論文2001-38SD-10-9

내장형 제어 RISC 코어를 위한 효율적인 랜덤 벡터 기능 검증 방법

(Efficient Verification Method with Random Vectors for Embedded Control RISC Cores)

梁 薰 模 *, 郭 承 浩 *, 李 文 基 *

(Hoon-Mo Yang, Sung-Ho Kwak, and Moon-Key Lee)

요 약

범용성이란 측면은 프로세서의 설계 과정 중 기능 검증의 중요도를 크게 부각시킨다. 따라서 본 논문은 기존 시뮬레이션 방법과 병행하여 기능 검증의 효율성을 높일 수 있는 효율적인 랜덤 벡터 기능 검증 방법을 제시한다. 본 기능 검증 방법은 내장형 제어 RISC 코어에 적합하며 실제 연세대학교와 삼성전자가 공동 개발한 32비트 프로세서인 CalmRISCTM-32의 코어 기능 검증에 적용하여 효율성을 확인한 바 있다. 본 기능 검증 방법은 클락 기반의 명령어 수준 시뮬레이터를 개발하여 이를 참조 모델로 삼고 랜덤 벡터로 이루어진 워크로드에 대해 HDL 시뮬레이션 결과와 비교함으로써 오류 검출을 수행하며 일반적인 테스트 벡터로써 발견하기 어려운 오류 유형을 보완하는 동시에 설계자에게 새로운 오류 유형의 기준을 제시하는 효과를 지닌다.

Abstract

Processors require both intensive and extensive functional verification in their design phase due to their general purpose. The proposed random vector verification method for embedded control RISC cores meets this goal by contributing assistance for conventional methods. The proposed method proved its effectiveness during the design of CalmRISCTM-32 developed by Yonsei Univ. and Samsung. It adopts a cycle-accurate instruction level simulator as a reference model, runs simulation in both the reference and the target HDL and reports errors if any difference is found between them. Consequently, it successfully covers errors designers easily pass over and establishes other new error check points.

Ⅰ. 서 론

프로세서 검증은 다양한 추상 레벨(예 : 아키텍쳐 레벨, RTL, 게이트 레벨)과 디자인의 특성(예 : 타이밍, 속도, 기능, 파워)을 포함하는 복잡한 과정이다. 프로세서의 복잡도가 증가할수록 프로세서 모델을 검증하기 위한 많은 방법이 만들어지고 있다. 검증 방법은 시뮬레이션에 기초한 디자인 검증(simulation-based design verification)과 정규 검증(formal verification)으로 크게 나눌 수 있다.[1]

정규 검증은 추상화된 디자인의 수학적인 기호를 이용하여 올바름을 증명하는 방법이고 게이트 수준 모델과 RTL 모델의 일치성을 확인하는 것과 같이 하위 수준의 추상화 모델에 적합하다. 이 방법은 아키텍쳐의 다른 구현 레벨 사이에 기능적 차이가 없음을 보여주는 equivalence checking과 디자인의 특정 기능의 올바

* 正會員, 延世大學校 電氣電子工學科
(Department of Electrical and Electronic Engineering, Yonsei University)

接受日字:2000年10月26日, 수정완료일:2001年9月20日

름을 검증하는 model checking을 포함한다. 이 방법은 대부분의 경우에 수작업으로 진행되며 디자인이 상위 수준 스펙을 만족시키는지 보증하기 어렵고 검증할 수 있는 회로나 상태 머쉰의 크기에 제한이 있다.[2]

시뮬레이션에 기초한 방법은 실제 하드웨어 시스템을 모델링한 환경에서 디자인의 소프트웨어 모델(프로세서의 경우는 명령어 셋)을 적용하여 검증하는 과정이다. 프로세서는 명령어에 의해 개별적인 동작이 정의되는 범용 칩으로서 최종적인 소프트웨어 응용 형태는 프로세서 설계자가 아닌 프로그래머가 명령어를 조합하여 결정한다. 따라서 프로세서 설계자는 개개 명령어의 고유 기능뿐만 아니라 임의의 명령어 조합에 대해 발생 가능한 모든 측면에 대해 동작의 안정성을 보장해야 한다. 이러한 이유로 시뮬레이션에 기초한 방법은 프로세서의 설계 과정에서 설계자가 디자인의 기능 검증과 타이밍 검증을 위해 일차적으로 사용하는 검증 방법으로 적용되고 있다.[3][4]

본 논문에서는 내장형 제어용 32비트 RISC 프로세서 : CalmRISC^{TM}-32의 검증을 위하여 아래와 같은 3단계의 시뮬레이션에 기초한 기능 검증 과정을 적용하였다.

(1) Compliance Check : 단일 명령어 수준 기능 검증

(2) Corner Check : 다수 명령어 조합 수준 기능 검증

(3) Algorithmic Check : 알고리즘 및 프로그램 수준 기능 검증

본 논문은 위 과정 중 기능 검증의 핵심이 되는 두 번째 단계의 미비점을 보완하는 효과적인 방법으로 랜덤 벡터 검증 방법을 제시한다.

벡터를 발생시키는 방법은 크게 세 가지로 구분할 수 있다. (i)수작업으로 발생시키는 방법은 마이크로아키텍처의 코너 케이스를 검증하기 위해 이용되지만 벡터를 만드는데 시간이 많이 걸리고 랜덤 벡터와 같이 사용되지 않는다면 디자인 에러를 충분히 잡아낼 수 없다.[3] (ii)결정적(deterministic) 테스트 발생 방법은 디자인의 자세한 구현 과정에 기초하여 디자인의 동작의 특정한 면을 검증하기 위한 테스트 벡터를 만들어 낸다. 이 방법은 파이프라인이나, 케쉬 제어기와 같은 특정 블록을 검증하기 위한 ad hoc 방법과 칩 전체를 검증하기 위한 일반적인 방법으로 나눌 수 있다. 이 방법은 적당히 작고 구조적으로 정의가 되어있는 디자인을 검증하는데 적합하다.[5][6][7][8] (iii)마지막으로 랜덤 벡터는 동시에 다수 개의 이벤트를 발생시키는 코드를 만들어내며 이 부분은 수작업에 의한 방법으로는 많은 시간을 소모하게 된다.[9] 랜덤 벡터는 설계자가 특정한 유형의 오류 검출에 유리한 방향으로 명령어 조합을 조절하지 못하기 때문에 독립적으로 사용할 경우에는 테스트 벡터로 비효율적이나 생성이 간단하고 오류 유형에 대한 설계자의 편견이 배제되기 때문에 실제 비-랜덤 벡터가 간과하기 쉬운 부분에 대한 오류 발견에 탁월하며 설계자가 미쳐 간파하지 못한 새로운 오류 유형 정립의 지침을 제공하기도 한다. 그러나 순수한 랜덤 벡터를 이용한 테스트는 테스트의 많은 부분이 유용하지 않고, 또 검증 환경의 물리적인 제약(예 : 메모리)을 벗어나기 때문에 시뮬레이션 시간을 낭비하게 된다. 따라서 랜덤 벡터를 보조 수단으로 비-랜덤 벡터와 병행하여 이용할 경우, 기능 검증의 효율성이 매우 높아진다. 그러나 랜덤 벡터 검증은 다음과 같은 비-랜덤 벡터와 차별되는 특수성을 고려해야 한다.

비-랜덤 벡터의 경우, 설계자가 워크로드 형식에 제약을 가하여 간편한 검증 환경 구축이 가능하다. 실제 CalmRISC^{TM}-32는 워크로드에 예상 정상 값과 도출된 값을 비교하는 루틴을 추가함으로써 HDL 시뮬레이션만으로도 기능 검증이 충분하였다. 그러나 랜덤 벡터는 동작을 예측하고 형식에 제약을 가하기가 매우 비능률적이므로 이러한 방식이 부적절하다. 따라서 본 연구는 HDL 모델과 비교하기 위한 참조 모델로서 명령어 수준 시뮬레이터를 설계하였으며 임의의 워크로드에 대해 두 모델의 정확한 동작 비교가 가능하도록 클락 사이클 수준의 타이밍 산출을 지원하도록 하였다. 한편 참조 모델은 그 자체로는 부담 요소이기 때문에 구축이 간편해야 하나 내장형 제어용 프로세서는 대체적으로 빠른 성능보다 경제적인 하드웨어를 선호하므로 동작 유형이 일관성이 결여되고 복잡하다. 따라서 본 연구는 적정 수준의 수행 속도를 보이면서도 복잡한 동작 기술이 용이한 규격성(modularity)을 갖춘 새로운 형식의 시뮬레이션 모델을 제안한다. 따라서 본 검증 환경은 HDL 모델과 참조 모델에 대한 시뮬레이션을 각각 수행하고 둘의 결과를 비교함으로써 오류를 보고하는 과정을 거치며 이 과정은 설계자의 중간 개입을 최소화하는 통합적이고도 일괄적인 방식으로 진행되므로 랜덤 벡터를 이용한 기능 검증의 편의성을 크게 향상시켰다.

본 논문의 체계는 다음과 같다. 제 II장에서 검증 대상 프로세서인 CalmRISC^{TM}-32의 아키텍처의 주요 사

양과 특성을 설명하고 제 Ⅲ 장에서 랜덤 벡터 검증 환경의 전체 구성 체계와 특성을 논한다. 제 Ⅳ 장에서는 제안한 랜덤 벡터 검증 환경의 핵심인 클락 사이클 기반 명령어 시뮬레이터의 구조에 대해 논하고 제 Ⅴ 장에서 제안한 랜덤 벡터 검증 환경의 실제 적용 예를 바탕으로 특성 및 성능을 분석하였고 마지막으로 최종 결론을 맺는다.

Ⅱ. CalmRISC^{TM}-32 아키텍쳐 개요

연세대학교 및 삼성전자간의 공동 프로젝트 일환으로 개발된 CalmRISC^{TM}-32는 저전력 고성능을 요구하는 내장형 응용 분야에 적합한 32비트 RISC 프로세서로서 내부적으로 코어, 부동 소수점 유닛, 캐쉬 메모리 및 내장 디버거 유닛으로 구성된다. 본 검증 환경은 이러한 유닛 중 검증 대상을 코어로 국한하며 다음과 같은 특성을 지녔다.[10]

2.1 명령어 군

내장형 제어용 프로세서의 명령어 군 채택은 디코딩의 효율성만큼 코드 밀집도가 중요 관건이며 CalmRISC^{TM}-32는 이에 부합하기 위해 다음과 같은 특성을 지닌다.

첫째, 본 프로세서는 데이터 워드 길이가 32비트인 반면에 코드 길이는 16비트이다.

둘째, 본 프로세서는 CISC 형태의 다중 사이클 명령어를 지원한다. 대표적인 예로서, 다중 바이트 크기 즉치 레지스터 저장 명령어, 고속의 컨텍스트 전환을 위한 다중 푸시/팝 명령어(*PUSHQn/POPQn*), 세마포어 등 배타적 메모리 접근을 위한 메모리 비트 처리 명령어 (*BIT[R/S/T/C]*) 등을 들 수 있다.

셋째, 본 프로세서는 ALU 연산에 의해 영향을 받는 상태 레지스터의 비트가 T비트라 불리는 단일 비트로 이루어졌다. T비트는 수행되는 명령어의 문맥에 따라 다르게 해석 적용되며 *BNZD*를 제외한 모든 분기 명령어는 T비트 값에 의해서만 조건 판단을 수행한다.

2.2 파이프라인 구조

본 프로세서는 5단 파이프라인 구조로 이루어졌다. 각각의 파이프라인 단 동작에 대한 요약을 표 1에 수록하였다. 본 프로세서는 하버드 아키텍처를 채용하였으므로 IFE 단의 명령어 페치와 MEM 단의 데이터 전송은 별도의 포트를 통해 개별적으로 이루어지나 예외적으로 *LDC* 계열 명령어의 경우, MEM 단에서 명령어 버스를 통해 데이터가 로드 된다.

본 프로세서는 지연 분기를 지원하며 하나의 지연 슬롯을 지닌다. 또한 본 프로세서는 정확한 예외 처리를(precise exception) 지원하며 예외 상황이 감지되면 바로 다음 클락 사이클에서 분기 주소에 대한 명령어 페치가 시작된다.

마지막으로 본 프로세서는 RAW 종속성에 의한 해저드 방지를 위해 레지스터 바이패스 및 메모리 잠금 제어를 지원한다. 이러한 제어 기법은 일반적인 프로세서 제어부 설계 시 폭넓게 적용되는 방식이기도 하다.

표 1. CalmRISC^{TM}-32 파이프라인 단

Table 1. Pipeline stages of CalmRISC^{TM}-32.

니모닉	기능
IFE	명령어 페치
DEC	명령어 디코딩, 코드 분기 실행
EXE	메모리 주소 계산 및 ALU 연산
MEM	메모리 로드/스토어
WRB	레지스터 갱신

Ⅲ. 랜덤 벡터 검증 환경

3.1 랜덤 벡터 검증 환경의 구성 체계

랜덤 벡터 검증 환경은 그림 1과 같이 총 5개의 과정을 거쳐 기능 검증이 수행되며 개개 과정의 목적은 아래와 같다.

SRC : 랜덤 벡터로 이루어진 어셈블리 소스 파일[ASM]을 생성한다.

ASM : ASM 파일을 이용하여 워크로드 파일을 생성한다. 생성되는 파일 중, HEX(또는 DAT) 파일은 HDL 시뮬레이션을 위한 명령어(또는 데이터) 리스트 파일이며 ILS 파일은 명령어 수준 시뮬레이션을 위한 이진 실행 파일이다.

HDL : HDL 시뮬레이션을 수행하여 VTR(Vertical TRace, 종적 트레이스) 파일을 생성한다.

ILS : 명령어 수준 시뮬레이션을 수행하여 HTR (Horizontal TRace, 횡적 트레이스) 파일을 생성한다.

CMP : 생성된 VTR 파일과 HTR 파일을 비교하여 서로 다른 점이 발견되면 오류를 보고한다.

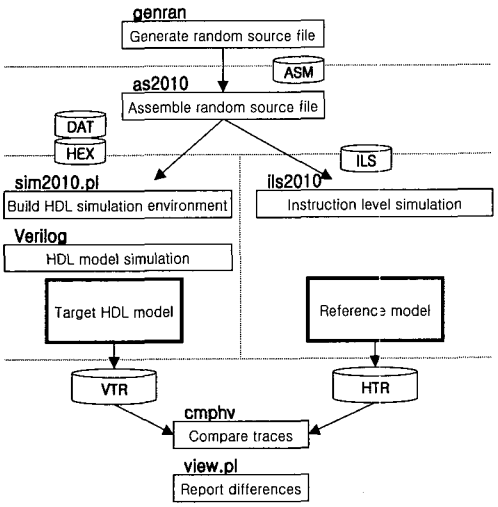

그림 1. 랜덤 벡터 검증 환경 구성 체계
Fig. 1. The structure of Random Vector Verification Environment.

본 검증 방법은 genran 유틸리티를 이용하여 랜덤 벡터 어셈블리 소스 파일을 생성하고 이를 CalmRISC^{TM}-32 전용 어셈블러인 as2010을 이용하여 시뮬레이션에 필요한 워크로드를 생성한다. as2010은 자체적으로 개발한 설계자 전용 어셈블러로서 범용 어셈블러와는 달리 링커는 포함되지 않았다. 이러한 모든 기능 검증 과정은 PERL 스크립트로 작성된 통합 환경 내에서 사용자가 지정한 옵션에 의거하여 일괄적으로 처리된다. 또한 명령어 수준 시뮬레이터는 검출된 오류에 대하여 편리한 디버깅을 위한 대화형 사용자 인터페이스를 제공한다.[11][12]

3.2 랜덤 벡터 시뮬레이션 고려 사항

본 랜덤 벡터 시뮬레이션은 정상적인 동작을 위해서 다음과 같은 특수성을 고려해야 한다.

첫째, 랜덤 벡터는 그 특성 상 코드 분기 주소나 데이터 메모리 주소가, 정의되지 않거나 접근 불가인 메모리 영역을 임의 참조함으로써 비정상적인 시뮬레이션 상태를 야기한다. 따라서 본 검증 환경은 매 버스 사이클마다 코어에서 인출되는 주소 대신 별도의 카운터에서 발생하는 순차적인 값을 명령어/데이터 참조 주소로 사용하고 코어에서 인출되는 주소는 오직 검증 대상으로서만 이용한다. 따라서 명령어는 코어 내부의 분기 여부에 상관없이 무조건 순차적으로 수행되며 명령어에서 계산된 데이터 주소와 실제 전송되는 데이터 사이에는 아무런 연관성이 없다.

둘째, 본 검증 환경은 메모리를 각각 CODE, DATA, KODE 영역으로 분할하며 CODE 영역에는 수행될 명령어 코드가 저장되고 DATA 영역에는 데이터가 저장된다. KODE 영역은 LDC 계열 명령어에 의해 전송되는 데이터를 저장하는 별도의 영역이다.

본 검증 환경의 명령어 수준 시뮬레이터는 비록 클락 단위의 정확한 타이밍 산출을 지원하나 연산 수행 순서가 이벤트 구동 방식처럼 시간 축에 정렬되는 것이 아니고 명령어 순서에 따라 순차적으로 결정된다. 따라서 IFE 단의 명령어 페치와 MEM 단의 데이터 전송이 메모리 영역을 공유하면 동일 워크로드를 수행해도 두 모델 간 연산 수행 순서가 부분 간격 상 서로 다르기 때문에 코드/데이터 참조를 위한 카운터 값의 일관성 유지가 매우 어려워진다. 본 검증 환경은 이를 해결하기 위해 데이터와 코드 메모리 영역을 분리하였다. 반면 LDC 명령어는 명령어 포트를 공유하며 MEM 단에서 데이터 전송이 수행되므로 이를 위해 별도로 KODE 영역을 두었다. 참조 카운터는 각각의 영역에 대해 별도로 존재하며 해당 영역 데이터 접근의 경우에만 값이 증가하고 최대 값에 도달하게 되면 CODE 영역일 경우, 시뮬레이션이 종료된다. 반면에 DATA 및 KODE 영역일 경우, 이들은 각기 초기 값으로 회귀한다.

셋째, 모든 랜덤 벡터 워크로드는 랜덤 테스트 벡터 이전에 프로세서 레지스터를 초기화하는 루틴이 배치되며 이러한 루틴은 오류가 발생하지 않았음에도 불구하고 초기화되지 않은 레지스터 값에 의해 시뮬레이션이 비정상적인 상태가 되는 것을 방지한다. 또한 레지스터 초기화는 더 다양한 조건을 충족시키기 위해 genran에 의해 생성되는 무작위 값으로 이루어진다.

3.2 참조 모델과 HDL 모델의 결과 비교

본 검증 방법의 오류 검출을 위한 비교 분석은 VTR 파일과 HTR 파일을 통해 이루어진다.

VTR 파일은 HDL 시뮬레이션을 통해 생성되며 클락 순서를 인덱스로 삼는 배열 구조로서 배열 요소에는 매 양의 클락 천이 시점에서 검출한 프로세서 주요 외부 신호 및 레지스터 값이 기록된다. HTR 파일은 명령어 수준 시뮬레이션을 통해 생성되며 실행되는 명령어 순서를 인덱스로 삼는 배열 구조로서 배열 요소는 5개의 파이프라인 단에 대응되는 슬롯이 각각 존재하며 이들 슬롯에는 발생한 이벤트의 내용과 시점이 기록된다. HTR 배열 요소에 기록된 이벤트의 발생 시점은

명령어 수준 시뮬레이터에 의해 정확히 계산된 값으로 참조 모델과 HDL 모델 사이의 결과를 매우 빠르고 간편하게 비교할 수 있게 한다.

이 두 파일의 비교 과정을 그림 2에 예시하였다. 그림에서 3번째 명령어는 HTR 파일 정보 상, 3번째 클락 시점에서 IFE 단을 점유하고 있다. 따라서 비교 루틴은 VTR 파일에서 3번째 요소를 가져와 명령어 주소인 PA를 포함한 대상 신호들의 값을 해당 HTR 요소와 비교한다.

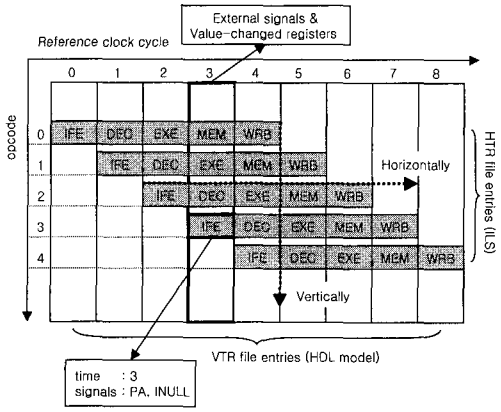

그림 2. 참조 모델과 HDL 모델 결과 비교
Fig. 2. The comparison between reference model and HDL model.

Ⅳ. 명령어 수준 시뮬레이터

4.1 명령어 수준 시뮬레이터 구성 체계

본 논문에서 제시하는 CalmRISC^{TM}-32 전용 명령어 수준 시뮬레이터는 C언어로 작성되었으며 전체적인 구성 체계는 그림 3과 같다.

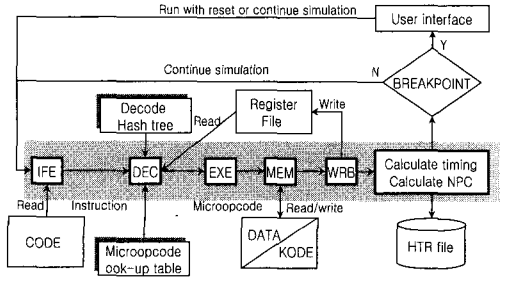

그림 3. 명령어 수준 시뮬레이터 구성 체계
Fig. 3. The structure of instruction level simulator.

본 시뮬레이터는 IFE 단에서 페치한 명령어 코드를 DEC 단에서 해시 테이블을 이용하여 명령어를 디코딩한 후, 마이크로코드 참조 테이블을 이용하여 명령어 유형에 따라 파이프라인 단 별로 각각 코드 분기(DEC), ALU 연산(EXE), 메모리 로드/스토어(MEM)를 수행하며 파이프라인 단 흐름 제어 및 클락 사이클 계산을 병행한다.

부가적으로, 본 시뮬레이터는 매 명령어 수행 직후, 실행 중지 주소 지정과 같은 사용자가 개입된 시뮬레이션 진행 제어나 코어 내부의 예외 상황 발생 여부 등을 검사하여 시뮬레이션을 중단하는 디버깅 기능을 지원한다. 따라서 사용자는 수행 도중 레지스터와 같은 내부 상태를 관찰하거나 시뮬레이션을 재개함으로써 대화형으로 디버깅을 수행할 수 있다.

4.2 명령어 디코딩 모델

프로세서의 명령어 디코딩은 연산자와 피연산자로 구분 수행된다. 이중 피연산자 해석은 비트 필드뿐만 아니라 연산자 유형과도 강하게 연관되므로 연산자만 일단 해석되면 쉽게 처리된다. 반면 연산자 해석은 연관 정보로서 비트 필드가 유일하며 이 또한 패턴과 연관 의미 간 대응 관계가 매우 자의적이므로 효율적인 처리를 위한 전용 탐색 알고리즘이 요구된다.

명령어 군의 피연산자 영역은 연산자 해석과 무관함에도 불구하고 비트 필드 상 연산자 지정 영역과 비슷한 비중을 차지한다. 따라서 탐색 과정에 피연산자 영역을 포함시킬 경우, 불필요한 메모리 소모와 느린 수행 속도를 초래한다. 따라서 본 논문은 그림 4와 같은 구조의 해시 테이블을 이용하여 연산자 해석과 무관한

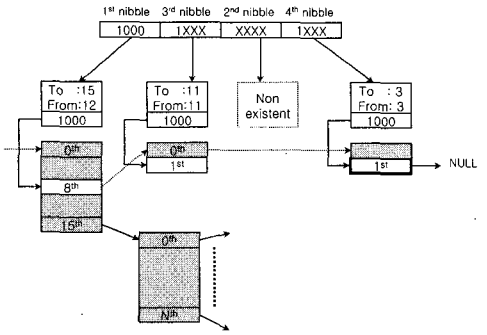

그림 4. 명령어 디코드 용 해시 테이블 구조
Fig. 4. The structure of hash table for instruction decoding.

피연산자 영역을 배제하는 새로운 탐색 알고리즘을 제안한다. 알고리즘 과정은 그림 5에 도시하였다.

```
/* 해시 테이블 노드 구조 */
struct hash_table {
 int from, to; // 탐색 키 부분 영역 지정
 struct duple {
   int id;  // 명령어 인식 번호
   struct hash_table *next; // 다음 노드 포인터
 } *slot;
} *root; //해시 테이블 루트 노드
```

```
/* 비트 필드 bf에 대한 명령어 인식 번호 탐색 */
int decode(long bf)
{
  int i;
  struct hash_table *p;
  struct duple *q;

  for(p=root; p!=NULL; p=q->next) {
  // 탐색키로부터 from, to가 지정하는 비트열 추출
    i=extract(bf, p->from, p->to);

  // 비트열을 인덱스로 삼아 slot 배열 접근
    q=p->slot[i];
    p=q->next;
  }

  // 명령어 인식 번호 반환
  return q->id;
}
```

그림 5. 해시 테이블을 이용한 탐색 알고리즘
Fig. 5. The search algorithm using hash table.

탐색키인 명령어 비트 필드는 고정된 L-길이 부분열로 분할되며 모든 해시 테이블 노드는 이들 부분열 중 하나와 연관된다. 노드는 다음 노드와 연결하기 위한 포인터 배열을 포함하며 연관되는 부분열 패턴을 모두 수용할 수 있는 배열 크기는 2^L이며 부분열이 K개의 무관 비트를 포함할 경우 배열 크기는 2^{L-K} $(K<L)$로 축소 가능하다. 단 본 알고리즘은 부분열의 축소된 영역 값을 그대로 포인터 배열 인덱스로 사용하기 위해 무관 비트가 부분열의 좌우 경계에 분포한 경우에만 영역 축소를 허용한다. 따라서 $0XX1_2$같은 패턴의 경우 영역 축소가 불가능하다. 또한 부분열 전체가 무관 비트일 경우, 연관 노드를 생성하지 않고 바로 이전 노드를 다음 노드와 연결시켜 메모리와 수행 시간을 동시에 감소시킨다.

본 탐색 알고리즘은 수행 속도가 명령어 군의 직교성에 크게 종속되지 않기 때문에 32비트 RISC 계열처럼 직교성이 높은 기종의 경우, 비트 패턴에 특화된 방식이 속도나 메모리 면에서 오히려 더 유리하다. 그러나 직교성이 낮은 내장형 제어 프로세서일 경우 이러한 경향은 역전된다. 일례로 본 탐색 알고리즘을 CalmRISC^{TM}-32 명령어 군에 적용한 결과 탐색 중 경유하는 노드의 개수는 최대 4번, 평균 2.7번인 반면 총 노드의 수는 332개로 제한되는 높은 효율성을 보였다. 또한 본 모델은 깊이 우선 탐색을 적용한 재귀 호출을 통하여 자동 생성이 가능하므로 설계자의 편이성을 크게 향상시킨다.

4.3 제어부 모델

CalmRISC^{TM}-32를 위시한 내장형 제어용 RISC 프로세서는 RISC 설계 철학이 고수해 온 간결한 명령어 군이라는 제약에서 다소 느슨한 입장을 취하여 복잡한 다중 사이클 명령어도 시스템 설계자의 요구에 의해 적극 도입하는 추세이다. 따라서 본 시뮬레이터는 실제 RISC 제어부 설계 시 적용되는 상태 머신 기반의 하드와이어(hard-wired) 방식을 답습하는 대신 규격성이 더 우월한 마이크로프로그램(microprogram) 방식으로 제어부 구조를 재구성하였다. 모든 명령어는 마이크로코드라는 동일 연산 단위로 이루어진 마이크로프로그램과 일대일 대응되며 단일 사이클 명령어는 단일 마이크로코드로, 다중 사이클 명령어는 여러 개의 마이크로코드로 구성된다.

본 시뮬레이터 제어부 모델의 전체 구조는 그림 6과 같다. 해시 테이블 탐색 결과인 명령어 인식 번호는 해당 마이크로프로그램의 첫 번째 주소이며 이후 해당 마이크로코드는 단일 연계 리스트(single linked list) 구조로 연결되고 EOP(end of program, -1)는 리스트의 종단임을 의미한다.

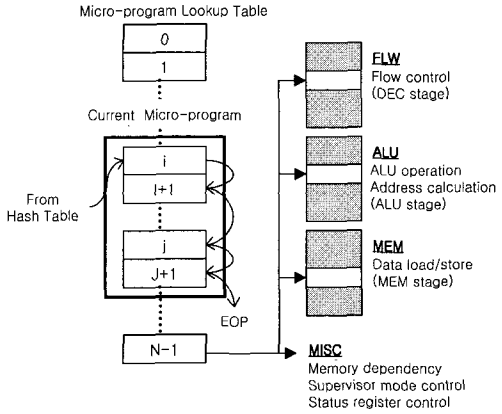

그림 6. 마이크로프로그램 방식의 제어부
Fig. 6. The control unit using microprogramming method.

본 시뮬레이터는 마이크로코드가 가리키는 FLW, ALU, MEM 테이블 상 제어 워드들을 참조하고 파이프라인 단 별로 해당 연산을 수행하며 현재 명령어에 대한 마지막 마이크로코드 수행이 완료되면 다음 명령어에 대한 페치 및 디코딩을 수행한다. 이와 같은 계층 구조(hierarchy)는 파이프라인 연산 수행과 타이밍 산출이 명령어 대신 마이크로코드와 직접 연관됨으로써 명령어 유형과 독립되어 시뮬레이터의 유연성과 확장성을 더욱 향상시킨다.

4.5 명령어 분기 제어 모델

본 시뮬레이터는 효율적인 코드 분기 처리를 위해 그림 7와 같은 구조의 파이프라인 상태 환형 큐와 윈도우를 이용한다. 환형 큐는 파이프라인 흐름 제어 정보를 기록하는 슬롯으로 구성되며 윈도우는 이러한 큐의 연속적인 일부 영역을 가리킨다. 한편 슬롯은 클락 사이클이 아닌 파이프라인 단에 일대일 대응한다.

본 시뮬레이터는 마이크로코드 수행이 종료될 때마다 marknpc 루틴을 통해 명령어 분기 요청이 발생한 파이프라인 단 바로 다음과 대응되는 슬롯에 분기 명령어 주소와 분기 유형을 기록하며 분기 유형은 각각 명령어 순차 진행(NORMAL), 비지연 분기(FLUSH), 지연 분기(DSLOT)로 분류된다. 이후 다음 마이크로코드를 수행하기 전에 윈도우는 환형 큐 상 슬롯 한 개 차이만큼 전진하며 IFE 단은 새로 대응되는 윈도우 슬롯으로부터 명령어 페치 주소를 가져온다. 본 시뮬레이터는 IFE 단을 제외한 모든 파이프라인 단을 통과할 때마다 연산 수행 전에 대응되는 슬롯을 참조한다. 따라서 마이크로코드가 분기 유형이 FLUSH인 슬롯을 통과하면 그 이후로 플러시(flush) 된다. 한편 IFE 단은

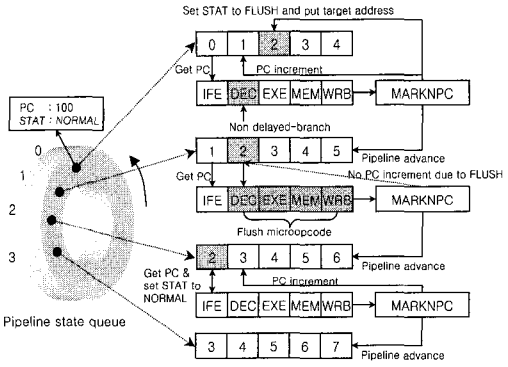

그림 7. 윈도우를 이용한 코드 분기 제어
Fig. 7. Code branch control using window.

명령어 페치 주소를 가져올 때, 대응되는 슬롯의 분기 유형을 무조건 NORMAL로 복귀시킴으로써 환형 큐 상 윈도우가 한 번 이상 순환한 후 이미 처리된 이전 기록에 의해 다시 플러시가 발생하는 것을 방지한다. 프로세서는 파이프라인의 특성에 의해 서로 다른 명령어 분기가 동시 발생할 수 있다. 예를 들어 현재 마이크로코드가 DEC 단에서 지연 분기가 발생하고 다음 마이크로코드가 IFE 단에서 순차 진행이 발생하면 동일 슬롯에서 충돌이 발생한다. 따라서 본 시뮬레이터는 FLUSH, DSLOT, NORMAL에 대해 내림차순으로 우선 순위를 부여하고 특정 슬롯의 분기 유형 갱신은 갱신할 유형이 이미 기록된 유형보다 우선 순위가 높은 경우에만 허용하도록 설정하였다. 본 내용은 종합적인 이해를 위해 그림 7에 비 지연 분기 명령어 처리 과정을 도시하였다.

4.4 클락 사이클 산출 모델

HDL 모델의 이벤트 구동 방식은 연산 수행 순서를

```
enum pipeline_stage {
  IFE=0, DEC, EXE, MEM, WRB};

time_t st[WRB+1]; // 단 점유 시작 시간
time_t et[WRB+1]; // 단 점유 종료 시간

/* 리셋 발생 시 타이밍 초기화 */
void initialize()
{
   st[IFE]=-1; et[IFE]=-1;

   for(i=IFE; i<=WRB; i++) {
    st[i]++;  et[i]++;
   }
}

/* 매 마이크로코드 수행 종료 이후 호출 */
void calctiming()
{
   st[IFE]=et[IFE]+1;
   st[DEC]=et[DEC]+1;
   et[IFE]=st[DEC]-1;

  // 메모리 잠금 검사
  if(check_memlock()) {
    // 메모리 잠금이 존재할 경우 (조건 A)
    et[DEC]=et[MEM];
    et[EXE]=et[EXE]=et[MEM]+1;

  } else {
    st[EXE]=et[EXE]=et[MEM];
    et[DEC]=st[EXE]-1;
  }

  st[MEM]=et[MEM]=et[EXE]+1;
  st[WRB]=et[WRB]=et[MEM]+1;
}
```

그림 8. 마이크로코드에 대한 클락 사이클 산출 알고리즘
Fig. 8. The algorithm of clock cycle calculation for microcode.

실시간 축에 정렬하므로 복잡한 예약 기법을 적용하여 서로 다른 마이크로코드를 파이프라인 상 동시에 동작시킨다. 반면 본 시뮬레이터는 예약 기법을 사용하지 않고 마이크로코드를 하나씩 순차적으로 동작시킨다.

다만 마이크로코드가 수행 종료될 때마다 파이프라인 통과 시 경유 단에 대한 점유 시작 시점과 종료 시점을 산출하기 때문에 연산 수행 순서는 서로 달라도 타이밍은 HDL 모델과 일치한다. 마이크로코드에 대한 클락 사이클 산출 알고리즘은 그림 8에 제시하였다. 제안한 방식은 이벤트 구동 방식과 달리 타이밍 산출이 불가능한 파이프라인 동작 유형이 일부 존재하나 그 제약은 미미하며 오히려 연산 처리 과정에서 데이터 종속성을 고려할 필요가 없으므로 레지스터 바이패스(Register bypass)나 메모리 잠금에 대한 처리가 간단해 진다. 다만 메모리 잠금은 레지스터 바이패스와 달리 파이프라인 타이밍에 영향을 주므로 클락 사이클 산출 알고리즘에서 별도의 처리가 요구된다. 또한 메모리 잠금 발생 여부(조건 A)는 현재와 바로 이전 마이크로코드의 유형을 서로 비교함으로써 감지된다.

단일 사이클 명령어는 단일 마이크로코드와 동일하여 마이크로코드에 대한 산출 알고리즘을 그대로 적용하나 다중 사이클 명령어는 유형에 따라 좀 더 복잡하다. CalmRISC™-32의 다중 사이클 명령어는 그림 9와 같이 크게 2가지 유형으로 구분된다.

이 중 첫 번째는 모든 마이크로코드가 코드 페치를 수행하는 반면 나머지는 처음 마이크로코드만 코드 페치를 수행하는 차이점이 있다. 따라서 전자는 단일 사이클 명령어에 대한 방식과 차이가 없으며 후자는 IFE 단과 DEC단에 대한 타이밍을 보정하기 위해 마이크로 프로그램의 마지막 마이크로코드가 수행 종료되면 마이크로프로그램 전체에 대한 파이프라인 단 점유 상태를 재구성하여 단일 마이크로코드로 대치시켜 다음 마이크로코드 타이밍 산출 시 참조하도록 한다. 재구성이 필요한 부류는 각각 DEC 단 또는 MEM 단을 연속 점유하는 두 가지 유형으로 세분화된다. 그림 7의 화살표는 재구성 과정의 파이프라인 단 점유 시간별 종속 관계와 산출 순서를 유형별로 나타낸다.

V. 결과 및 고찰

본 장에서 제시하는 성능 측정은 512M 메모리가 장착된 SPARC-60 Solaris 2.7에서 유닉스 명령어 time을 이용하였다. 제안한 검증 환경의 단계별 처리 속도 측정 결과는 표 2와 같다. 이중 HDL 시뮬레이션은 합성 가능한 Verilog™ RTL 모델에 대해 Cadence사의 VerilogExcel™ 시뮬레이터를 적용하였다. 전체 기능 검증 처리 시간 중 HDL 시뮬레이션을 제외한 나머지 과정이 차지하는 비율은 불과 3% 미만으로 참조 모델의 시뮬레이션이 초래하는 부담은 무시할 수 있을 정도로 극히 미미하다.

본 연구는 또한 랜덤 벡터 검증 환경의 핵심인 명령어 수준 시뮬레이터의 성능 비교를 수행하였다. 비교 대상은 MIPS 프로세서 명령어 군 기반인 SimpleScalar 버전 2.0을 선택하였다. 속도 측정 시 적용한 SimpleScalar 옵션은 클락 사이클 산출이 가능하며 단일 명령어 인출 파이프라인 구조인 1-way out-of-order 모드로서 워드 크기가 64비트인 점을 제외하면 CalmRISC™-32 구조와 거의 유사하다. 표 2을 보면 제안한 시뮬레이터는 트레이스를 생성할 경우 약 4.2배 더 빠르며 생성 트레이스 특성이 서로 다르다는 것을 감안하여 트레이스를 생성하지 않을 경우만을 비교해도 약 1.2배 더 빠르다.

CalmRISC™-32는 랜덤 벡터 검증을 수행하기 전에 이미 HDL 모델에 대해 1차 수준인 단일 명령어에 대한 기능 검증은 모든 경우에 대해 완벽하게 수행하였

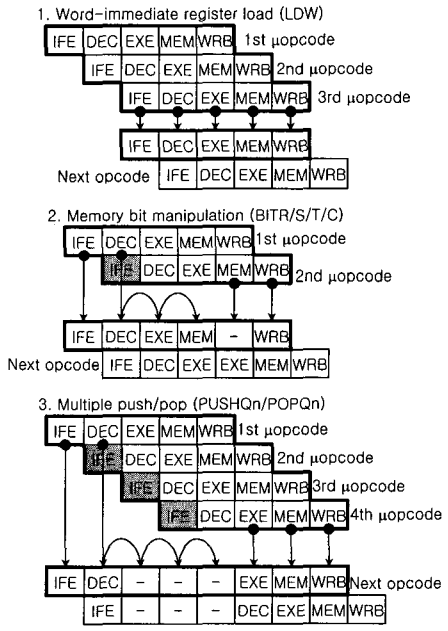

그림 9. 다중 사이클 명령어 타이밍 산출
Fig. 9. Timing calculation of multi-cycle instruction.

으며 2차 수준인 소수 명령어 조합의 경우, 동작 유형별로 분류되어 구성 명령어 총합이 10만 개인 워크로드에 대해 검증을 완료하였다. 그러나 이후 대략 36만 개의 명령어를 이용한 랜덤 벡터 기능 검증을 수행한 결과 총 21개의 오류가 추가로 관찰되었다. 발견된 오류 유형의 분포는 표 4와 같다.

명령어 유형 중 가장 오류가 빈번한 경우는 예외 처리 과정이며 특히 CalmRISC^{TM}-32 명령어 비트 필드의 비정형성에 의해 비정의 명령어를 인식하지 못하는 오류가 빈번하였다.

표 2. 본 검증 환경의 단계 별 수행 속도

Table 2. Execution speed of the verification environment at each level.

검증 과정	속도(IPS)	점유율(%)
SRC/ASM	12,000	1.2
ILS	22,000	0.7
HDL	150	97.3
CMP	18,000	0.8

표 3. 타 기종과의 수행 속도 비교

Table 3. The comparison of execution speed of other processors.

트레이스 생성	기 종	속도(IPS)
유	SimpleScalar	5,300
	제안한 시뮬레이터	22,000
무	SimpleScalar	89,000
	제안한 시뮬레이터	105,000

하드웨어 유형 중 가장 오류가 빈번한 부분은 상태 레지스터로서 예외 처리 과정과 맞물린 특권 모드 비트는 물론 일반 ALU 명령어와 관련된 상태 비트 변환 오류도 관찰되었다. 또한 다중 바이트 즉치 저장 명령어, LDC 계열 명령어와 명령어 페치의 공유에 의한 명령어 버스 오류 발생률도 높은 수치를 보였다.

랜덤 벡터에 의해 검출된 오류 유형의 특성은 오류 발생 빈도가 높은 영역이 상태 레지스터나 명령어 버스처럼 서로 다른 명령어에 의한 공유가 빈번하다는 점이다. 또한 이러한 오류는 단일 명령어가 아닌 명령어 조합에 의해서만 발견되나 명령어 조합이 2개 이상만 되어도 모든 경우를 대처하기에 워크로드 용량이 너무 크다. 한편 CalmRISC^{TM}-32를 위시한 내장형 제어용 프로세서는 서로 다른 명령어에 대한 하드웨어 공유를 구현 회로 절약을 위한 주요 기법으로 중시하므로 이러한 오류가 발생할 확률이 훨씬 높다. 따라서 CalmRISC^{TM}-32의 기능 검증이 예시하듯이 랜덤 벡터는 유형별로 최적화된 2차 수준 워크로드가 수용하지 못한 오류 검출을 효율적으로 보완함으로써 기능 검증의 효율성을 극대화한다.

표 4. 오류 유형의 분포

Table 4. The distribution of error patterns.

명령어 유형	개수	하드웨어 유형	개수
예외 처리	13	상태 레지스터	12
다중 사이클 명령어	2	코드 버스	6
단일 사이클 명령어	5	종속성 검사	1
포괄적인 경우	1	기타	2
총 합	21	총 합	21

VI. 결 론

본 논문은 32비트 내장형 제어용 RISC 프로세서인 CalmRISC^{TM}-32의 기능 검증을 위한 효율적인 랜덤 벡터 기능 검증 방법을 제안하였다. 본 기능 검증은 가장 임의적인 형태의 워크로드인 랜덤 벡터를 수행하기 위해 클락 수준 사이클 타이밍 산출이 가능한 명령어 수준 시뮬레이터를 개발하였고 이를 참조 모델로 삼아 HDL 시뮬레이션 결과와 비교함으로써 오류를 검출한다. 본 시뮬레이터는 해시 테이블 구조 명령어 디코딩 모델과 마이크로프로그램 기반 제어부 구조를 이용하여 높은 규격성을 유지함으로 높은 수행 속도와 더불어 설계 편이성을 증진시켰다. 본 기능 검증 방식은 CalmRISC^{TM}-32에 대해 이미 단일 명령어 수준 및 동작 유형별 명령어 조합 수준 기능 검증을 완료한 후에도 약 36만 개의 명령어를 사용하여 총 21개의 오류를 추가로 검출하였다. 따라서 본 환경은 유형별로 분류된 명령어 조합 수준 기능 검증이 간과하기 쉬운 오류 유형을 효과적으로 보완함으로써 기능 검증의 효율성을 높일 수 있다. 한편 본 명령어 수준 시뮬레이터의 구조는 다른 내장형 응용 프로세서에 대해 쉽게 확장 가능하며 향후 이 구조를 바탕으로 모든 내장형 응용 프로

세서를 통합할 수 있는 범용 동작 기술 언어를 개발할 계획이다.

참 고 문 헌

[1] C. Pixley, N. Strader, W. Bruce, J. Park, M. Kaufmann, K. Shultz, M. Burns, J. Kumar, J. Yuan, and J. Nguyen, "Commercial Design Verification : Methodology and Tools," Proc. IEEE Int. Test Conf., pp. 839~848, 1996.

[2] Windley, P.J. "Formal modeling and verification of microprocessors," IEEE Transactions on Computers, Vol.44, No.1, pp.54~72, Jan. 1995.

[3] M. Kantrowitz and L.M. Noack, "I m Done Simulating ; Now What? Verification Coverage Analysis and Correctness Checking of the DECchip 21164 Alpha Microprocessor," in Proc. Design Automation Conf., pp. 325 330, 1996.

[4] S. Taylor, M. Quinn, D. Brown, N. Dohm, S. Hildebrandt, J.Huggins, and C. Ramey, "Functional Verification of a Multiple-Issue, Out-Of-Order, Superscalar Alpha Processor : The DEC Alpha 21264 Microprocessor," Proc. Design Automation Conf., pp.638--643, 1998.

[5] H. Iwashita, T. Nakata, and F. Hirose, "Integrated Design and Test Assistance for Pipeline Controllers," IEICE Trans. Information and Systems, Vol. E76-D, No. 7, pp.747~754, 1993.

[6] D.C. Lee and D.P. Siewiorek, "Functional Test Generation for Pipelined Computer Implementations," Proc. Int. Symp. Fault Tolerant Computing, pp.60~67, 1991.

[7] M.S. Abadir, J. Ferguson, and T.E. Kirkland, "Logic Design Verification via Test Generation," IEEE Trans. Computer-Aided Design, Vol. 7, No. 1, pp.138~148, Jan. 1988.

[8] R.C. Ho, C.H. Yang, M.A. Horowitz, and D.A. Dill, "Architecture Validation for Processors," Proc. Int. Symp. on Computer Architecture, pp. 404~413, 1995.

[9] Ta-Chung Chang, "A Biased Random Instruction Generation Environment for Architectural Verification of Pipelined Processor," in Journal of Electronic Testing : Theory and Applications 16, pp.13~27, 2000.

[10] Sangyeun Cho et.al., "CalmRISC^{TM}-32: 32-Bit Low Power MCU Core," in Proceedings of the Second IEEE Asia Pacific Conference on ASICs, pp.285~289, Aug., 2000.

[11] Moon-Key Lee, Young-Wan Kim, Kwang-Soo Seo, Seung-Il Sonh, "A Compatibility Verification Environment for HDL-modeled Microprocessors'" in Journal of The Korean Institute of Communication Sciences, Vol.21, No.2 , pp.409~416, Feb.,1996.

[12] Moon Key Lee, Jeong Yop Lee, Young Wan Kim, and Kwang Soo Seo, "The Environment for Verifying MS-DOS Compatibility of HDL modeled Microprocessor," in Journal of KITE, Vol.32-A, No.7, pp.115~122, Jul. 1995.

저 자 소 개

梁 薰 模(正會員)

1994년 : 연세대학교 전자공학과 학사. 1996년 : 연세대학교 전자공학과 석사. 1996년 9월~현재 : 연세대학교 대학원 전기전자공학과 박사과정 재학중. <주관심분야> 마이크로프로세서 구조 및 설계. 상위수준 시뮬레이터 설계

郭 承 浩(正會員)

1994년 : 연세대학교 전자공학과 학사. 1996년 : 연세대학교 전자공학과 석사. 1996년 9월~현재 : 연세대학교 대학원 전기전자공학과 박사과정 재학중. <주관심분야> 저전력 프로세서 아키텍쳐 연구 및 설계. 마이크로프로세서 구조 및 설계

李 文 基(正會員)

1965년 : 연세대학교 전기공학과 학사. 1967년 : 연세대학교 전자공학과 석사. 1973년 : 연세대학교 전자공학과 박사. 1980년 : 미국 Univ. of Oklahoma 전자공학과 박사. 1992. 9~1996. 8 : 연세대학교 부설 아식 설계공동연구소 소장. 1995년 : 대한전자공학회 회장. 1996년 7월 : 대한전자공학회 창립 50주년 기념 학술대회(ITC-CSCC'96) 대회장. 1996년 8월 : 헝가리 부다페스트 계측 및 컴퓨터 연구소 초빙 연구원. 1996년 9월~1997년 8월 : 일리노이즈 대학교 전기전산공학과 방문연구교수. 1998년 : 국민훈장(석류장) 수여. 1999년 2월 : 한국반도체학술대회 공동위원장. 1999년 8월 : IEEE AP-ASIC 학술대회장. 2000년 8월 : IEEE AP-ASIC 학술대회장. 1982년 2월~현재 : 연세대학교 전기전자공학과 교수. <주관심분야> 마이크로프로세서 구조 및 설계. IP 개발 연구. ATM 스위치 구조 연구 및 설계. 3D 그래픽 가속기 연구. 신경망 프로세서 연구 및 설계