임베디드 코어 설계시 효율적인 설계 공간 탐색을 위한 컴파일드 코드 방식 시뮬레이터 생성 시스템 구축

준회원 김 상 우*, 정회원 황 선 영*

Construction of a Compiled-code Simulator Generation System for Efficient Design Exploration in Embedded Core Design

Sang-Woo Kim* *Associate Member*, Sun-Young Hwang* *Regular Member*

요 약

본 논문은 어플리케이션에 최적화된 임베디드 시스템 설계에 있어 효율적인 설계 공간을 탐색할 수 있도록 머신 기술 언어를 기반으로 한 컴파일드 코드 방식 시뮬레이터 생성 시스템을 제안한다. 제안된 시스템은 event-driven 시뮬레이션의 융통성을 유지하면서 많은 시뮬레이션 시간을 소요하는 인스트럭션 펫치와 디코딩 과정을 정적으로 결정하여 빠른 수행시간을 갖는 컴파일드 코드 방식 시뮬레이터를 생성한다. 생성된 시뮬레이터는 임베디드 코어의 성능 측정을 위한 사이클 수준과 인스트럭션 수준의 시뮬레이션을 가진다. 구축된 컴파일드 코드 방식 시뮬레이터 생성기의 효율성을 확인하기 위해 JPEG 인코더 어플리케이션에 대한 아키텍처 탐색을 수행하였다. 제안된 시스템은 MIPS R3000 프로세서의 초기 임베디드 코어로 시작하여 어플리케이션에 최적화된 임베디드 코어를 얻어내었다. 이 과정에서 많은 시뮬레이션 시간이 요구되었다. 사이클 수준 컴파일드 코드 방식 시뮬레이터는 event-driven 시뮬레이션의 정확성을 가지며 평균 21.7%의 향상된 시뮬레이션의 수행 속도를 보인다.

Key Words : 컴파일드 코드 방식 시뮬레이터, 성능 측정, 임베디드 시스템, 아키텍처 탐색, 머신 기술 언어

ABSTRACT

This paper proposes a compiled-code simulator generation system based-on machine description language for efficient design space exploration in designing an embedded system optimized for a specific application. The proposed system generates a compiled-code simulator which maintains the functional accuracy of an event-driven simulator by determining instruction fetch and decoding processes statically. Generated simulator takes instruction-level and cycle-level simulation for estimating performances in embedded core. To show the efficiency of the constructed compiled-code simulator generator, architecture exploration had been performed for the JPEG encoder application. Starting with MIPS R3000 processor for one embedded core, the proposed system can produce the core showing optimized execution time for the application programming. In this process, a huge amount of simulation time has been used. Cycle-level compiled-code simulator has the functional accuracy and shows performance improvement by 21.7% in terms of simulation speed on the average when compared with an event-driven simulator.

※ 본 연구는 교육과학기술부의 재원으로 한국연구재단의 지원을 받아 수행되었습니다. (#2010-0008043).

* 서강대학교 전자공학과 CAD & ES 연구실 (hwang@sogang.ac.kr)

논문번호 : KICS2010-08-373, 접수일자 : 2010년 8월 4일, 최종논문접수일자 : 2010년 12월 21일

I. 서 론

모바일 멀티미디어 기기를 중심으로 발전하고 있는 임베디드 시스템은 고성능과 저전력 소비, 그리고 다양한 어플리케이션에 대한 수요자의 요구를 동시에 만족하기 위해 시스템의 핵심기능을 하나의 칩으로 집적하는 시스템 반도체 설계기술이 중요하게 되었다[1-3]. ASIC (Application Specific Integrated Circuits)은 특정 어플리케이션에 대해 고성능 프로세서의 설계가 가능하지만 고려해야 할 어플리케이션이 많을수록 시스템 설계 복잡도가 증가하므로 time-to-market을 만족시키기가 어렵게 되어, 이미 검증된 IP (Intellectual Property)를 재사용하는 설계 방식이 사용된다[2]. IP 기반의 ASIC 설계 방식은 멀티미디어, 통신 시스템 등의 다양한 어플리케이션 IP을 이용하여 다양한 시스템 설계가 가능하나 어플리케이션의 변화와 설계 스펙이 변동할 경우에 능동적으로 대처하기가 어렵고 긴 설계 시간과 많은 비용이 요구된다. ASIP (Application Specific Instruction-set Processor)은 어플리케이션에 적합한 프로세서 구조와 인스트럭션 셋을 가지며 짧은 설계 기간 내에 사용자의 다양한 요구사항을 만족시키는 높은 수준의 융통성과 재사용성을 제공한다[3-5].

ASIP 설계를 위하여 configurable 프로세서 기반의 설계 방식과 머신 기술 언어 기반의 설계 방식이 제안되었다[6-10]. Configurable 프로세서 기반의 설계 방식은 범용 프로세서를 바탕으로 레지스터 파일 크기, 버스 데이터 폭 등의 매개변수 설정을 통해 ASIP의 성능을 극대화시키는 설계 방법이다[6]. 이 방법은 타켓 프로세서와 관련 개발 도구를 쉽게 얻어내지만 범용 프로세서를 기반으로 제한된 ASIP을 생성하기 때문에 설계 융통성의 감소와 어플리케이션에 최적화된 프로세서 설계의 어려움이 있다. 머신 기술 언어 기반의 설계 방식은 설계자가 머신 기술 언어로 기술한 타겟 프로세서의 모델을 바탕으로 ASIP 설계 자동화 시스템에서 어플리케이션에 적합한 ASIP과 관련 컴파일러, 시뮬레이터 등을 자동 생성하며 제한된 소비전력과 고성능을 가진 ASIP을 설계하는 방법이다[3,5,11-15]. 머신 기술 언어로부터 컴파일러, 시뮬레이터 등의 자동 생성은 설계 정확성과 일관성을 검증하는 아키텍처 탐색을 수행하여 어플리케이션에 최적화된 프로세서의 설계를 가능케 한다[13,14]. 아키텍처 탐색은 설계자가 어플리케이션의 특성을 분석한 결과를 바탕으로 초기 프로세서 모델을 결정하고, ASIP의 성능을 향상시킬 수 있는 설계 모듈을 사용하여 프로세서 구조와 인스트럭션 셋 등을 개선하며 어플리케이션에 최적화된 프로세서를 얻을 때까지 최적화 과정을 반복한다[16]. 다양하고 넓은 탐색 공간에서 어플리케이션에 최적화된 ASIP과 관련 설계 도구를 빠르게 얻어내는 아키텍처 탐색은 고성능과 저전력을 동시에 만족하는 시스템 반도체를 적시에 설계하여 성공적인 시장진입이 가능하다[16-18]. 시뮬레이터는 ASIP 설계 결과물의 검증과 함께 아키텍처 탐색 과정에서 다음 단계의 프로세서 설계를 위한 프로파일 정보를 얻어낼 수 있어 최적화된 ASIP 설계에 있어 중요한 설계 도구이다[17,19-23]. Event-driven 시뮬레이터를 이용한 아키텍처 탐색은 타겟 프로세서의 성능 측정에 대해 높은 융통성을 가지므로 올바른 설계 검증이 가능하나 느린 시뮬레이션의 수행 속도로 인해 설계 시간과 비용이 증가하게 된다[13,14,23]. 이를 극복하기 위해 event-driven 시뮬레이터의 융통성을 유지하는 범위에서 정적으로 시뮬레이션을 수행하는 컴파일드 코드 방식 시뮬레이터가 제안되었으나, 아키텍처 탐색 과정의 설계 단계에 알맞은 시뮬레이션의 수행 속도와 성능 측정의 정확성을 고려하지 않아 시뮬레이션에 많은 시간이 요구된다[20-23].

본 논문에서는 임베디드 코어 설계와 효율적인 아키텍처 공간 탐색을 위하여 머신 기술 언어를 이용한 컴파일드 코드 방식 시뮬레이터 생성시스템의 설계에 관해 기술한다. 구현된 시스템에서 생성하는 컴파일드 코드 방식 시뮬레이터는 event-driven 시뮬레이션의 높은 융통성과 성능 측정의 정확성을 유지하면서 빠른 시뮬레이션의 수행 속도를 갖기 위해, 인스트럭션 펫치와 디코딩 과정을 정적으로 결정하고 인스트럭션 execute 과정을 동적으로 시뮬레이션을 수행한다. 효율적인 아키텍처 탐색을 위해 설계 단계에 알맞은 시뮬레이션 속도와 성능 측정의 정확성을 가진 인스트럭션 수준과 사이클 수준에서의 시뮬레이션이 가능한 컴파일드 코드 방식 시뮬레이터를 생성한다. 보다 추상적인 인스트럭션 수준 시뮬레이터는 파이프라인 구조의 세부적인 동작을 고려하지 않으므로 빠른 시뮬레이션의 수행 속도를 가지며, 사이클 수준 시뮬레이터는 타겟 프로세서의 파이프라인 구조를 총체적으로 반영한 시뮬레이션을 수행하여 성능 측정의 정확성이 높다.

본 논문의 구성은 다음과 같다. 제2절에서는 event-driven 시뮬레이터에 기반한 설계 공간 탐색과 관련 개발 도구를 자동 생성하는 전체 framework를 설명하고, 제3절에서는 구축된 컴파일드 코드 방식 시뮬레이터 생성 시스템을 보이고, 시뮬레이션의 수행

속도와 성능 측정의 정확성을 고려한 컴파일드 코드 방식 시뮬레이션 흐름과 컴파일드 코드 방식 시뮬레이터 생성시스템에서 프로파일링에 대해 설명한다. 제4절에서는 제안된 시스템에서 생성한 컴파일드 코드 방식 시뮬레이터의 정확성과 유용성에 대한 실험 결과를 보이며, 마지막으로 제5절에서 결론 및 추후과제를 제시한다.

Ⅱ. 관련 연구 및 시스템 framework

2.1 관련 연구

머신 기술 언어로부터 시뮬레이터의 자동 생성 설계 기법과 설계 공간 탐색에 관한 많은 연구가 진행되었다. ISDL 언어로부터 생성한 타겟 프로세서의 구조적 정보와 행위 정보를 바탕으로 구축된 event-driven 시뮬레이터는 인스트럭션 펫치와 인스트럭션 디코딩, 그리고 execute 과정을 동적으로 시뮬레이션을 수행한다[24]. 구축된 시뮬레이터는 높은 시뮬레이션의 융통성을 보이나 발전된 프로세서 구조인 VLIW 프로세서에 대해 event-driven 시뮬레이션을 수행할 경우 인스트럭션 디코딩 과정에 많은 시간이 소요되어 제한된 설계 시간 내에 설계 검증이 어렵다. LISA 언어의 행위 기술 모델인 인스트럭션 수준과 사이클 수준에 따른 기술된 타겟 프로세서 모델로부터 생성한 시뮬레이터는 event-driven 시뮬레이션의 융통성을 유지하는 수준에서 인스트럭션 디코딩을 정적으로 결정하여 시뮬레이션의 수행 속도를 향상시키는 컴파일드 코드 방식 시뮬레이션을 적용하는 기법인 JIT-CCS (Just-In-Time Cache Compiled Simulation)을 수행한다[22]. JIT-CCS 기법은 인스트럭션 디코딩 과정의 정보에 대한 재사용성을 극대화시켜 정적으로 결정된 인스트럭션 디코딩 과정의 정보를 저장하는 데이터 공간을 많이 요구하는 컴파일드 코드 방식 시뮬레이션의 문제점을 해결하였으나, 제한된 데이터 공간으로 이상적인 시뮬레이션 과정을 제시하므로 보다 다양한 실제 프로세서에 대한 시뮬레이션에 한계가 보인다[20]. EXPRESSION 언어로부터 생성한 설계 모듈 라이브러리를 기반으로 구축된 시뮬레이터는 event- driven 시뮬레이션의 융통성을 유지하는 범위에서 시뮬레이션의 수행 속도를 향상시키는 최적화된 인스트럭션 행위 정보를 이용하여 시뮬레이션을 수행한다[20,21]. 구축된 시뮬레이터는 설계 공간 탐색을 위해 파이프라인 구조, 데이터패스 구조, 인스트럭션 셋, ILP (Instruction-Level Parallelism)을 향상시키는 설계 모듈, 메모리 구조에 대한 세부적인 시뮬레이션을 수행한다[19]. 설계 공간 탐색을 통해 어플리케이션에 최적화된 ASIP 설계가 가능하지만 불필요한 성능 측정 과정을 최소화하여 아키텍처 탐색의 성능을 더욱 향상할 수 있는 방법은 제안되지 않고 있다.

2.2 시스템의 Framework

SMDL 시스템은 ASIP 설계 자동화를 위해 구축된 시스템으로 머신 기술 언어 SMDL을 이용한 타겟 프로세서 기술로부터 임베디드 코어와 관련 컴파일러와 시뮬레이터를 자동 생성하는 임베디드 코어 생성기, SRCC (Sogang Retargetable Compiler Compiler), RISGen (Retargetable Instruction-set Simulator Generator)로 구성된다[13-15]. 설계자는 SMDL 시스템에서 지원하는 아키텍처 탐색을 수행하여 어플리케이션에 최적화된 임베디드 코어와 관련 개발 도구를 얻을 수 있다[16]. 그림 1은 SMDL 시스템의 개관을 보인다.

SMDL parser는 SMDL 언어로 타겟 프로세서를 기술된 코드를 입력으로 구문 분석 을 통해 타겟 프로세서의 리소스, 파이프라인, 인스트럭션 셋의 구조적 정보와 행위 정보, 그리고 타겟 프로세서와 외부의 디바이스에 대한 인터페이스 정보를 가진 IR (Intermediate Representative)인 프로세서 모델을 생성한다. 임베디드 코어 생성기, SRCC, RISGen은 프로세서 모델을 의미 분석을 수행하여 임베디드 코어와 관련 컴파일러, 시뮬레이터의 자동 생성에 필요한 정보를 얻는다. 임베디드 코어 생성기는 프로세서 모델로부터 파이프라인 스테이지 리소스 정보를 제약조건으로 파이프라인 스케줄링을 수행하여 데이터패스 모델과 파이프라인 스테이지에 대한 동작 정보를 가진 ACT (Active Component Table)[25]를 생성하며, 프로세서 모델과 ACT를 참조하여 임베디드 코어의

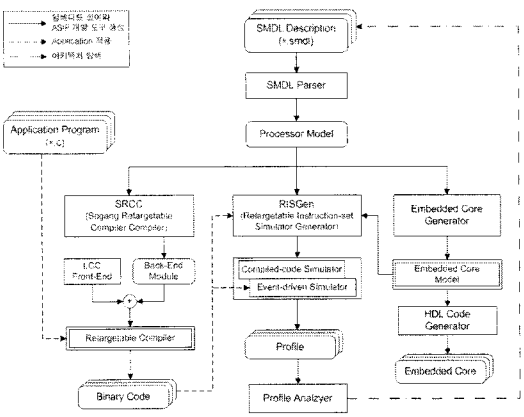


그림 1. SMDL 시스템.

파이프라인 정보, 데이터패스 정보, 기능유닛 간의 연결 정보, 그리고 전력 컨트롤 유닛 등의 정보를 가진 임베디드 코어 모델을 생성한다[15]. HDL 코드 생성기는 생성한 임베디드 코어 모델로부터 합성 가능한 HDL 코드로 변환하여 임베디드 코어를 얻는다. SRCC는 프로세서 모델에서 합성된 인스트럭션 셋, reservation table, 인스트럭션 latency table을 생성하여 컴파일러의 인스트럭션 선택기와 인스트럭션 스케줄러를 가진 back-end 모듈을 생성한다[13][26]. 생성된 back-end 모듈의 컴파일러 인스트럭션 스케줄러는 인스트럭션의 recoding을 고려한 스케줄링을 하므로 타겟 프로세서에 저전력 동작이 가능한 인스트럭션을 재구성한다[26]. RISGen은 프로세서 모델에서 시뮬레이션 커널, 운영체제에 동작하는 시스템 콜을 시뮬레이션하기 위한 시스템 콜 모델, trade-off 관계에 있는 시뮬레이션의 수행 속도와 성능 측정의 정확성에 대한 성능 측정 모델 등을 가진 RISGen 라이브러리를 생성하며, 생성된 RISGen 라이브러리와 임베디드 코어 모델을 참조하여 event-driven 시뮬레이터[14]와 컴파일드 코드 방식 시뮬레이터를 생성한다. SMDL 시스템의 아키텍처 탐색은 어플리케이션 프로그램의 정적 분석 결과와 시뮬레이션의 결과인 프로파일을 입력으로 한 프로파일 분석기를 통해 얻은 동적 분석 결과를 이용하여 어플리케이션에 최적화된 타겟 프로세서의 구조와 인스트럭션 셋을 결정하며, 다양한 컴파일러의 최적화 기법을 고려한 고성능 임베디드 코어를 얻는다[13].

Ⅲ. 제안된 컴파일드 코드 방식 시뮬레이터 생성 시스템

본 절에서는 제안된 컴파일드 코드 방식 시뮬레이터 생성 시스템의 개관, 인스트럭션 수준과 사이클 수준의 컴파일드 코드 방식 시뮬레이션 흐름, 그리고 컴파일드 코드 방식 시뮬레이션 흐름에서 프로파일링을 설명한다.

3.1 컴파일드 코드 방식 시뮬레이터 생성

컴파일드 코드 방식 시뮬레이터 생성 시스템은 프로세서 모델로부터 생성한 RISGen 라이브러리와 임베디드 코어 모델, 그리고 어플리케이션 프로그램을 retargetable 컴파일러에 의해 변환된 바이너리 코드를 참조하여 성능 측정 모델에 해당되는 컴파일드 코드 방식 시뮬레이터를 생성한다. 성능 측정 모델은 서로 trade-off 관계에 있는 시뮬레이션의 수행 속도와 성능 측정의 정확성을 고려한 basic-block 수준, 인스트럭션 수준과 사이클 수준에 알맞은 성능 측정할 설계 모듈을 가진다. Basic-block 수준은 컴파일러의 인스트럭션 스케줄링 결과를 이용하여 보다 넓은 설계 공간인 프로세서 리소스와 프로세서 구조에 대해 빠르게 프로파일을 얻는다. 인스트럭션 수준은 파이프라인 구조에 대한 세부적인 프로파일을 제외한 기능 유닛, 레지스터 파일, 인스트럭션 셋, 데이터패스 구조 등에 대해 프로파일을 얻으며, 사이클 수준은 인스트럭션 수준보다 더욱 구체적인 산술 논리 유닛, 분기 처리 유닛, 레지스터 크기, 레지스터 파일 정보, 인스트럭션 행위 정보, 어드레싱 모드, 파이프라인의 세부적인 구조 등에 대한 프로파일을 얻는다. 그림 2는 컴파일드 코드 방식 시뮬레이터 생성 시스템을 보인다.

제안된 시뮬레이터 생성기는 프로세서 모델을 참조하여 생성한 인스트럭션 디코딩 모듈, RISGen 라이브러리, 그리고 컴파일드 코드 방식 시뮬레이션 엔진은 event-driven 시뮬레이션이 가지는 장점을 유지하면서 빠른 시뮬레이션의 수행 속도를 위해 인스트럭션 펫치와 디코딩 과정을 정적으로 결정한다. 인스트럭션 디코딩 모듈은 바이너리 코드를 입력으로 펫치된 인스트럭션의 바이너리 포맷 정보와 필드별 정보를 이용하여 동적으로 시뮬레이션 수행 시에 관련 인스트럭션의 행위 정보와 오퍼랜드의 값을 가진 디코딩된 인스트럭션을 생성한다. 성능 측정 모델은 사이클 수준 시뮬레이터와 인스트럭션 수준 시뮬레이터의 생성을 위한 정보[14]를 가진다. 인스트럭션 수준 시뮬레이터 생성을 위한 성능 측정 모델은 단일 인스트럭션이 타겟 프로세서의 파이프라인 스테이지에 따른 행위 정보 등을 가지며, 사이클 수준 시뮬레이터 경우는 데이터 해저드와 컨트롤 해저드를 해결하는 기법에 따른 파이프라인의 세부적인 구조와 스테이지 행위에

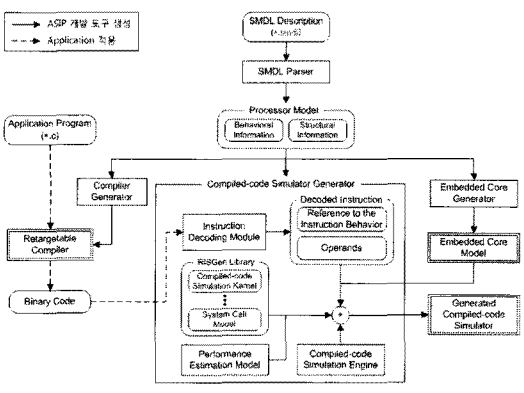


그림 2. 컴파일드 코드 방식 시뮬레이터 생성 시스템.

대한 정보 등을 가진다. 디코딩된 인스트럭션, RISGen 라이브러리, 컴파일드 코드 기반 시뮬레이션 엔진, 그리고 임베디드 코어 모델을 참조하여 컴파일드 코드 방식 시뮬레이터를 생성된다.

3.2 성능 측정 모델을 포함한 컴파일드 코드방식 시뮬레이션 흐름

생성된 컴파일드 코드 방식 시뮬레이터는 trade-off 관계에 있는 시뮬레이션의 수행 속도와 성능 측정의 정확성에 대한 성능 측정 모델을 고려하였다. 그림 3은 제안된 컴파일드 코드 방식 시뮬레이션 흐름을 보인다. 그림 3 (a)과 (b)는 각각 인스트럭션 수준과 사이클 수준의 컴파일드 코드 방식 시뮬레이션 흐름을 보인다. 인스트럭션 수준의 시뮬레이터는 파이프라인 구조 정보를 이용한 인스트럭션 수행 모듈에서 단일 인스트럭션이 파이프라인 구조에 동작하고 파이프라인 해저드를 고려하지 않은 시뮬레이션을 수행하며, 빠른 시뮬레이션의 수행 속도를 가지므로 파이프라인 구조를 고려하지 않은 프로파일을 얻을 경우에 유용하다. 인스트럭션 수준의 시뮬레이션 흐름은 인스트럭션 메모리에 저장된 디코딩된 인스트럭션의 참조할 인스트럭션 행위 정보와 오퍼랜드 값을 이용하여 해당 인스트럭션 행위에서 인스트럭션 펫치와 디코딩 과정이 완료된 파이프라인 스테이지 상태로 초기화하며, 인스트럭션 수행 모듈에서 관련 인스트럭션 행위와 프로세서 리소스의 현재 결과를 바탕으로 단 한 번

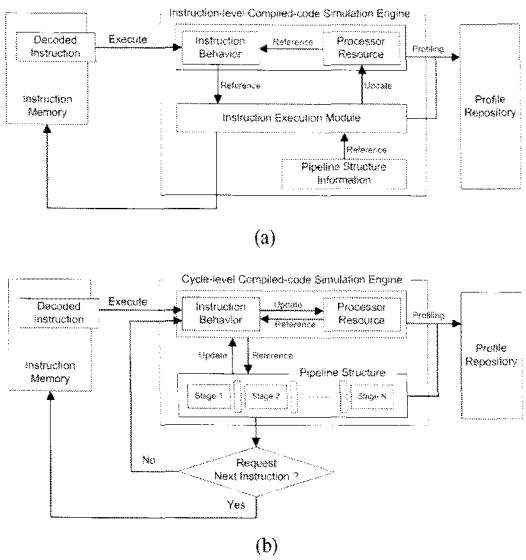


그림 3. 제안된 컴파일드 코드 방식 시뮬레이션 흐름. (a) 인스트럭션 수준 컴파일드 코드 방식 시뮬레이션 흐름, (b) 사이클 수준 컴파일드 코드 방식 시뮬레이션 흐름.

에 파이프라인의 최종 스테이지까지 동적으로 시뮬레이션을 수행한다. 그 결과를 프로세서 리소스에 반영하며 업데이트된 프로세서 리소스와 파이프라인 구조에 대해 프로파일링을 하여 프로파일 저장소에 저장하고, 인스트럭션 메모리에서 다음 프로그램 카운터 값에 해당되는 디코딩된 인스트럭션을 위와 같이 동적으로 시뮬레이션을 수행한다. 사이클 수준의 시뮬레이터는 정확한 성능 측정을 위해 타겟 프로세서의 파이프라인 구조를 총체적으로 반영하는 시뮬레이션 흐름을 가진다. 사이클 수준의 시뮬레이션 흐름은 인스트럭션 수준의 시뮬레이션과 동일한 초기화 과정을 완료하며, 해당 인스트럭션 행위에 관련 파이프라인 구조의 구체적인 동작을 동적으로 시뮬레이션을 수행하여 프로세서 리소스의 값과 파이프라인 구조의 레지스터 값을 업데이트한다. 하나의 파이프라인 스테이지를 동작한 파이프라인 구조와 프로세서 리소스에 대한 프로파일링을 하여 프로파일 저장소에 저장한다. 파이프라인에 동작 가능한 슬롯이 있을 경우는 인스트럭션 메모리에서 다음 프로그램 카운터 값에 해당되는 디코딩된 인스트럭션을 위와 같이 동적으로 시뮬레이션을 수행하고, 그렇지 않으면 파이프라인의 모든 슬롯들이 다음 파이프라인 스테이지로 시뮬레이션을 수행한다.

3.3 컴파일드 코드 방식 시뮬레이터 생성시스템에서 프로파일링

인스트럭션 수준과 사이클 수준의 컴파일드 코드 방식 시뮬레이터 생성시스템에서 프로파일러는 코어의 설계 수준에 알맞은 프로파일을 생성하여 불필요한 프로파일링 과정과 시간을 최소화할 수 있다. 그림 4는 컴파일드 코드 방식 시뮬레이터 생성시스템에서 프로파일링을 보인다. 프로세서 모델로부터 인스트럭션 수준과 사이클 수준의 성능 측정 정보를 추출한 IR (Intermediate Representation)은 인스트럭션 사이클 테이블, 인스트럭션 지연 테이블, 파이프라인 리저베이션 테이블, 파이프라인 해저드를 해결하는 기법에 따른 성능 측정 매개변수 등으로 구성된다. 인스트럭션 수준의 프로파일러를 위한 IR은 인스트럭션 사이클 테이블, 인스트럭션 지연 테이블, 파이프라인 리저베이션 테이블 등이며, 파이프라인 세부적인 구조를 고려하지 않은 개괄적인 파이프라인 구조에 대한 프로파일을 생성한다. 인스트럭션 사이클 테이블은 단일 인스트럭션이 타겟 프로세서의 파이프라인에 동작하는 사이클 값을 가지며, 인스트럭션 지연 테이블은 RAW (Read After Write), WAW (Write After

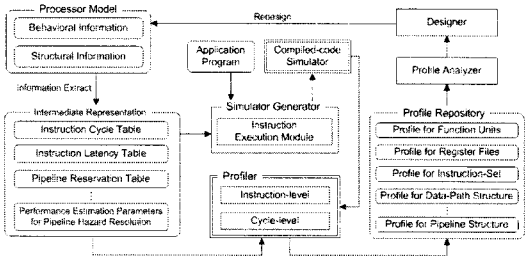


그림 4. 컴파일드 코드 방식 시뮬레이터 생성시스템에서 프로파일링.

Write), WAR (Write After Read) 해저드를 해결하기 위한 인스트럭션의 지연 사이클 값을 가지며, 파이프라인 리저베이션 테이블은 단일 인스트럭션이 파이프라인 스테이지의 행위에 따른 프로세서 리소스 정보를 가진다.

인스트럭션 수준의 프로파일러는 현재 시뮬레이션 상태의 인스트럭션 수행 모듈과 프로세서 모델의 리소스 정보, 인스트럭션 행위의 리소스 사용 정보, 그리고 인스트럭션 수준의 프로파일러를 위한 IR의 성능 측정 정보를 바탕으로 기능 유닛, 레지스터 파일, 인스트럭션 셋, 데이터패스 구조 등에 대한 성능 데이터를 얻는다. 이전 단계에 시뮬레이션이 완료된 인스트럭션 정보와 인스트럭션 사이클 테이블, 인스트럭션 지연 테이블, 파이프라인 리저베이션 테이블을 이용하여 파이프라인의 데이터 해저드가 일어나지 않은 범위에서 단일 인스트럭션의 최종 사이클 값을 구하여 이전 단계에 해당되는 임베디드 코어의 전체 수행 사이클 값에 더한다. 생성된 성능 데이터는 프로파일 저장소에서 임베디드 코어의 설계 모듈에 따른 프로파일 항목별로 저장된다. 사이클 수준의 프로파일러를 위한 IR은 데이터 해저드와 컨트롤 해저드를 해결하는 기법에 따른 파이프라인의 세부적인 구조와 프로세서 리소스에 대한 성능 측정 매개변수 등으로 구성된다. 사이클 수준의 프로파일러는 하나의 파이프라인 스테이지에 대한 사이클 수준의 시뮬레이션 상태에서 파이프라인 구조와 프로세서 모델의 리소스 정보, 인스트럭션 행위의 리소스 사용 정보, 그리고 사이클 수준의 프로파일러를 위한 IR의 성능 측정 정보를 바탕으로 산술 논리 유닛, 분기 처리 유닛, 레지스터 크기, 레지스터 파일 정보, 인스트럭션 행위 정보, 어드레싱 모드, 파이프라인 스테이지에 따른 구조 등에 대한 성능 데이터를 얻는다. 생성된 성능 데이터는 인스트럭션 수준의 프로파일러가 생성한 성능 데이터에 비해 보다 다양하고 세부적인 임베디드 코어의 설계 모듈에 대해 성능 측정되었으며 프로파일 저장소에서 계층적인 설계 모듈의 프로파일 항목별로 구성할 수 있도록 저장된다. 프로파일 저장소의 프로파일을 입력으로 한 프로파일 분석기는 코어의 설계 모듈에 관련 있는 성능 데이터를 분석한 결과를 생성하며, 설계자는 이 결과를 바탕으로 코어의 성능이 향상시킬 수 있도록 타겟 프로세서를 재설계한다.

Ⅳ. 실험 결과

구축된 컴파일드 코드 방식 시뮬레이터 자동 생성 시스템에서 생성한 시뮬레이터의 정확성과 유용성을 검증하기 위해 MIPS R3000 프로세서를 SMDL 언어로 기술하여 프로세서로 사용하였으며, 점차 최적화된 프로세서를 생성하는 과정을 밟았다[27].

표 1은 MIPS R3000의 초기 임베디드 코어와 아키텍처 탐색을 수행하여 JPEG 인코더에 최적화된 임베디드 코어에 대해 시뮬레이터에 따른 JPEG 인코더 프로그램 수행시 필요 사이클 수를 보인다[16]. 최종 임베디드 코어의 수행 사이클 결과를 통해 SMDL 시스템은 아키텍처 탐색에 효율적임을 보인다.

표 2는 아키텍처 탐색시 시뮬레이터 방식에 따른 시뮬레이션의 평균 수행 시간을 보인다. 컴파일드 코

표 1. 임베디드 코어의 수행 성능.

	MIPS R3000의 초기 임베디드 코어	JPEG 인코더에 최적화된 임베디드 코어	비고
JPEG 인코더 프로그램 수행시 필요 사이클 수	72,990,212	36,192,147	-50.4%

표 2. 아키텍처 탐색시 시뮬레이션 방식에 따른 시뮬레이션의 평균 수행 시간.

사용된 시뮬레이터	MIPS R3000의 초기 임베디드 코어	JPEG 인코더에 최적화된 임베디드 코어	비고
Event-driven 시뮬레이터	6,007.4 ms	2,969.8 ms	-50.6%
인스트럭션 수준 컴파일드 코드 방식 시뮬레이터	1,300.6 ms	641.7 ms	-50.6%
사이클 수준 컴파일드 코드 방식 시뮬레이터	4,705.1 ms	2,325.9 ms	-50.7%

드 방식 시뮬레이터는 event-driven 시뮬레이터에 비해 빠르게 시뮬레이션을 수행하며, 인스트럭션 수준의 시뮬레이터는 파이프라인의 세부적인 구조를 고려하지 않으므로 사이클 수준의 시뮬레이터보다 빠른 수행 속도를 가진다.

그림 5는 JPEG 인코더에 최적화된 최종 임베디드 코어를 생성될 때까지 시뮬레이션의 평균 수행 시간 비교를 보인다. Event-driven 시뮬레이션의 평균 수행 시간을 100로 놓고 인스트럭션 수준 컴파일드 코드 방식 시뮬레이션과 사이클 수준 컴파일드 코드 방식 시뮬레이션의 평균 수행 속도를 구하였다. 인스트럭션 수준 컴파일드 코드 방식 시뮬레이터는 event-driven 시뮬레이터에 비해 평균 78.4% 시뮬레이션 시간이 감소하였고, 사이클 수준 컴파일드 코드 방식 시뮬레이터는 event-driven 시뮬레이터에 비해 평균 21.7% 시뮬레이션 시간이 감소하였다.

JPEG 인코더에 최적화된 임베디드 코어 설계를 위한 아키텍처 탐색의 결과를 통해 사이클 수준 컴파일드 코드 방식 시뮬레이터는 동적으로 동작하는 인스트럭션 행위와 운영체제의 시스템 콜 등을 예측 가능한 범위 내에 구축된 RISGen 라이브러리를 이용하여 인스트럭션 펫치와 디코딩 과정을 정적으로 결정하였다. 정적으로 결정된 인스트럭션 펫치와 디코딩 과정에 기반한 동적으로 시뮬레이션의 결과는 event-driven 시뮬레이터의 정확성과 융통성을 유지하며 평균 21.7% 시뮬레이션 수행 속도가 향상되었다. 인스트럭션 수준 컴파일드 코드 방식 시뮬레이터는 세부적인 파이프라인 구조를 제외한 기능 유닛, 데이터패스, 인스트럭션 셋 등에 대한 프로파일을 얻으므로 임베디드 코어의 설계에 한계가 있으나, 사이클 수준 컴파일드 코드 방식 시뮬레이터의 느린 시뮬레이션의 수행 속도를 보완할 수 있다. 성능 측정 모델을 고려한 컴파일드 코드 방식 시뮬레이터는 체계적인

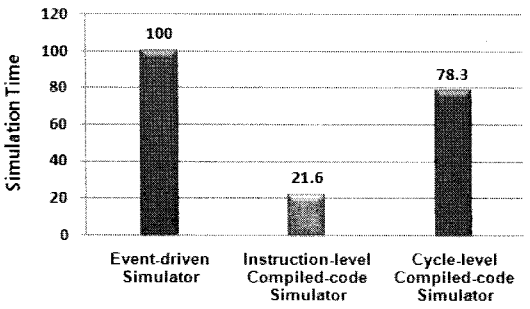


그림 5. 최적화된 임베디드 코어 설계에서 시뮬레이터별 시뮬레이션의 평균 수행 시간.

아키텍처 탐색의 단계에 맞게 이용하면 불필요한 성능 측정과정을 최소화할 수 있으므로 어플리케이션에 최적화된 임베디드 코어의 설계를 위한 아키텍처 탐색의 성능 향상이 가능하다.

V. 결론 및 추후과제

본 논문은 event-driven 시뮬레이터의 느린 시뮬레이션의 수행 속도를 보완하고 성능 측정과정을 설계 수준에 맞추어 효율적으로 수행할 수 있도록 성능 측정 모델을 포함한 컴파일드 코드 방식 시뮬레이터 생성 시스템을 제안하였다. 생성된 컴파일드 코드 방식 시뮬레이터는 event-driven 시뮬레이터의 정확성과 융통성을 유지할 수 있는 범위 내에 RISGen 라이브러리를 이용하여 인스트럭션 펫치와 디코딩 과정을 정적으로 결정하여 빠른 시뮬레이션의 수행 속도를 가진다. 서로 trade-off 관계가 있는 시뮬레이션의 수행 속도와 성능 측정의 정확성을 고려한 성능 측정 모델을 채택하며 인스트럭션 수준과 사이클 수준의 시뮬레이션 수행과 프로파일링이 가능하다. 구축된 컴파일드 코드 방식 시뮬레이터는 ARM9 프로세서와 MIPS R3000 프로세서에서 JPEG 인코더에 대한 시뮬레이션을 수행하여 시뮬레이션의 정확성을 검증하였다. JPEG 인코더에 최적화된 임베디드 코어 설계를 위해 전용 시뮬레이터로 사용된 event-driven 시뮬레이터, 인스트럭션 수준과 사이클 수준의 컴파일드 코드 방식 시뮬레이터를 이용하여 아키텍처 탐색을 수행하였으며, 그 결과는 사이클 수준 컴파일드 코드 방식 시뮬레이터는 event-driven 시뮬레이터의 정확성과 융통성을 유지하면서 그에 비해 평균 21.7% 시뮬레이션 수행 속도가 향상되었다. 인스트럭션 수준 컴파일드 코드 방식 시뮬레이터는 사이클 수준 컴파일드 코드 방식 시뮬레이터에 비해 시뮬레이션의 정확성이 떨어지나 빠른 성능 측정이 가능하므로 파이프라인의 세부적인 구조를 고려하지 않은 아키텍처 탐색의 설계 단계에 사용하면 보다 빠르게 어플리케이션에 최적화된 임베디드 코어를 설계할 수 있다.

추후 과제는 설계된 임베디드 코어와 특정 어플리케이션에 최적화된 IP 등의 동시 시뮬레이션을 수행할 수 있는 시뮬레이터 생성 시스템으로 확장하는 연구가 필요하다.

참 고 문 헌

[1] J. Rabaey and M. Pedram, Eds., *Low Power*

Design Methodologies, Kluwer Academic Pub., 1996.

[2] S. Abdi, D. Shin, and D. Gajski, "Automatic Communication Refinement for System Level Design", *in Proc. Design Automation Conference*, Anaheim, CA, pp.300-305, June 2003.

[3] M. Jain, M. Balakrishnan, and A. Kumar, "ASIP Design Methodologies : Survey and Issues", *in Proc. IEEE/ACM Int. Conf. VLSI Design. (VLSI 2001)*, Bangalore, India, pp.76-81, Jan. 2001.

[4] C. Liem, *Retargetable Compilers for Embedded Core Processors*, Kluwer Academic Pub., 1997.

[5] R. Leupers, "Compiler Design Issues for Embedded Processors", *IEEE Design & Test of Computers*, Vol.19, No.4, pp.51-58, July/Aug. 2002.

[6] R. Gonzalez, "Xtensa : A Configurable and Extensible Processor", *IEEE Micro*, Vol.20, No.2, pp.60-70, March/April 2000.

[7] A. Fauth, M. Fredericks, and A. Knoll, "Generation of Hardware Machine Models from Instruction Set Descriptions", *in Proc. IEEE Workshop VLSI Signal Processing*, Veldhoven, Netherlands, pp.242-250, Oct. 1993.

[8] M. Itoh et al. "Synthesizable HDL Generation for Pipelined Processors from a Micro-Operation Description", *IEICE Trans. on Fundamentals of Electronics Communications and Computer Sciences*, Vol.E83-A, No.3, pp.394-400, Mar. 2000.

[9] O. Schliebusch et al, "A Novel Methodology for the Design of Application-Specific Instruction Set Processors Using a Machine Description Language", *IEEE Trans. Computer-Aided Design*, Vol.20, No.11, pp.1338-1354, Nov. 2001.

[10] P. Mishra, A. Kejariwal, and N. Dutt, "Rapid Exploration of Pipelined Processors through Automatic Generation of Synthesizable RTL Model", *in Proc. IEEE Int. Workshop on Rapid System Prototyping*, San Diego, CA, pp.226-232, June 2003.

[11] P. Marwedel and G. Goossens, *Code Generation for Embedded Processors*, Kluwer Academic Pub., 1995.

[12] M. Jacome and G. Veciana, "Design Challenges for New Application-Specific Processors", *IEEE Design & Test of Computers*, Vol.17, No.2, pp.40-50, April/June 2000.

[13] 이성래, 황선영, "머쉰 행위기술로부터 Retargetable 컴파일러 생성시스템 구축", 한국통신학회논문지(네트워크 및 서비스), 32권 5호, pp.286-294, 2007년 5월.

[14] 홍성민, 박창수, 황선영, "DSP 프로세서용 인스트럭션 셋 시뮬레이터 자동생성기의 설계에 관한 연구", 한국통신학회논문지(무선통신), 32권 9호, pp.931-939, 2007년 9월.

[15] 조재범, 유용호, 황선영, "임베디드 프로세서 코어 자동생성 시스템의 구축", 한국통신학회논문지, 30권 6A호, pp.526-534, 2005년 6월.

[16] 이성래, 황선영, "Application에 최적의 ASIP 설계를 위한 효율적인 Architecture Exploration 방법", 한국통신학회논문지(통신이론 및 시스템), 32권 9호, pp.913-921, 2007년 9월.

[17] T. Kempf, K. Karuri, S. Wallentowitz, G. Ascheid, R. Leupers, and H. Meyr, "A SW Performance Estimation Framework for Early System-Level-Design Using Fine-grained Instrumentation", *in Proc. Conf. Design Automation and Test in Europe*, Munich, Germany, pp.468-473, March 2006.

[18] P. Hallschmid and R. Saleh, "Fast Design Space Exploration Using Local Regression Modeling With Application to ASIPs", *IEEE Trans. Computer-Aided Design*, Vol 27, No.3, pp.508-515, March 2008.

[19] A. Khare, N. Savoiu, A. Halambi, P. Grun, N. Dutt, and A. Nicolau, "V-SAT: A Visual Specification and Analysis Tool for System-On-Chip Exploration", *in Proc. EUROMICRO Conf.*, Vol.1, Milan, Italy, pp.196-203, Sept. 1999.

[20] M. Reshadi, P. Mishra, and N. Dutt, "Instruction Set Compiled Simulation: A Technique for Fast and Flexible Instruction Set Simulation", *in Proc. Design Automation Conference*, Anaheim, CA, pp.758-763, June 2003.

[21] M. Reshiadi, N. Bansal, P. Mishra, and N. Dutt,

"An Efficient Retargetable Framework for Instruction-Set Simulation", *in Proc. IEEE/ACM/IFIP Int. Conf. Hardware /Software Codesign & System Synthesis*, Newport Beach, CA, pp.13-18, Oct. 2003.

[22] A. Nohl, G. Braun, O. Schliebusch, R. Leupers, H. Meyr, and A. Hoffmann, "A Universal Technique for Fast and Flexible Instruction-Set Architecture Simulation", *in Proc. Design Automation Conference*, New Orleans, LA, pp.22-27, June 2002.

[23] M. Gries and K. Keutzer, Eds., *Building ASIPs: The Mescal Methodology*, Springer, 2005.

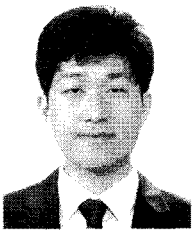
[24] G. Hadjiyiannis and S.Devadas, "Techniques for Accurate Performance Evaluation in Architecture Exploration", *IEEE Trans. VLSI Systems*, Vol.11, No.4, pp.601-615, Aug. 2003.

[25] 이해동, 황선영, "파이프라인 데이타패스 자동 생성을 위한 상위수준 합성 시스템의 설계", 대한전자공학회 논문지, 31-A권 3호, pp.290-304, 1994년 3월.

[26] S. Lee, S. Lee, and S. Hwang, "A Concurrent Instruction Scheduling and Recoding Algorithm for Power Minimization in Embedded Systems", *IEICE Transactions on Information and Systems*, Vol.93-D, No.8, Aug 2010.

[27] G. Kane and J. Heinrich, *MIPS RISC Architecture*, Prentice Hall, 1992.

김 상 우 (Sang-Woo Kim) 준회원



2009년 2월 서강대학교 전자공학과 졸업

2009년 3월~현재 서강대학교 전자공학과 석사과정

<관심분야> ASIP Design, Retargetable Compiler for Embedded System

황 선 영 (Sun-Young Hwang) 정회원

1976년 2월 서울대학교 전자공학과 졸업

1976년 2월 한국과학원 전기 및 전자공학과 공학석사 취득

1986년 10월 미국 Stanford 대학교 전자공학 박사학위 취득

1976년~1981년 삼성 반도체(주) 연구원, 팀장

1986년~1989년 Stanford 대학 Center for Integrated Systems 연구소 책임 연구원 및 Fairchild Semiconductor, Palo Alto Research Center 기술자문

1989년~1992년 삼성전자(주) 반도체 기술자문

2002년 4월~2004년 3월 서강대학교 정보통신대학원장

1989년 3월~현재 서강대학교 전자공학과 교수

<관심분야> SoC 설계 및 framework 구성, CAD 시스템, Embedded 시스템, DSP 시스템 설계 등