행 노드(001, proceeding node)

그림 2와 그림 3과 같이 진행 노드는 주소가 불연속, 연속적일 경우 모두에서 발생할 수 있다. 두 가지 모두, 이 노드와 연관된 주소가 도착하고 기준 주소 값과 일치하면 바로 다음 노드로 진행하는 역할을 하며 현재 주소에서 1 증가한 주소로 메모리를 접근, 다음 검사를 준비한다. 그렇지 않을 경우, 에러 신호를 프로세서로 보낸다.

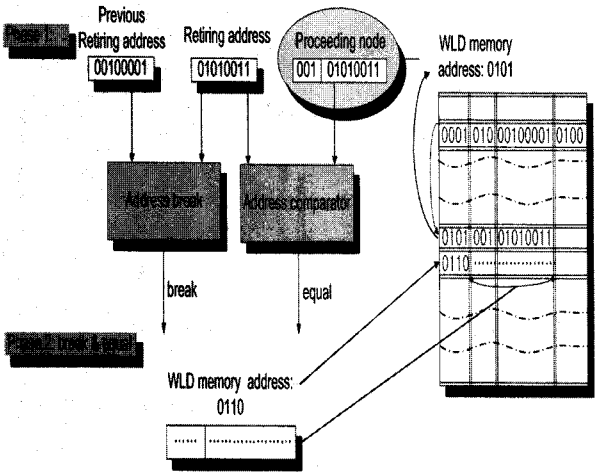


그림 2. 주소 불연속 뒤 진행 노드 도착 시 예상 동작
Fig. 2. Operation of proceeding node at address break.

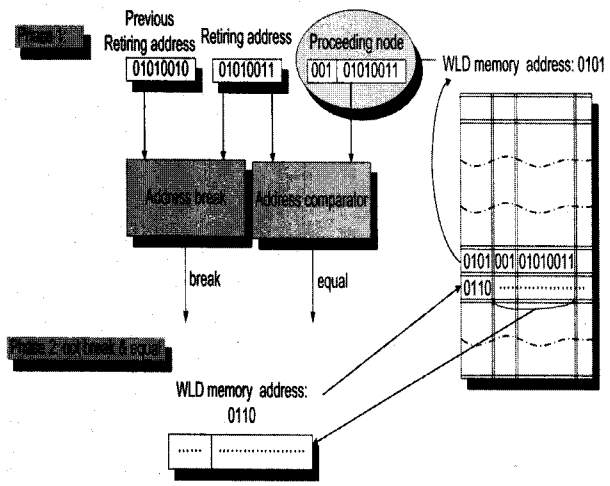


그림 3. 주소 연속 뒤 진행 노드 도착 시 예상 동작
Fig. 3. Operation of proceeding node at no address break.

다. 무조건 전방 분기 노드

(010, unconditional, forward branch)

그림 4와 같이 이 노드와 연관된 주소가 프로세서에서 도착하면 프로세서는 다음 동작으로 무조건 분기 대상이 되는 주소로 분기를 하기 때문에 이를 실시간으로 검사하기 위해서 현재 노드의 기준 정보 메모리 주소에 분기 주소 간격을 더하여 분기가 예상되는 기준 정보 메모리를 접근하여 해당 기준 정보를 가져온 뒤에 다음 비교 검사 동작을 준비한다. 만일 다음 비교 검사 수행 시에 기준 정보 내의 연관된 노드 주소와 현재 수행이 끝난 주소와 일치하지 않는다면, 바로 에러 신호를 보내지 않고 주소 불연속 여부를 본 뒤에 만일 주소가 연속적이라면 이전 기준 정보 메모리 주소를 1 증가시켜서 기준 정보를 다시 가져온 뒤에 기준 정보 내 주소

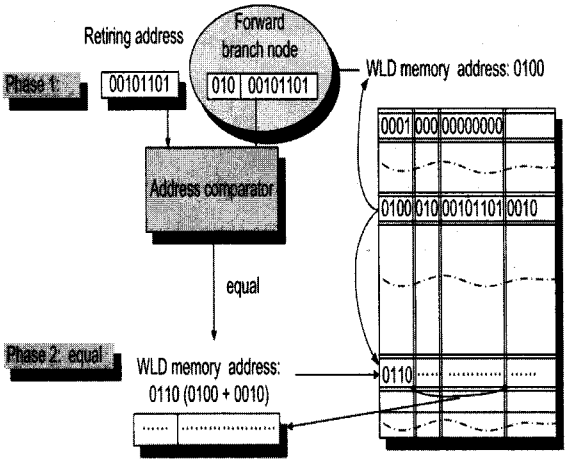


그림 4. 전방 분기 노드의 예상 동작
Fig. 4. Operation of forward branching node.

값과 다시 비교 동작을 수행하며 여기서도 일치하지 않을 때만, 에러 신호를 내보낸다.

라. 조건 전방 분기 노드

(011, conditional, forward branch)

그림 4와 같이 이 노드는 프로그램 내의 노드 타입은 다르나 무조건 전방 분기와 같은 동작을 수행한다.

마. 무조건 후방 분기 노드

(100, unconditional, backward branch)

그림 5와 같이 이 노드의 동작은 무조건 전방 분기 노드와 유사하나 후방 분기는 루프를 형성하므로 현재 가져온 기준 정보를 저장하고 다음 분기 주소의 정보를 가져오기 위하여 앞서 계산한 분기 주소로 기준 정보 메모리를 접근한다. 이 때 접근하는 기준 정보 메모리

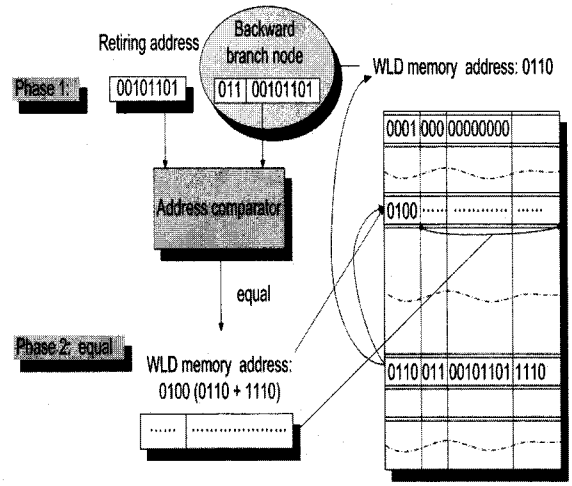


그림 5. 후방 분기 노드의 예상 동작
Fig. 5. operation of backward branching node.

의 주소가 루프 머리에 해당하며 이 주소의 기준 정보를 가져올 때 이 정보 역시 따로 저장하여 다음 루프 반복 시 예측을 위하여 사용한다. 또한 다음 반복 시에 현재 노드를 예측하기 위하여 이전 기준 정보 메모리 주소를 계산하고 저장하며 루프를 벗어나는 경우를 대비하여 다음 기준 정보 메모리 주소를 계산하고 저장한다. 만일 다음 노드에서 기준 정보의 주소와 불일치가 발생되면 이전에 저장된 다음 기준 정보 메모리 주소를 접근하고 기준 정보를 가져온 뒤, 그 중에서 기준 주소 값과 다시 비교를 수행하여 일치하면 에러 신호를 보내지 않고 그렇지 않을 경우는 에러 신호를 프로세서로 보낸다.

바. 조건 후방 분기 노드

(101, conditional, backward branch)

그림 5에서와 같이 이 노드는 프로그램 내의 노드 타입은 틀리지만 무조건 후방 분기와 동일한 동작을 수행한다.

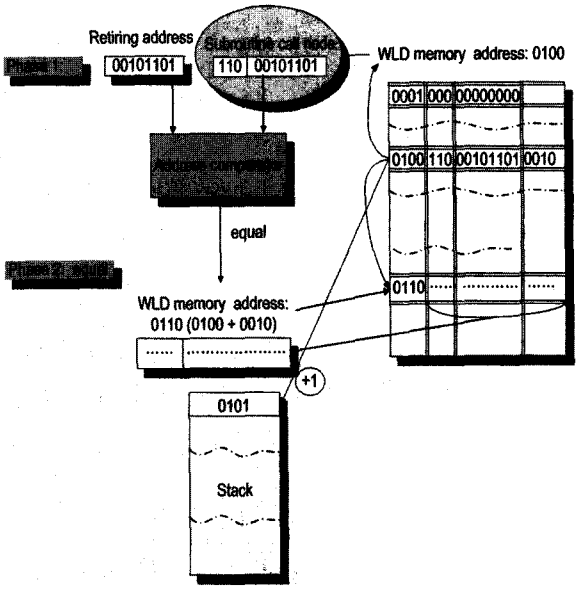


그림 6. 서브루틴 호출 노드의 예상 동작
Fig. 6. Operation of subroutine call node.

사. 서브루틴 호출 노드(110, subroutine call node)

그림 6과 같이 이 노드와 연관된 주소가 프로세서에서 도착하고 기준 정보 주소와 일치하면, 현재 기준 정보 메모리에서 가져온 기준 정보 중 분기 주소 간격과 현재 메모리 주소를 더하여 서브루틴에 해당하는 기준 정보 메모리 주소를 계산하고 이 주소를 접근하여 다음 비교 동작을 준비한다. 그리고 현재 기준 정보 메모리

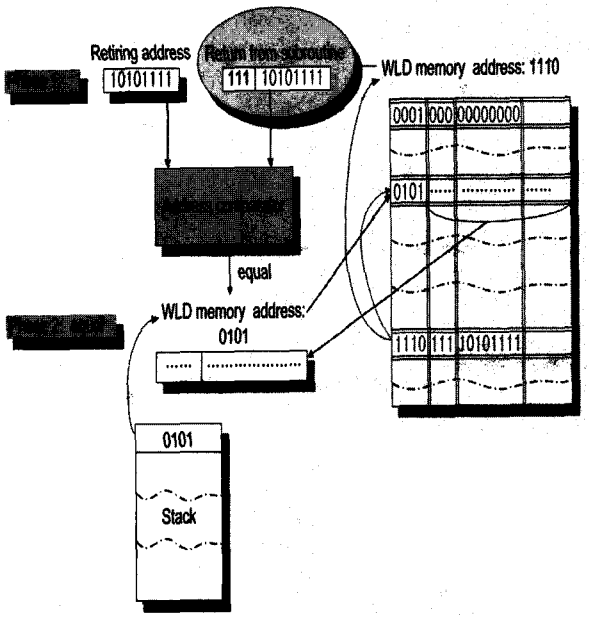


그림 7. 서브루틴 복귀 노드의 예상 동작
Fig. 7. Operation of return from subroutine node.

에 1 증가하여 서브루틴 복귀 시를 위하여 스택에 이 값을 저장한다. 그렇지 않고 주소 비교가 불일치하면 에러 신호를 프로세서로 내보낸다.

아. 서브루틴 복귀노드

(111, return from subroutine node)

그림 7과 같이 이 노드와 연관된 주소가 프로세서에서 도착하고 기준 정보 주소와 일치하면, 현재 스택에 저장되어 있는 값을 꺼내서 이 주소 값이 가리키는 주소로 기준 정보 메모리를 접근하여 다음 비교 검사를 준비한다. 그렇지 않을 경우, 에러 신호를 프로세서로 보낸다.

2. 루프 타입과 동작

실시간 분기 흐름 검출방식에 루프 검출 방식을 사용하기 위해서는 먼저 응용 프로그램에서 존재하는 여러 가지 루프의 타입을 정의할 필요가 있다. 그림 8은 발생 가능한 각각의 루프 타입을 정의한 것이다.

각 루프 타입은 독립적으로 사용되거나 두 가지 이상의 타입이 혼합되어 또 다른 루프를 형성할 수도 있다. 본 논문에서는 서로 독립적으로 정의 가능한 루프 타입에 대해서 언급할 것이며, 이를 통하여 이들을 동시에 혼용하여 실험을 수행하였다.

가. 단일 루프(single loop)

다음의 그림 8의 (1)에서와 같이 오직 하나의 루프

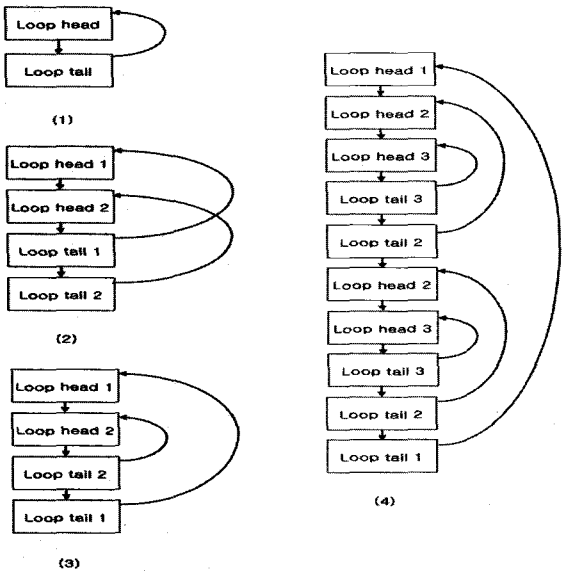


그림 8. 루프 타입
Fig. 8. Types of loops.

머리와 루프 끝으로만 구성된 루프를 단일 루프라고 정의한다. 단일 루프는 가장 기본적인 루프의 형태로서 교차 루프와 중첩 루프를 구성하는 원소로도 사용된다.

나. 교차 루프(overlapped loop)

교차 루프는 그림 8의 (2)에서 하나의 단일 루프의 루프 몸체 내부에 또 다른 루프의 루프 머리 또는 끝이 들어 있는 형태이다. 그림 8의 (2)는 두 개의 단일 루프가 중첩되어 하나의 교차 루프를 형성하고 있지만, 하나의 교차 루프가 다른 단일 루프와 중첩되어 또 다른 교차 루프를 형성할 수도 있다.

다. 단순 중첩 루프(purely nested loop)

그림 8의 (3)에서 하나의 단일 루프 내에 또 다른 단일 루프가 완전히 포함되는 경우를 중첩 루프라 정의한다. 중첩 루프에서 가장 바깥을 감싸고 있는 루프를 외각 루프(outermost loop), 가장 내부에 있는 루프는 내각 루프(innermost loop)라고 정의한다. 이중 특별히 내각 루프와 외각 루프 두 개의 루프로만 구성된 중첩 루프를 이중 중첩 루프(doubly nested loop)라고 정의하며, 외각 루프와 내각 루프 사이에 하나 이상의 루프가 중첩 될 때마다 중첩 루프의 중첩 깊이는 2 이상의 값을 가지게 된다.

라. 혼성 중첩 루프(mixed nested loop)

혼성 중첩 루프는 그림 8의 (4)에서 보는 바와 같이 외각 루프 내에 중첩 루프와 함께 또 다른 타입의 루프가 포함되어 있는 경우를 말한다. 외각 루프는 중첩 루프이외에 단일 루프, 교차 루프 그리고 또 다른 중첩 루프를 포함 할 수 있으며 혹은 중첩 루프와 단일 혹은 교차 루프가 다시 교차될 수도 있다.

WLD가 반복적인 루프를 수행하는 것을 막고 기존에 저장하고 있는 정보를 최대한 활용하여 루프를 미리 예측하고 메모리 접근을 줄이도록 하기 위해서는 루프의 출발은 루프 끝에서 분기가 발생될 때 시작 된다는 사실을 활용하여야 한다. 즉, 첫 번째 루프 수행 시에 루프의 끝을 지나면서 루프가 형성되게 되면 현재 루프 끝에 해당하는 노드의 기준 정보들 중 노드 타입, 노드와 연관된 기준 주소를 저장하면서, 분기 주소 간격을 현재 기준 정보 메모리의 주소 값과 더한 분기 주소를 계산하여 함께 레지스터에 따로 저장해 둔다.

이와 동시에 현재 기준 정보 메모리 주소에서 하나 감소한 주소 값을 이전 노드 주소(previous node address)로 따로 저장하여 가지고 있게 된다. 동시에 실제 분기가 일어나지 않고 루프 끝의 바로 다음 노드로 진행 할 경우 발생되는 잘못된 예측을 막기 위하여 현재 기준 정보 메모리 주소에 1 증가시킨 다음 노드 주소(next node address)도 함께 저장한다. 즉 하나의 루프에 대해서 가져야 할 정보는 이전, 다음 노드 주소와 루프 끝의 기준 정보들 및 루프 끝의 분기 주소인 루프 머리의 기준 정보 값, 총 4 가지의 정보가 저장되어야 예측이 가능하게 된다. 따라서 루프의 중첩 깊이가 증가되고 이중 중첩 루프보다 더 중첩 깊이가 깊은 루프를 다루기 위해서는 루프 하나가 더 중첩 될 때마다 4개의 저장 공간에 대한 오버헤드와 중첩 루프 간의 taken 혹은 not taken일 경우를 다루기 위한 하드웨어의 복잡도가 기하급수적으로 커진다는 단점이 생기게 된다.

그러나 루프의 중첩 깊이는 다양한 값을 가지지만 실제로 중첩된 내부의 루프들이 모두 쓰이는 것은 아니고, 중첩 깊이가 깊다 하더라도 실제 사용된 루프의 사용량을 이용하여 가중치를 부여, 계산한 중첩 깊이는 대략 2에 해당한다[15]. 따라서 본 논문은 이중 중첩 루프까지만 예측 가능한 구조를 제안한다. 또 이중 중첩 이상의 경우, 먼저 검출된 두 개의 루프 외의 다른 중첩 루프는 루프를 예측하지 않고 메모리 접근을 매번 수행하는 구조를 적용하였다.

중첩된 루프가 taken 되거나 not taken 되는가의 여부는 프로세서가 결정하기 때문에 루프의 중첩 순서에 따라서 반드시 루프가 형성된다고 단정할 수 없다. 그

러나 일단 외각 루프에 대한 정보를 모두 저장하고 나서 내부의 중첩 루프는 해당 루프가 반복 수행이 끝날 경우 저장 정보를 삭제함으로써 다른 중첩 루프가 수행될 때 분기 예측이 가능하도록 하였다. 이 구조는 단순 혹은 혼성 중첩 루프 모두에 일관되게 적용되며 외각 루프가 일단 검출 되면 외각 루프 내에서 중첩 깊이나 교차 여부에 상관없이 not taken된 루프는 무시하고 taken 된 루프만을 저장하도록 하였으므로 하드웨어의 복잡 도를 줄일 뿐만 아니라 모든 중첩 루프 타입에 대해서 일관된 동작과 단순한 구조를 제공할 수 있다는 장점을 가진다.

3. 제안된 루프 검출 기반 실시간 분기 흐름 검사 하드웨어

WLD는 CPM(Central Processing Module), LDM(Loop Detection Module), RPM(Retiring address Processing Module)의 세 가지 모듈로 구성되어 있다. 그림 9는 세 가지 모듈 전체를 통합한 구조이다

가. RPM

그림 10의 RPM은 현재 수행이 끝난 프로세서의 명령어들을 매 프로세서의 동작 사이클 마다 받으며 이전 사이클에서 수행이 끝났던 명령어의 주소를 저장해 놓은 뒤, 수행이 끝난 명령의 주소가 도착되면 주소 값을 하나 증가 시켜서 현재 수행이 끝난 명령어와 일치 여부를 비교한다. address_break 신호와 현재 수행이 끝난 명령어의 주소 값은 분기 흐름 검사에 사용되기 위하여 CPM으로 넘겨지게 된다.

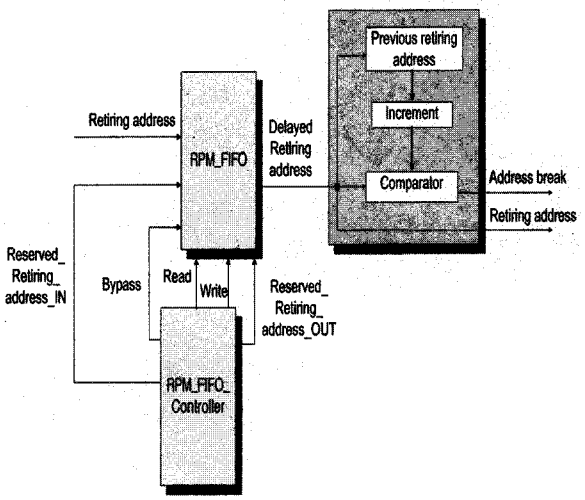


그림 10. RPM의 구조
Fig. 10. RPM.

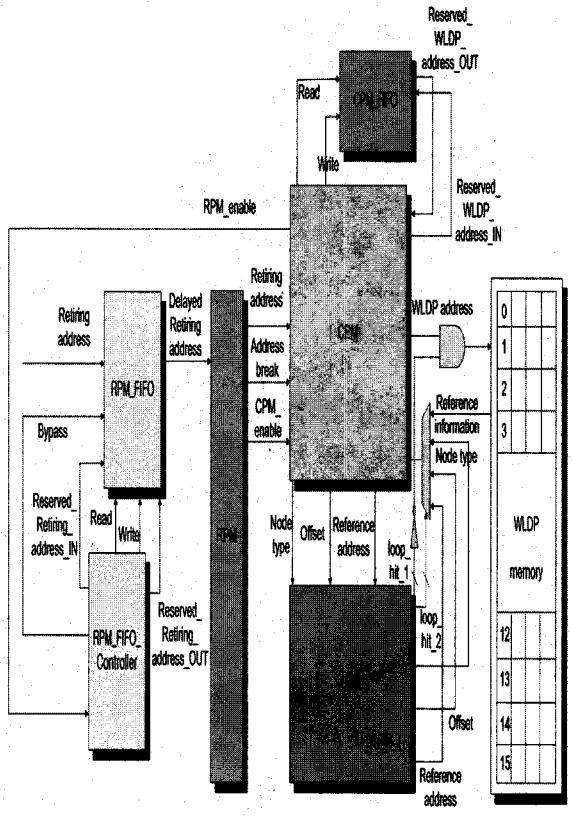


그림 9. WLD의 구조
Fig. 9. Proposed architecture of the WLD.

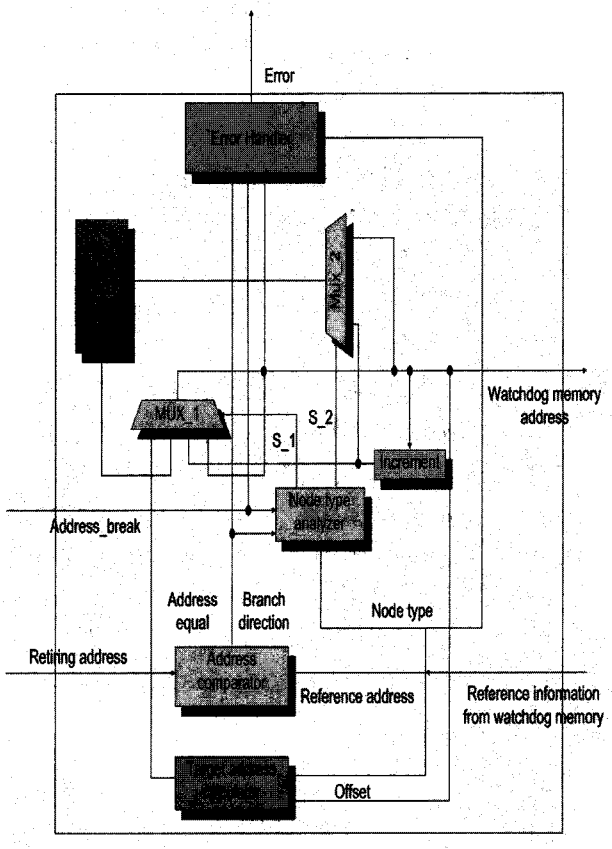


그림 11. CPM의 구조
Fig. 11. CPM.

나. CPM

RPM에서 address_break 신호와 현재 수행이 끝난 명령어의 주소를 받으면, 그림 11의 CPM은 미리 기준 정보 메모리에서 가져온 뒤에 비트 필드별로 분리해 놓은 기준 정보들을 사용하여 분기 흐름을 검사하게 된다. 우선 각 모듈별 동작을 기술하면 다음과 같다.

(1) Address comparator

RPM이 보내는 수행이 끝난 명령어 주소와 기준 정보 중에서 연관된 노드 주소를 비교하여 일치 하는지 비교한다.

(2) Target address calculator

분기 주소 간격과 현재 기준 정보 메모리를 접근하는 주소를 입력으로 받아서 이를 더한 값을 출력으로 내보내는 역할을 한다.

(3) Node type analyzer

노드 타입과 address_break, address_equal 신호 값을 입력으로 받아 MUX_1, MUX_2의 선택 신호들을 출력으로 내보내주게 된다. MUX_1은 다음 기준 정보 메모리를 접근하기 위한 주소 값을 선택하는 역할을 한다. MUX_2는 노드 타입이 서브루틴 호출 노드일 경우, 스택(FIFO)에 현재의 기준 정보 메모리 접근 주소 값 또는 1을 더하여 만든 주소 값을 호출 뒤에 복귀하기 위한 복귀 주소로 저장하기 위하여 사용되었다. 가능한 메모리 접근 주소 계산 방법에 대해서 모두를 한꺼번에 생성한 뒤에 MUX_1, MUX_2의 값에 따라서 그 중 한 가지에 해당하는 주소 값에 따라서 기준 정보 메모리를 접근하며 서브루틴 호출일 경우는 동시에 스택에 값을 저장한다.

(4) Error handler

address_break, address_equal을 입력으로 받아 각 노드 타입에 따라서 이들 신호의 조합을 가지고 에러 여부를 판별한다. address_equal 신호가 0일 경우는 주소 값의 불일치를 나타내는 것이며 이때는 기본적으로 주소 에러를 가정할 수 있으나 분기 예측의 잘못된 예측 결과로도 볼 수 있으므로 이때 동시에 address_break 신호와 노드 타입 별 성격에 따라서 에러를 판단한다.

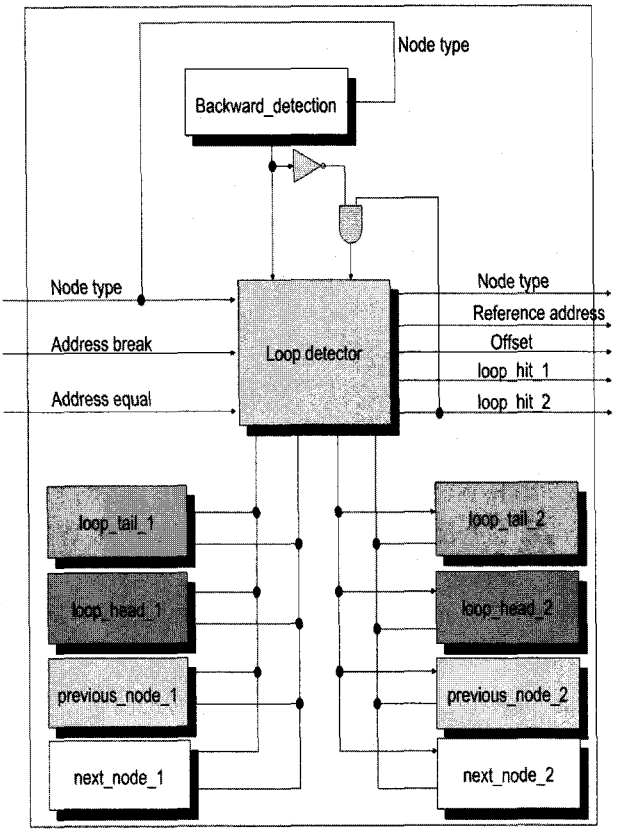


그림 12. LDM의 구조
Fig. 12. LDM.

다. LDM

그림 12는 LDM의 구조를 나타낸 것이며 loop decision maker의 동작을 기술하면 다음과 같다. 후방 분기 인지 여부를 backward detection 모듈을 통해서 감지한 뒤 만일 후방 분기이면 loop_hit_1을 1로 만들고 기준 정보를 loop_tail_1, previous_node_1, next_node_1에 저장한 뒤 루프 머리를 접근하여 loop_head_1에 저장한다. 만일 loop_hit_1에서 저장된 루프 끝이 not taken 되거나 주소 에러가 발생하면 일단 next_node_1에 저장된 주소를 다시 접근하여 기준 정보를 가져온 뒤에 다시 비교를 수행하여 에러 여부를 판별한다. 만일 현재 loop_hit_1이 1인 상태에서 다시 후방 분기가 도착하면, loop_hit_2를 1로 만들고, loop_tail_2, previous_node_2, next_node_2에 정보를 저장하고 루프 머리를 접근하여 loop_head_2에 기준 정보를 저장한다.

III. 실험 결과

1. WDP와 WLD의 에러 검출률 비교

표 1은 제안하는 구조의 하드웨어 구현 결과를 보여

표 1. 하드웨어 구현 결과
Table 1. Hardware implementation results.

CPM, LDM, RPM	622	16	61.77
Memory interface	24	16	2.38
CPM_FIFO	76	8	7.55
RPM_FIFO	285	8	28.30
Total	1007	48	100

주고 있다. WLD는 100M gate, 4M logic cell의 FPGA development board에서 구현되었다. 간단하게 WLD의 특성을 정리해 보면 다음과 같다.

- 16bit-embedded memory, memory interface
- 8bit-microprocessor address processing module
- 16bit-loop detection, central processing module
- Dedicated retiring address line of target processor
- Synchronous operation with target processor (50Mhz)

실험을 위한 대상 프로세서에는 간단한 패킷 처리를 위한 프로세서가 사용되었다. 해당 프로세서의 특성을 간단히 정리하면 다음과 같다.

- 4-stage pipeline architecture
- 32bit instruction set and 32bit data-path
- 32-General Purpose Registers (32bit), 64-read/write Transfer Registers (TRs)

표 1의 결과와 같이, WLD의 총 게이트 수는 전체 FPGA logic cell의 4%에 불과하며 이를 통하여 실제 시스템에 적용될 경우 생길 수 있는 면적 오버헤드로 인한 문제를 해결하였다.

WLD의 실험에 사용된 에러 타입은 3 가지이며 각 타입의 에러에 대해서는 표 2에 설명되어 있다. 삽입된 에러의 개수와 해당 타입 별 에러 검출률은 표 3에서 보는 바와 같다.

표 3에서 보는 바와 같이 각각의 노드 타입에서 발생 가능한 에러에 대해서, WLD는 WDP와 대등한 100%의 에러 검출률을 보임을 알 수 있다.

표 2. 에러 타입 별 관련된 노드 타입과 발현 동작
Table 2. Error types and description.

Type 1	Proceeding node Starting node	Address error at non-branching node
Type 2	Cond., backward branch Cond., forward branch Uncond., backward branch Uncond., forward branch Subroutine call Return from subroutine	Address error making illegal loop
		Address error making illegal forward branch
		Address error making illegal subroutine call
		Address error making illegal return from subroutine
		Address error making illegal branch from the inside of the loop to the outside
Type 3	All types	Address error making illegal branch form the inside of node to the inside or the outside

표 3. 에러 타입 별 에러 개수 및 에러 검출률
Table 3. Error coverage.

Type 1	000	16	100%	100%
	001	137	100%	100%
	010	4	100%	100%
Type 2	011	129	100%	100%
	100	1	100%	100%
	101	1	100%	100%
Type 3	110	2	100%	100%
	111	2	100%	100%
	All nodes	1044	100%	100%

2. 루프 검출률

표 4는 각 루프 타입 별 루프 개수 및 루프 검출 여부와 검출률, 그리고 루프에서 삽입된 타입 별 에러에 대한 에러 검출률을 나타내고 있다.

표 4를 통해서 루프 검출 및 예측 방식이 추가된 구조가 에러 검출률에 영향을 미치지 않을 뿐더러 삼중 중첩 루프의 경우 가장 내부의 루프에 대해서 루프 검출 및 예측 동작만을 수행하지 않을 뿐, 실시간 분기 흐름을 검사하는 것에는 영향을 미치지 않고 있음을 알 수 있다.

표 4. 타입 별 루프 검출률 및 에러 검출률
Table 4. Loop detection coverage.

Loop type	Loop number	Loop detection	Loop detection coverage	Error coverage of each error type		
				Type 1	Type 2	Type 3
Single	10	Yes	100%	100%	100%	100%
Overlapped	2	Yes	100%	100%	100%	100%
Doubly nested	65	Yes	100%	100%	100%	100%
Tripple nested	34	Only two	99%	100%	100%	100%
Total	111	-	99.75%	100%	100%	100%

3. WDP와 WLD의 메모리 접근 비교

루프 검출 성능에 대한 검증과 더불어 루프 검출을 통해 얻고자 하는 메모리 접근 감소가 실제 어느 정도인지 실험하였다. 실험 대상 루프 타입은 루프 검출률 계산에 사용된 루프 타입과 동일하게 적용하였으며, 표 5는 이러한 루프 타입에 대해서 총 반복 회수 중 메모리 접근이 차단되는 회수를 루프 머리, 끝에 대해서 첫 번째 반복과 두 번째 반복 이후로 구분하여 정리하였다. 표 5에서와 같이 WDP는 루프 머리와 루프 끝 부분에서 총 1138번의 반복을 모두 수행하여야만 하지만 WLD는 첫 회에서, 예측을 위한 기준 정보 저장에 필요한 총 142번의 루프 머리, 끝의 메모리 접근을 제외하고는 996번의 반복된 루프 머리, 끝의 접근을 차단함을 알 수 있으며, 더불어 마지막 반복이 끝난 뒤에, 루프 끝 부분의 추가적인 예측을 통해서 접근이 차단되므로

표 5. 루프 타입 별 WDP 대비 메모리 접근 감소 회수
Table 5. Analysis for memory access reduction.

		Number of loop iterations	Total loop iterations	Total loop heads	First iteration: Initial access			Second final iteration: Reduced access			
					Tail	Head	Iteration	Tail	Head	Iteration	%
Single		100	100	100	4	4	4	96 +4	96	96	98
overlapped		27	27	27	6	6	6	21 +6	21	21	92.3
Doubly nested	Outer	15	15	15	3	3	3	12 +3	12	12	90
	Inner	60	60	60	5	5	5	55 +5	55	55	95.8
Triply nested	Outer most	13	13	13	5	5	5	8+5	8	8	80.8
	Inner	46	46	46	12	12	12	34 +12	34	34	86.9
	Inner most	308	308	308	36	36	36	272 +36	272	272	94.2
Total		569	569	569	71	71	71	569	498	498	91.1

추가로 71번의 루프 끝 부분의 접근을 예측하여 평균 루프 머리와 끝의 메모리 접근에서 91%의 접근 감소율을 얻을 수 있음을 알 수 있다. 이러한 이점은 루프의 반복 회수가 커지면 커질수록 더욱 뚜렷이 나타나게 되는데 표 5에서 루프 반복 회수가 커지면서 메모리 접근 감소율이 향상됨을 알 수 있다.

4. WDP와 WLD의 메모리 오버헤드 비교

표 6은 WLD와 WDP가 사용하는 메모리 크기를 분석한 결과이다.

먼저 WLD의 메모리 크기 증가율과 WDP의 메모리 증가율에 대한 비교에서, WDP는 데이터 비트를 그대로 사용하기 때문에 이것이 메모리 오버헤드에 그대로 반영이 되어, 프로그램의 노드 수가 증가할수록, 즉 프로그램의 크기가 커질수록 선형적으로 메모리의 크기가 증가한다. 그러나 WLD의 경우, 전체 프로그램이 가지는 노드 수 M의 $\log 2^M$에 해당하는 부분만이 분기 주소 간격으로 사용이 되기 때문에 메모리의 크기가 프로그램의 노드 수가 증가하는 것에 비해 큰 변화를 보이지 않고 메모리 크기가 선형적으로 증가하는 것이 아닌 로그 함수를 따르는 분포를 보임을 알 수 있다.

두 번째는 실제 물리적인 메모리 크기 감소이다. WDP와는 달리 WLD는 분기 주소 값을 알기 위하여

표 6. WDP 및 WLD의 메모리 크기 비교
Table 6. Memory overhead of the WDP and WLD.

Number of nodes(M)	Offset	Third field	
		WLD	WDP
100	6.64	2564	3500
200	7.64	2664	7000
300	8.23	2723	10500
400	8.64	2764	14000
500	8.97	2797	17500
600	9.23	2823	21000
700	9.45	2845	24500
800	9.64	2864	28000
900	9.81	2881	31500
1000	9.97	2897	35000
2000	10.96	2996	70000
4000	11.96	3096	140000
6000	12.55	3155	210000
8000	12.96	3196	280000
10000	13.29	3229	350000

데이터 비트가 아닌 분기 주소간의 간격에 해당하는 크기만을 저장하기 때문에, 검사 대상이 되는 프로그램의 노드 개수에 따라서, 분기 주소를 저장하는 세 번째 필드의 메모리 크기가 최소 26.7%에서 최대 99.1%까지 감소하는 결과를 얻을 수 있다.

외와 같이 메모리 크기가 노드 개수의 증가에 대해 큰 영향을 받지 않으므로 다양한 크기의 응용 프로그램에 대해서 적용이 가능하다는 장점을 가지게 된다.

Ⅳ. 결과 분석

제안하는 WLD구조와 기존의 WDP의 공통점은 주소 불연속성(address break)을 통한 분기검출방식을 기반으로 한다는 것이다. 이를 위하여 WDP에서 사용된 응용 프로그램 내 노드 타입, 해당 명령어의 실제 주소, 그리고 해당 명령어가 사용하는 데이터의 압축치(signature)로 이루어진 세 가지의 비트필드(bit field)로 이루어진 instruction 개념을 수용하였다.

그러나 본 구조와의 차이점은 단순히 루프 검출 프로세서만을 추가한 것이 아니라 기존의 WDP의 주소 불연속성에 근거한 하드웨어 구조를 보다 최적화하고 기존 방법이 고려하지 못했던 적용상의 문제점에 대해 고려함으로써 commercial off the shelf processor에 적용 가능한 최적화된 구조를 제안했다는 것이다. 기존 구조와의 차이점은 다음과 같다.

① 서론에서 역설한 바와 같이 데이터 에러의 빈도가 분기 에러보다 현저히 낮다는 점에 착안하여 세 번째 비트 필드인 데이터 압축치 필드를 제거하고 대신 주소 분기 offset을 도입하였고 이를 통하여 분기 노드에 해당하는 명령에 대해서만 offset을 저장하므로 나머지 다른 노드 명령에서는 데이터 압축치 필드 분만큼의 비트 감소가 발생한다. 따라서 전체 에러 발생 빈도 대비 에러 검출률의 최소한의 저하로 기준 정보를 저장하기 위한 내부 메모리의 용량을 현저히 저하시키게 된다.

② WDP에서 사용하는 데이터 압축치와 압축치 생성을 위하여 사용되는 MISR을 사용하지 않는다. 데이터 압축치는 데이터 에러의 실시간 검출뿐만 아니라 분기 노드에서 분기가 예상되는 watchdog 프로그램의 분기 주소(target address)와 exclusive OR을 한 압축치 값을 가지게 된다. 데이터 에러를 검출하기 위하여, 현재 수행이 끝난 명령의 데이터 압축치를 MISR을 이용하여 생성하고 이 값과 저장된 압축치 값을 비교함으로써 데이터 에러를 검출하게 된다. 그러나 이에 사용되는 압축치는 매우 큰 메모리 오버헤드를 요구하며, 의도되지 않은 분기를 검사하기 위한 목적으로는 오직 분기 주소 정보를 저장하기 위한 방편으로 사용되고 있다. 그리고 MISR은 통상적으로 많은 수의 Flip-Flop과 XOR gate의 조합으로 이루어지기 때문에 이를 제거함으로써 추가적인 하드웨어 오버헤드 감소 효과를 이룰 수 있어 상용 프로세서 적용이 더욱 가능해졌다.

③ 기존 구조는 검사 대상 프로그램의 뼈대에 해당하는 control flow graph를 모방하여 동작한다. 즉 대상 프로그램이 분기 동작을 할 때 메모리에 저장된 기준 정보를 불러 동일한 분기 동작을 수행하게 된다는 것이다. 그러나 프로그램 내부에서 빈번히 발생되는 루프(loop)를 매번 검사할 때, 동일한 동작을 watchdog 프로세서가 반복하게 됨으로써 watchdog 프로세서와 검사 대상 프로세서가 서로 공통의 버스를 공유할 때 트래픽을 발생시키게 된다. 이는 결국 비효율적인 메모리 접근, 버스 점유 경쟁의 추가적인 시스템 수준의 오버헤드를 야기 시키게 된다. 본 구조는 이런 단점을 줄이고자 루프 검출기능을 추가하였다. 루프 검출 기능을 위한 하드웨어 구조를 추가하였고 해당 기능을 지원하기 위하여 기존 WDP의 instruction 구조를 이에 맞게 변형하였다. 이를 통하여 기준 정보 메모리 접근 횟수를 현저히 줄일 수 있었고 추가적인 시스템 수준의 오버헤드 및 성능 열화를 극복 할 수 있었다.

다음으로 제안하는 구조와 기존 루프 검출 관련 연구와의 차이점은 다음과 같다.

기존 루프 검출 관련 연구는 주로 프로세서의 분기 예측 방식(branch prediction)을 통한 성능 개선을 목적으로 연구된 것이다. 본 논문이 지향하는 실시간 에러 검출 분야에서는 본 논문을 제외하고는 루프 검출 구조가 적용된 사례가 없다.

제안하는 구조에서는 이런 점에 착안하여 기존 프로세서 아키텍쳐 분야의 기술을 실시간 에러 검출 분야에 접목하되 해당 분야의 특성 상 전체 시스템 구조에 큰 오버헤드를 주지 않고 이를 구현 할 수 있는 방안을 모색하였다. 따라서 기존 프로세서 분야의 루프 검출 기술에 비할 때는 매우 단순화된 기능만을 수행하지만 그로 인해 얻을 수 있는 효율성 증대는 실험 결과를 통하여 이미 적용할 만하다는 결론을 보여주고 있다.

V. 결 론

실시간 테스트는 더 이상 고 신뢰성을 요구하는 분야에 국한되지 않고 상용화된 프로세서에서도 중요한 이슈가 되어가고 있다. 따라서 실시간 동작 검사를 수행하는 구조의 하드웨어 오버헤드를 최소화하고 해당 검사 구조가 시스템의 성능에 미치는 영향을 최소화 할 수 있는 것이어야만 할 것이다. 제안하는 방법은 기존 연구들이 가지는 하드웨어 오버헤드 및 시스템에 미치는 성능 감소의 문제를 해결함과 동시에 검사 대상 프로세서가 처리하는 응용 프로그램에서 루프와 같은 반복적인 동작 패턴을 감지하여 검사를 위한 기준 정보들을 가져오는 메모리 접근 회수를 최소화 시켰다. 이를 통하여 기존 연구 성과들에서 보인 시스템 내 모듈과의 버스 경쟁 및 기준 정보를 가져오는데 필요한 메모리 접근 시간 지연 등의 문제를 최소화 시킬 수 있었다. 더불어 메모리 크기 감소를 위한 분기 주소 간격의 사용으로 실질적인 메모리 크기의 감소라는 성과를 얻은 것과 동시에 응용 프로그램의 크기에 대해서 기준 정보 메모리의 크기가 민감하지 않아 프로세서가 처리하는 다양한 프로그램에 대해서 적용이 가능하도록 하였다.

본 논문에서 제시하는 새로운 실시간 에러 검출 방법은 기존의 실시간 검사를 수행하는 구조보다 비용 효율적인 방식을 제안함으로써 실제 상용화된 프로세서에도 적용이 가능한 방법이라 할 수 있다.

참 고 문 헌

[1] A. Messer, P. Bernadat, G. Fu, D. Chen, Z. Dimitrijevic, D. Lie, D. D. Mannaru, A. Riska, D. Milojicic, "Susceptibility of Modern Systems and Software to Soft Errors," Technical Report, Hewlett Packard Lab, 2001.

[2] U. Gunneflo, J. Karlsson, J. Torin. "Evaluation of error detection schemes using fault injection by heavy-ion radiation," Proceedings of 19th International Symposium on Fault-Tolerant Computing, pp. 340-347, 1989.

[3] A. Mahmood and E. J. McCluskey, "Concurrent Error Detection Using Watchdog Processors - A Survey," IEEE Transactions on Computers, Vol. 37, pp. 160-174, 1988.

[4] N. R. Saxena and E. J. McCluskey, "Control-flow Checking Using Watchdog Assists and Extended Precision Checksums," IEEE Transactions on Computers, Vol. 39, pp. 554-559, 1990.

[5] P. Shen and M. A. Schuette, "On-line Self-monitoring Using Signatured Instruction Streams," Proceedings of International Test Conference, pp. 275-282, 1982.

[6] N. Oh, P. P. Shirvani, E. J. McCluskey, "Control-Flow Checking by Software Signatures," IEEE Transactions on Reliability, Vol. 51, pp. 111-122, 2002.

[7] D. J. Lu, "Watchdog Processor and Structural Integrity Checking," IEEE Transactions on Computers, Vol 31, pp. 681-685, 1982.

[8] Ohlsson and M. Rimen, "Implicit Signature Checking," Proceedings of 25th International Symposium on Fault-Tolerant Computing, pp. 218-227, 1995.

[9] Rajesh Venkatasubramanian, John P. Hayes, Brian T. Murray, "Low-Cost On-Line Fault Detection Using Control Flow Assertions," Proceedings of 9th IEEE International On-Line Testing Symposium, pp137-143, 2003.

[10] G. Xenoulis, Dimitris Gizopoulos, Nektarios Kranitis, Antonis M. Paschalis, "Low-Cost, On-Line Software-Based Self-Testing of Embedded Processor Cores," Proceedings of 9th IEEE International On-Line Testing Symposium, pp. 149-154, 2003.

[11] Alfredo Benso, Stefano Di Carlo, Giorgio Di Natale, Paolo Prinetto, "A Watchdog Processor to Detect Data and Control Flow Errors," Proceedings of 9th IEEE International On-Line Testing Symposium, pp. 144-148, 2003.

[12] M. Namjoo, E. J. McCluskey, "Watchdog Processor and Capability Checking," Proceedings of 25th International Symposium On Fault-Tolerant Computing, pp. 94-97, 1995.

[13] T. Michel, R. Leveugle, G. Saucier, "A New Approach to Control Flow Checking without Program Modification," Proceedings of 21st International Symposium on Fault-Tolerant Computing, pp. 334-341, 1991.

[14] M. R. de Alba and D. R. Kaeli, "Runtime Predictability of Loops," Proceedings of 4th IEEE Workshop on Workload Characterization, pp. 91-98, 2001.

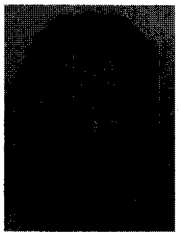
[15] M. Kobayashi, "Dynamic Characteristics of Loops," IEEE Transactions on Computers, Vol 32, pp. 125-132, 1984.

[16] T. Sherwood and B. Calder, "Loop Termination Prediction," Proceedings of the 3rd International Symposium on High Performance Computing, pp. 73-87, 2000.

저 자 소 개

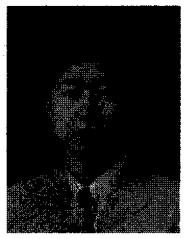
김 근 배(정회원)

2003년 연세대학교 전기공학과 학사 졸업.
2004년 연세대학교 전기전자 공학과 석사 졸업.
2005년 현재 연세대학교 전기전자 공학과 박사 과정.
<주관심분야 : On-line Test, Memory Test>



안 진 호(정회원)

1995년 연세대학교 전기공학과 학사 졸업.
1997년 연세대학교 전기공학과 석사 졸업.
2002년 LG전자 DTV연구소 선임연구원.
2005년 현재 연세대학교 전기전자공학과 박사과정. <주관심분야: SoC 설계 및 응용, NoC Test>



강 성 호(정회원)

1986년 서울대학교 제어계측 공학과 학사 졸업.
1988년 The University of Texas, Austin 전기 및 컴퓨터 공학과 석사 졸업.
1992년 The University of Texas, Austin 전기 및 컴퓨터 공학과 박사 졸업.
1992년 미국 Schlumberger 연구원.
1994년 Motorola 선임 연구원.
2005년 현재 연세대학교 전기전자공학과 교수.
<주관심분야: SoC 설계 및 SoC 테스트 >