DSP 프로세서용 인스트럭션 셋 시뮬레이터 자동생성기의 설계에 관한 연구

준회원 홍 성 민*, 박 창 수*, 정회원 황 선 영*

Design of an Automatic Generation System for Cycle-accurate Instruction-set Simulators for DSP Processors

Sung-Min Hong*, Chang-Soo Park* *Associate Members*, Sun-Young Hwang* *Regular Member*

요 약

본 논문은 SMDL (Sogang Machine Description Language)을 이용한 DSP 프로세서용 인스트럭션 셋 시뮬레이터 자동 생성기 시스템의 설계에 관해 기술한다. SMDL은 DSP 어플리케이션에 최적화된 아키텍처를 포함한 임베디드 코어의 효율적 기술을 위한 머신 기술 언어로서, 구현된 인스트럭션 셋 시뮬레이터 자동 생성 시스템은 타겟 ASIP의 SMDL 기술을 입력으로 하여 인스트럭션들의 파이프라인 스테이지 별 행위 정보를 분석한 후 cycle-accurate 인스트럭션 셋 시뮬레이터를 C++ 파일로 자동 생성한다. 구현된 자동 생성 시스템의 검증을 위해 ARM9E-S, ADSP-TS20x와 TMS320C2x 아키텍처들을 SMDL로 기술하여 시뮬레이터들을 자동 생성하였으며, 생성된 시뮬레이터들을 이용하여 4X4 매트릭스 곱셈, 16비트 IIR 필터, 32비트 곱셈, 그리고 FFT에 연산에 대한 시뮬레이션을 수행하였다. 결과 생성된 시뮬레이터의 정확한 동작을 확인하였다.

Key Words : MDL, DSP, ASIP, Automatic Generation, Instruction-set Simulator

ABSTRACT

This paper describes the system which automatically generates instruction-set simulators cores using the SMDL. SMDL describes structure and instruction-set information of a target DSP machine. Analyzing behavioral information of each pipeline stage of all instructions on a target ASIPs, the proposed system automatically generates a cycle-accurate instruction set simulator in C++ for a target processor. The proposed system has been tested by generating instruction-set simulators for ARM9E-S, ADSP-TS20x, and TMS320C2x architectures. Experiments were performed by checking the functions of the 4x4 matrix multiplication, 16-bit IIR filter, 32-bit multiplication, and the FFT using the generated simulators. Experimental results show the functional accuracy of the generated simulators.

I. 서 론

최근 휴대용 전자 제품들은 하나의 시스템 안에 무선 인터넷, 이동 통신, 음원, 그리고 동영상 파일 재생 등과 같은 다양한 기능을 지원하는 특징을 가지고 있어 시스템 설계 복잡도가 증가되고 있다. 또한 제품의 life cycle이 짧아져 빠른 시장 진입이 요구된다. 이러한 복잡한 시스템 개발의 time to market을 만족시키기 위해서는 SoC (System-on-a-Chip) 설계 기술을 필요로 한다[1].

※ 본 연구는 정보통신부 및 정보통신연구진흥원의 대학 IT 연구센터 육성지원사업의 연구결과로 수행되었으며 Simulation 및 synthesis에 IDEC에서 지원받은 CAD tool을 사용하였음.
* 서강대학교 전자공학과 대학원 CAD & ES연구실
논문번호 : KICS2007-01-009, 접수일자 : 2007년 1월 22일, 최종논문접수일자 : 2007년 7월 31일

복잡한 시스템의 SoC 설계에는 다양한 어플리케이션의 구현, 설계 유연성과 재사용률을 높이기 위해 두 개 이상의 프로세서를 쓰는 MPSoC (Multiple-Processor System-on-a-Chip) 설계 방식이 이용되고 있다[2]. MPSoC에는 시스템 전체 제어를 담당하는 범용 임베디드 프로세서(MIPS, ARM 등)와 각 어플리케이션 수행을 위해 특화된 프로세서들로 구성된다[2]. 어플리케이션 수행을 위한 프로세서들은 제한된 소비전력과 비용으로 최대의 성능을 만족시키기 위해 어플리케이션에 최적화된 인스트럭션 셋을 가지는 ASIP (Application Specific Instruction-set Processors)이 사용된다[3,4]. ASIP 설계에는 코어의 하드웨어 모델과 수행될 어플리케이션의 개발과 검증을 위한 임베디드 소프트웨어 툴 셋 구축이 필요하여 매뉴얼 설계 시 많은 시간과 비용이 소모된다[5]. 이에 따라 시장의 요구를 만족시키면서 어플리케이션에 최적화된 코어를 개발하기 위해서는 ASIP 소프트웨어 툴 셋 설계의 자동화가 요구된다. ASIP 설계 자동화 방식에는 라이브러리에서 제공 받은 코어 스켈레톤과 인스트럭션 셋에서 설계자가 파라미터를 조절하여 ASIP을 생성하는 파라미터 기반의 ASIP 설계 방식[6]과, 설계자가 타겟 코어의 인스트럭션 셋을 MDL (Machine Description Language)로 기술하여 ASIP을 자동생성하는 MDL 기반의 ASIP 설계 방식[5,7,8,9,10]이 제안되었다. 파라미터 기반의 ASIP 설계 방식은 생성된 ASIP이 라이브러리에서 제공 받은 아키텍처 스켈레톤에 최적화된 장점을 가지고 있으나 라이브러리의 범주를 벗어나는 다양한 아키텍처를 생성할 수 없는 단점을 가지고 있다. MDL 기반의 ASIP 설계 방식은 MDL 기술을 통한 다양한 아키텍처의 생성이 가능하지만 생성될 코어의 HDL 모델과 컴파일러에 대한 최적화가 어려운 단점을 가지고 있다[11]. 다양한 ASIP의 개발을 위해서는 타겟 프로세서에 대한 효율적인 기술이 가능한 MDL 기반의 ASIP 설계 방법이 적합하다. 사용될 MDL은 다양한 아키텍처에 대한 기술이 가능하며 임베디드 소프트웨어 툴 셋 생성과 최적화를 위한 충분한 정보를 포함하고 있어야 한다.

본 논문에는 다양한 아키텍처에 대한 상위 수준의 효율적 기술이 가능한 SMDL (Sogang Machine Description Language)을 이용한 DSP cycle-accurate 인스트럭션 셋 시뮬레이터 자동생성 기법에 대해 제안한다. SMDL은 서강대학교에서 개발한 언어로서 DSP 어플리케이션에 최적화된 아키텍처를 포함한 임베디드 코어의 기술을 위한 머신 기술 언어이다. 논문의 Ⅱ장에는 MDL 관련 연구들을 기술하며, Ⅲ장에는 SMDL 언어의 특징과 SMDL을 이용한 ASIP 설계 자동화 시스템에 대해 보인다. Ⅳ장에는 인스트럭션 셋 시뮬레이터 생성 기법을 제안한다. 타겟 코어의 SMDL 기술 정보는 IF (Intermediate Form) 로의 변환을 통해 자동생성에 적합한 IF로 변경되며, ISSGen (Instruction-Set Simulator Generator) 시스템은 IF를 입력으로 하여 코어의 cycle-accurate 인스트럭션 셋 시뮬레이터를 C++ 형태로 자동 생성한다. Ⅴ장에는 생성된 시뮬레이터를 이용한 실험 결과를 보인다. 구현된 자동생성 시스템 검증을 위해 ARM9E-S[12], ADSP-TS20x[13], TMS320C2x[14] 아키텍처들을 SMDL로 기술하고, ISSGen을 통해 시뮬레이터를 생성하였다. 생성된 시뮬레이터의 검증을 위해 4X4 매트릭스 곱셈, 16비트 IIR 필터, 32비트 곱셈, 그리고 FFT 연산에 대한 시뮬레이션을 수행하였다. 끝으로 Ⅵ장에는 결론 및 추후 연구 과제를 제시한다.

Ⅱ. 관련 연구

MDL 기반의 ASIP 설계 자동화 기법에는 최근까지 많은 연구가 진행되었다. 기존에 개발된 MDL은 Technical University of Berlin의 nML[15,16], Aachen University of Technology의 LISA[5,17], UCI의 EXPRESSION[9], 오사카 대학에서 개발한 Micro Operation[10]등이 있다. nML은 리타겟터블 코드 생성기인 CHESS[7] 시스템과 인스트럭션 셋 시뮬레이터 CHECKERS 시스템 등에서 사용된 MDL이며, 타겟 코어의 내부 구조 정보와 인스트럭션/어드레싱 모드의 행위 정보 기술을 포함하여 특수한 어드레싱 모드에 대한 기술이 가능하다[15]. 하지만 파이프라인 기술과 실행 유닛 사용 정보가 없어 cycle-accurate 시뮬레이션을 지원하는 시뮬레이터의 생성이 불가능한 단점이 있다. LISA는 행위 기술에 중점을 둔 머신기술 언어로서 각 동작들을 연산단위로 기술하고 정의된 연산들을 합쳐 새로운 연산을 정의하는 방식을 취하고 있으며, 인스트럭션의 동작을 파이프라인 스테이지 단위로 기술하므로 cycle-accurate 시뮬레이터의 생성이 가능하다[17,18,19]. 그러나 파이프라인 기술을 위한 사용자의 기술양이 많은 단점을 가지고 있다. EXPRESSION은 기존에 검증된 코어와 메모리 IP 정보를 이용한 타겟 프로세서 기술로부터,

cycle-accurate 인스트럭션 셋 시뮬레이터와 컴파일러등의 툴 셋을 자동 생성한다[20]. 그러나 타겟 프로세서의 버스 연결정보와 파이프라인 레지스터와 같은 구조정보, RTL 수준의 포트 정보 등의 구체적인 기술이 요구되어 기술 효율성이 낮으며, 검증된 코어 기반의 기술만을 요구하므로 다양한 아키텍처의 기술이 불가능하다[21]. Micro Operation은 타겟 코어의 RTL 기술로부터 코어를 생성하는 방식을 취하고 있어 낮은 수준의 구체적인 기술로 인해 기술양이 많으며, 인스트럭션 셋 시뮬레이터 생성을 지원하지 않는다[10].

Ⅲ. SMDL 시스템

3.1 SMDL 언어

제안한 인스트럭션 셋 시뮬레이터 생성기의 입력 언어인 SMDL은 타겟 아키텍처의 structure 기술을 하는 내/외부 구조 기술부와 behavior 기술을 하는 인스트럭션 셋 기술부로 나뉜다. 그림 1에 SMDL의 구조를 보인다.

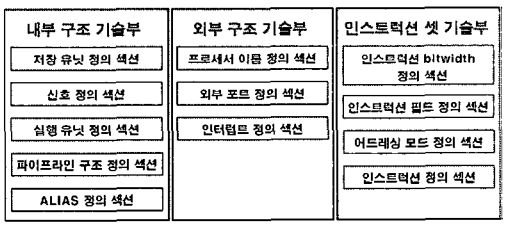


그림 1. 인터페이스 생성기 흐름도

SMDL의 내부 구조 기술부에는 타겟 프로세서의 내부 리소스들 및 아키텍처를 정의한다. 내부 구조 기술은 저장 유닛, 신호, 실행 유닛 정의 섹션, 파이프라인 구조 정의 섹션, 그리고 alias 정의 섹션으로 나뉜다. 외부 구조 기술은 프로세서 이름 정의 섹션, 외부 포트 및 인터럽트 정의 섹션으로 나뉜다. SMDL의 사용자는 내/외부 기술을 바탕으로 하여 타겟 코어의 인스트럭션 셋을 기술한다. 인스트럭션 셋 기술부는 인스트럭션 bit-width와 field 정의 섹션, 어드레싱 모드 정의 섹션, 인스트럭션 정의 섹션으로 나뉜다. SMDL은 어드레싱 모드의 행위 정보 기술을 지원하여 DSP 아키텍처의 auto-increment, bit-reverse등의 특수한 어드레싱 모드 기술이 가능하다. 또한 타겟 코어의 인스트럭션의 행위 정보는 파이프라인 스테이지 별로 기술을 하지 않고, C와 유사한 syntax의 behavior 기술을 지원하므로 기술효율성이 높다. 그림 2에 DSP의 bit-reverse 어드레싱 모드와 MAC 인스트럭션의 SMDL 기술 예를 보인다.

```
AM_Indirect_BR_Increment /* Bit-reverse addressing mode*/
{
  BEHAVIOR
  {
    TargetAddress_Indirect = AR.load [ARP];
    ARAU_Result = ARAU.add_ROP (TargetAddress_Indirect, AR0);
    AR.store [ARAU_Result, ARP];
    return (TargetAddress_Indirect);
  }
  binformat = #AM_Indirect #AM_BR_Increment #DC;
  mnemonic = *BR+;
}
```

(a)

```
MAC /* Multiply-Accumulate instruction */
{
  BEHAVIOR
  {
    ACC = ALU.add (ACC, P_Register);
    T_Register = AM_All;
    P_Register = Multiplier.mult(T_Register, PMA);
  }
  binformat = #MAC AM_All PMA;
  mnemonic = MAC PMA, AM_All;
}
```

(b)

그림 2. DSP 아키텍처의 SMDL 기술 예. (a) Bit-reverse 어드레싱 모드, (b) MAC 인스트럭션.

3.2 SMDL 시스템

SMDL 시스템은 SMDL을 이용한 ASIP 설계 자동화 시스템으로서 타겟 코어의 합성 가능한 HDL 코드를 생성하는 ECGen (Embedded Core Generator), 생성된 코어에서 수행될 최적화된 어셈블리 코드를 생성해 주는 SCC (Sogang C Compiler), 그리고 인스트럭션 셋 시뮬레이터를 자동 생성해주는 ISSGen (Instruction-Set Simulator Generator)의 하위 기능 블록들로 구성된다. 그림 3에 SMDL 시스템의 개관을 보인다.

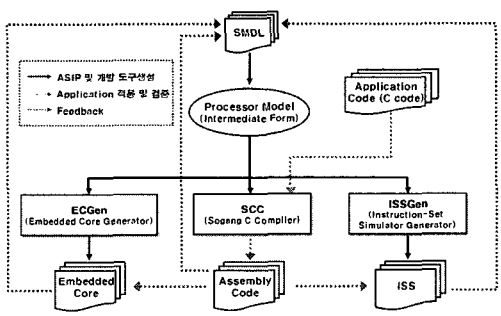


그림 3. SMDL 시스템의 개관.

타겟 코어의 SMDL 기술은 파싱되어 자동생성에 적합한 IF (Intermediate Form)로 변환된다. 생

성된 IF는 코어의 내/외부 구조 정보 및 파이프라인 스테이지 별 행위 정보를 포함하며, 하위 기능 블록인 ECGen, SCC, ISSGen의 입력으로 이용된다. SCC를 통해 생성되는 어플리케이션의 어셈블리 코드는 ISSGen을 통해 생성되는 인스트럭션 셋 시뮬레이터를 통해 검증되며, ECGen을 통해 생성되는 코어 HDL 모델의 시뮬레이션에 이용된다. 설계자는 생성기들을 통해 생성된 HDL 모델과 시뮬레이터를 이용하여 개발할 ASIP 및 어플리케이션에 대한 검증을 수행한다.

Ⅳ. 인스트럭션 셋 시뮬레이터 생성

본장에는 타겟 코어의 SMDL 입력 정보로부터 인스트럭션 셋 시뮬레이터를 자동 생성하는 기법을 제안한다. 그림 4에 인스트럭션 셋 시뮬레이터의 자동 생성 흐름도를 보인다.

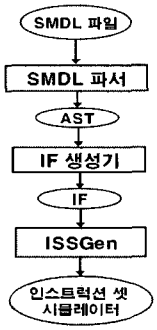


그림 4. 인스트럭션 셋 시뮬레이터의 자동 생성 흐름도

사용자에 의해 기술된 SMDL 정보는 파서를 통해 AST (Abstract Syntax Tree) 형태의 구조체로 변환된다. AST는 SMDL로 기술된 정보를 항목별로 저장하고 있으나 코어, 컴파일러 백엔드 및 시뮬레이터를 자동 생성하기 위해서는 충분한 정보나 구조를 가지고 있지 않아 자동 생성에 적합한 형태인 IF로 생성하는 과정이 요구된다. IF 생성기는 AST를 입력으로 받아 타겟 코어의 인스트럭션 셋 정보를 파이프라인 스테이지 별 행위 정보로 변환한다. ISSGen 시스템은 파이프라인 스테이지 별 행위 정보를 가지고 있는 IF를 입력으로 하여 cycle-accurate 시뮬레이션을 지원하는 인스트럭션 셋 시뮬레이터를 자동 생성한다.

4.1 IF 생성기

IF는 파싱된 타겟 코어의 AST를 입력으로 하여 자동생성에 적합한 중간 형태를 생성한다. 그림 5에 IF 생성기의 개관을 보인다.

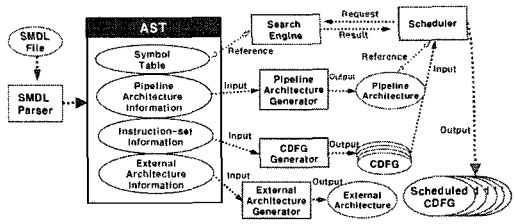


그림 5. IF 생성기의 개관.

IF 생성기는 'Search Engine', 'Pipeline Architecture Generator', 'CDFG (Control-Data Flow Graph) Generator', 'External Architecture Generator', 'Scheduler'의 5개 하위 기능 블록들로 이루어진다. SMDL로 기술된 인스트럭션 셋에 대한 기술은 'CDFG Generator'를 통해 컨트롤/데이터 흐름 정보를 가지는 구조체인 CDFG로 변환되며, 'Scheduler'를 통해 파이프라인 스테이지 별로 스케줄링된다. 스케줄링은 'Pipeline Architecture Generator'를 통해 생성된 코어의 파이프라인 구조 모델의 정보를 참조하여 수행되며, AST의 모든 symbol에 대한 검색은 'Search Engine'을 통해 이루어진다. 최종적으로 생성된 스케줄된 CDFG들은 시뮬레이터 생성기인 ISSGen 시스템에 사용된다.

4.2 ISSGen 시스템

ISSGen 시스템은 IF 생성기를 통해 생성된 IF를 입력으로 받아 타겟 코어의 인스트럭션 셋 시뮬레이터를 C++형태로 자동 생성한다. 그림 6에 시뮬레이터 자동 생성기의 개관을 보인다.

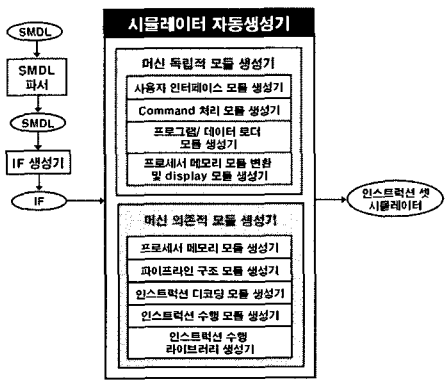


그림 6. 시뮬레이터 자동생성기의 개관.

시뮬레이터 자동생성기는 크게 머신 독립적인 모듈 생성기들과 머신 의존적 모듈 생성기들로 나뉜다. 머신 독립적인 모듈 생성기들은 타겟 코어의 SMDL 기술정보와 관계없이 존재하는 시뮬레이터들의 기본적 구성 모듈들을 생성한다. 머신 독립적인 모듈 생성기들에는 사용자가 shell 환경에서 command를 통해 시뮬레이션을 수행할 수 있도록 해주는 '사용자 인터페이스 모듈 생성기', 사용자의 command들을 처리해주는 모듈을 생성해주는 'Command 처리 모듈 생성기', 시뮬레이션할 코드들의 로딩을 담당하는 '프로그램/데이터 로더 모듈 생성기', 그리고 타겟 코어 내부의 모든 리소스의 값을 변환하거나 display 해주는 '프로세서 메모리 모듈 변환 및 display 모듈 생성기'로 구성된다. 머신 의존적인 모듈 생성기들은 타겟 코어의 SMDL 기술로부터 정보를 추출하여 머신 의존적인 모듈들을 생성한다. 머신 의존적 모듈 생성기에는 타겟 코어 내/외부 기술 정보를 입력으로 하여 코어 내부의 모든 리소스의 모델을 생성하는 '프로세서 메모리 모듈 생성기'와 타겟 코어의 파이프라인 구조 기술 정보를 입력으로 하여 코어의 시뮬레이션 모델을 생성하는 '파이프라인 구조 모듈 생성기', 타겟 코어의 인스트럭 셋 기술정보를 입력으로 하여 디코딩 모듈을 생성하는 '인스트럭션 디코딩 모듈 생성기', 인스트럭션의 수행을 담당하는 '인스트럭션 수행 모듈 생성기', 그리고 타겟 코어의 인스트럭션별 수행 코드를 생성하는 '인스트럭션 수행 라이브러리 생성기'로 구성된다.

4.2.1 프로세서 메모리 모듈 생성기

프로세서 메모리 모듈 생성기는 SMDL의 내/외부 구조 기술을 입력으로 하여 타겟 코어 내부의 모든 저장 유닛들을 포함하는 프로세서 메모리 모듈을 생성한다. 생성된 프로세서 메모리 모듈의 내부 저장 유닛들은 API (Application Program Interface)를 통해 access된다. API의 종류는 타겟 리소스의 입력 정보가 이름인 경우와 이름 및 address인 경우로 나뉜다. 타겟 리소스의 이름으로 데이터를 access할 수 있는 리소스는 (이름, 타입, bit-width) 정의된 레지스터, 프로그램 카운터, 플래그에 해당되며, 타겟 리소스의 이름과 address를 통해 데이터를 access할 수 있는 리소스는 (이름, 타입, bit-width, size) 정의된 레지스터 파일, 메모리, 프로그램 메모리, 스택 등이 있다. 그림 7에 프로세서 메모리 모듈 생성기를 통해 생성된 프로세서 메모리 모듈과 API의 구조를 보인다.

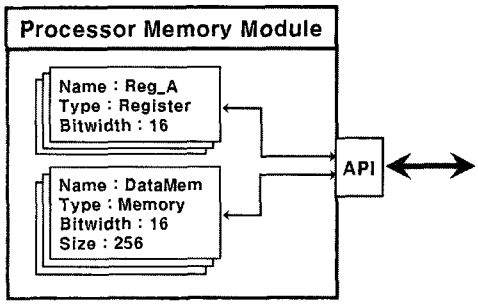


그림 7. 생성된 프로세서 메모리 모듈과 API의 구조

4.2.2 인스트럭션 디코딩 모듈 생성기

인스트럭션 디코딩 모듈 생성기를 통해 생성된 인스트럭션 디코딩 모듈은 fetch된 이진형태의 인스트럭션 정보를 입력으로 받아 시뮬레이션 오브젝트를 생성하는 역할을 한다. 시뮬레이션 오브젝트는 각 파이프라인 스테이지에서 시뮬레이션을 수행할 때 필요한 모든 정보를 가지는 구조체로서 수행할 인스트럭션의 이진형태 정보, fetch될 당시의 프로그램 카운터 값, 필드 별 정보 및 어셈블리 코드, 시뮬레이션 시 참조할 인스트럭션 실행 함수 포인터로 이루어져 있다. 그림 8에 인스트럭션 디코딩 모듈이 fetch된 인스트럭션의 이진형태 문자열을 필드별 정보 및 어셈블리 코드로 변환하는 과정을 보인다.

Fetch된 인스트럭션의 이진 정보는 SMDL로 기술된 인스트럭션 셋의 각 binary 포맷과 비교된다. Fetch된 인스트럭션과 매치되는 binary 포맷을 찾게 되면 인스트럭션의 필드별 정보와 어셈블리 코드가 생성된다. 생성된 필드별 정보는 인스트럭션 실행 시 이용되며, 어셈블리 코드는 시뮬레이터 사용자에게 해당 인스트럭션에 대해 display 해줄 때 사용된다. 매치되는 인스트럭션 정보의 필드별 정보 및 어셈블리 코드 분석이 끝나면 해당 인스트럭션의 실

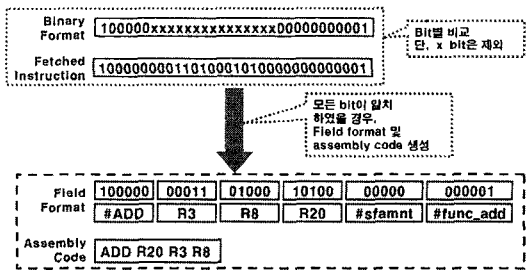


그림 8. 필드별 정보 및 어셈블리 코드 변환 과정.

행 시 참조할 시뮬레이션 함수 포인터 정보를 저장한다.

4.2.3 인스트럭션 수행 라이브러리 생성기

인스트럭션 수행 라이브러리 생성기는 인스트럭션 동작정보의 스케줄된 CDFG 정보를 입력으로 받아 인스트럭션 별 시뮬레이션 코드를 생성한다. 생성된 함수들은 라이브러리화되어 시뮬레이션 시 각 파이프라인 스테이지의 인스트럭션 실행에 이용된다. 그림 9에 인스트럭션 수행 라이브러리 생성과정을 보인다.

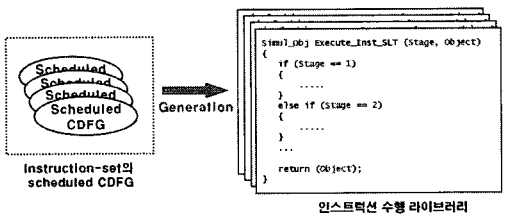


그림 9. 인스트럭션 수행 라이브러리 생성 과정.

4.2.4 파이프라인 구조 모듈 생성기

파이프라인 구조 모듈 생성기는 타겟 코어의 파이프라인 아키텍처의 SMDL 기술을 입력으로 하여, 시뮬레이션 수행 모델인 파이프라인 구조 모듈을 생성한다. 생성된 파이프라인 구조 모듈의 각 스테이지에는 스테이지 이름 정보와 해당 스테이지에서 수행될 시뮬레이션 오브젝트 정보를 가지고 있다. 시뮬레이션 오브젝트는 스테이지에서 수행될 인스트럭션의 이진형태 정보, fetch될 당시의 프로그램 카운터 값, 시뮬레이션 시 참조할 시뮬레이션 코드의 함수 포인터, 해당 인스트럭션이 각 파이프라인 스테이지마다 실행되면서 저장될 로컬 변수 리스트를 가지고 있다. 그림 10에 파이프라인 구조 모듈 생성기를 통해 생성된 파이프라인 구조 모듈을 보인다.

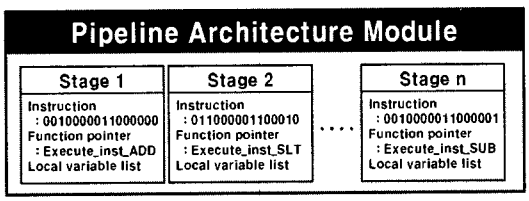


그림 10. 생성된 파이프라인 구조 모듈.

4.3 생성된 시뮬레이터

그림 11에 생성된 시뮬레이터의 개관을 보인다.

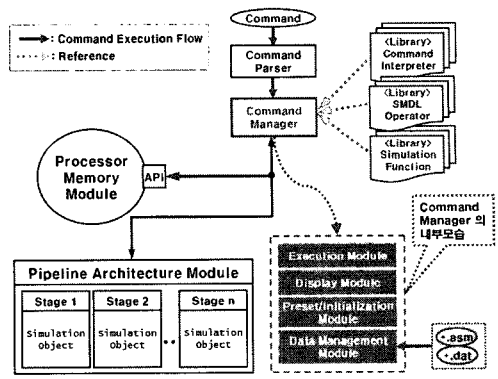


그림 11. 생성된 시뮬레이터의 개관.

생성된 시뮬레이터는 크게 'Command Parser', 'Command Manager', 'Processor Memory Module', 'Pipeline Architecture Module'로 이루어져 있다. 'Command Parser'는 사용자의 입력 command를 파싱하는 역할을 수행하며, 'Command Manager'는 파싱된 command를 실행시키는 역할을 한다. 'Processor Memory Module'은 타겟 코어의 내부 메모리 모듈을 포함하는 구조체로서 메모리 모듈들의 데이터는 API를 통해 access된다. 'Pipeline Architecture Module'은 타겟 코어의 실제적인 시뮬레이션 모델로서 내부의 각 스테이지에는 시뮬레이션을 수행할 모든 정보를 담고 있는 구조체인 시뮬레이션 오브젝트가 존재한다. 'Command Manager'는 입력 받은 command를 분석한 후 'Command Interpreter Library'를 참조하며 각 command에 상응하는 동작을 수행한다. 'Command Manager'의 내부에는 'Execution Module', 'Display Module', 'Preset/Initialization Module', 'Data Management Module'이라는 하위 모듈들로 구성된다. 'Execution Module'은 각 파이프라인 스테이지의 시뮬레이션 오브젝트를 입력으로 하여 인스트럭션을 실행하는 모듈로서, 인스트럭션의 실행은 'SMDL Operator Library'와 'Simulation Function Library'를 참조한다. 'SMDL Operator Library'는 SMDL 언어에서 제공하는 31개의 오퍼레이션에 대한 함수들의 집합이며, 'Simulation Function Library'는 코어의 모든 인스트럭션에 대한 스테이지 별 행위 정보를 가지고 있는 함수들의 집합이다. 'SMDL Operator Library'에는 'add_RCP' (add with reverse carry propagation)와 같은 오퍼레이션이 포함되어 DSP의 특수한 어드레싱 모드에 대한 시뮬레이션이 가능하다. 'Display Module'은 'Processor Memory Module'의 저장 유닛들의 데이

터를 보여주며, 'Preset/Initialization Module'은 'Processor Memory Module'내의 메모리 모듈들의 데이터를 초기화 하거나 변경하는 역할을 한다. 'Processor Memory Module'의 메모리 모듈들에 대한 access는 API를 통해 이루어진다. 'Data Management Module'은 시뮬레이션이 수행될 코드 및 데이터 코드를 로딩하여 'Processor Memory Module'내부의 프로그램 메모리와 데이터 메모리에 저장하는 역할을 한다. 시뮬레이터는 사용자의 command를 입력받아 'Command Parser'를 통해 command argument들로 파싱한다. 'Command Manager'는 파싱된 command를 분석하여 상응하는 내부 모듈들을 수행시키며, 이때 'Processor Memory Module'과 'Pipeline Architecture Module'이 이용된다.

V. 실험 결과

5.1 생성된 시뮬레이터의 동작 검증

생성된 시뮬레이터의 동작을 검증하기 위해 SMDL로 ARM9E-S와 ADSP-TS20x, TMS320C2x 아키텍처들을 기술하여 시뮬레이터를 생성하였으며 4X4 매트릭스 곱셈, 16비트 IIR 필터, 32비트 곱셈, 그리고 8-point DIT FFT 곱셈에 대한 시뮬레이션을 수행하였다. 시뮬레이션 결과를 표 1에 보인다.

4x4 매트릭스 곱셈 연산의 시뮬레이션에는 ARM사의 ARM9E-S 아키텍처를 SMDL로 기술하여 생성된 시뮬레이터를 이용하였다. 192 싸이클 동안 시뮬레이션을 수행하여 결과의 정확성을 확인하였다. 16비트 FIR 필터에 대한 시뮬레이션은 Analog device사의 ADSP-TS20x 코어를 이용하여 시뮬레이션을 수행하였다. 총 231 싸이클 동안 시뮬레이션을 수행하였으며 결과의 정확성을 확인하였다. 32비트 곱셈 연산과 8-point DIT FFT 연산에는 TI사의 TMS320C2x 아키텍처를 이용하였다. 각각 46, 178 싸이클 동안 시뮬레이션을 수행하여 옳은 결과가 나왔음을 확인하였다. 표1의 결과와 같이 SMDL로 기술하여 생성된 타겟 프로세서의 인스트럭션 셋 시뮬레이터들은 타겟 프로세서의 리타겟터블 컴파일러, 코어 생성기와 연동되어 ASIP설계에 있어 자동화된 캐드 프레임워크를 구축하여 임베디드 시스템 개발 시간을 단축시킬 수 있다.

표 1. 어플리케이션 프로그램 시뮬레이션 결과.

Application / Modules	Target Core	Number of Instructions	Number of Execution Cycles
4x4 Matrix Multiplication	ARM9E-S	6	192
16-bit IIR Filter	ADSP-TS20x	23	231
32-bit Multiplication	TMS320C2x	17	46
8-point DIT FFT	TMS320C2x	28	178

5.2 타 MDL과의 기술 양 비교

SMDL은 어드레싱 모드 및 인스트럭션 셋 기술에 있어서 상위 수준의 기술을 지원하며, 스테이지별 기술을 요구하지 않는다는 점에서 타 MDL에 비하여 기술 효율성이 높은 장점을 가진다. 타 MDL과의 기술 양 비교를 위해 표1에서 SMDL로 기술되었던 코어들에 대하여 LISA[22]로 기술한 후 비교하였다. 비교 결과를 표 2에 보인다.

표 2. SMDL과 LISA의 기술 양 비교

Target		LISA로 기술시(A)	SMDL로 기술시(B)	B/A (%)
Core	Number of Instructions			
ARM9E-S	6	708	634	89.5
ADSP-TS20x	23	1202	1026	85.3
TMS320C2x	17	1169	981	83.9
TMS320C2x	28	1404	1162	82.7

범용 임베디드 프로세서인 ARM9E-S코어의 인스트럭션 6개를 기술한 경우 SMDL로는 634라인, LISA로는 708라인이 기술되었으며, SMDL기술이 LISA기술 대비 89.5%만큼 기술 양이 감소하였음을 확인하였다. DSP 프로세서인 ADSP-TS20x의 경우 23개의 인스트럭션에 대해서 기술한 결과 SMDL의 경우 1026라인, LISA로는 1202라인이 기술되었으며 SMDL기술이 LISA기술 대비 85.3% 만큼 감소되었다. 또한 DSP 프로세서인 TMS320C2x의 경우 17, 28개 인스트럭션에 대해서 SMDL이 LISA 기술 대비 각각 83.9, 82.7% 만큼 기술 양이 감소하였음을 확인하였다. ADSP-TS20x, TMS320C2x와 같은 DSP 계열의 코어의 경우 SMDL의 어드레싱 모드에 대한 행위 기술부로 인하여 기술 효용성이

더 증가하였음을 알 수 있다.

Ⅵ. 결론 및 추후과제

본 논문에는 SMDL을 이용한 ASIP 설계 자동화 시스템 중 인스트럭션 셋 시뮬레이터 자동생성기의 설계에 대해 제시하였다. 제안된 시스템에 쓰이는 SMDL 언어는 DSP의 특수한 어드레싱 모드, MAC 오퍼레이션을 포함한 임베디드 코어에 대한 효율적 기술이 가능하다. 타겟 코어에 대해 기술된 SMDL 파일은 SMDL 파서를 통해 의미 있는 구조체인 AST로 변환되며, IF 생성기는 AST에서 추가적인 정보를 추출하여 자동 생성에 적합한 형태인 IF를 생성한다. 시뮬레이터 생성기 ISSGen은 생성된 IF를 입력으로 하여 타겟 코어의 cycle-accurate 시뮬레이션을 지원하는 인스트럭션 셋 시뮬레이터를 자동 생성한다. 제안한 시스템의 검증을 위해 4x4 매트릭스 곱셈 연산, 16비트 IIR 필터, 32비트 곱셈 연산, 8-point DIT FFT에 대한 시뮬레이션을 수행하였으며 결과의 정확성을 확인하였다.

추후 연구 과제로는 생성된 시뮬레이터가 MPSoC 설계 시 다른 프로세서, 하드웨어 블록등과 co-simulation이 가능할 수 있도록 외부 포트 기능을 지원해야 한다. 또한 SMDL이 VLIW 아키텍처 등의 다양한 아키텍처에 대한 기술이 가능하도록 언어를 확장해야 하며, 시뮬레이터가 기능 검증뿐만 아니라 추가적으로 파워 소모, latency등의 실시간 정보를 제공할 수 있도록 기능을 추가하여야 한다.

감사의 글

본 연구는 정보통신부 및 정보통신연구진흥원의 대학 IT 연구센터 육성지원사업의 연구결과로 수행되었으며 Simulation 및 synthesis에 IDEC에서 지원받은 CAD tool을 사용하였음.

참 고 문 헌

[1] D. Edenfeld, A. Kahng, M. Rodgers, and Y. Zorian, "2003 Technology Roadmap for Semiconductors", Trans. on Computer, Vol. 37, No. 1, pp. 47-56, Jan. 2004.

[2] G. Martin, "Overview of the MPSoC Design Challenge", in Proc. Design Automation Conference, ACM/IEEE, pp. 274-279, July 2006.

[3] N. Dutt and K. Choi, "Configurable Processor for Embedded Computing", IEEE Computer, Vol. 36, No. 1, pp. 120-123, Jan. 2003.

[4] 최기영, 조영철, "SoC 설계방법의 최근 동향", 전자공학회지, 30권 9호, pp. 17-27, 2003년 9월.

[5] O. Schliebusch et al, "A Novel Methodology for the Design of Application-Specific Instruction Set Processors (ASIPs) Using a Machine Description Language", IEEE Transactions CAD of Int. Circuits and Systems, Vol. 20, No. 11, pp. 1338-1354, Nov. 2001.

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

[7] P. Marwedel and G. Goossens, Code Generation for Embedded Processors, Kluwer Academic Pub., pp. 138-152, 1995.

[8] 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.

[9] 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.

[10] M. Itoh et al. "Synthesizable HDL Generation for Pipelined Processors from a Micro-Operation Description", IEICE Trans., Vol. E83-A, No. 3, pp. 394-400, Mar. 2000.

[11] Alomary, T. Nakata, Y. Honoma, and N. Hikichi, "An ASIP Instruction Set Optimization Algorithm with Functional Module Sharing Constraint", in Proc. IEEE/ACM Int. Conf. Computer Aided Design, Santa Clara, CA, pp. 526-532, Nov. 1993.

[12] ARM, ARM9E-S Technical Reference Manual, Dec. 1999.

[13] Analog Devices, A 16-bit IIR Filter on the ADSP-TS20x TigerSHARC@ Processor, Nov. 2003.

[14] Texas Instrucments, TMS320C2x User's Guide,

Oct. 1992.

[15] A. Fauth, J. Van Praet, and M. Freericks, "Describing Instruction Set Processors Using nML", in Proc. Eur. Design and Test Conf, Paris, France, pp. 503-507, Mar. 1995.

[16] M. Freericks, "The nML Machine Description Formalism DRAFT Version 1.5", Technical Report, TU Berlin, 1991-1993.

[17] V. Zivojnovic, S. Pees, and H. Meyr, "LISA-Machine Description Language and Generic Machine Model for HS/SW Co-design", in Proc. IEEE Workshop VLSI Signal Processing, San Francisco, CA, pp. 127-136, Oct. 1996.

[18] S. Pees, A. Hoffmann, V. Zivojnovic, and H. Meyr, "LISA-Machine Description Language for Cycle-accurate Models of Programmable DSP Architectures", in Proc. Design Automation Conf., New Orleans, LA, pp. 933-938, June 1999.

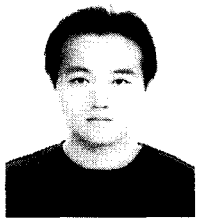
[19] S. Pees, V. Zivojnovic, A. Hoffmann, and H. Meyr, "Retargetable Timed Instruction Set Simulation of Pipelined Processor Architectures", in Proc. International Conference on Signal Processing Applications and Technology, Toronto, pp. 595-599, Sep. 1998.

[20] A. Halambi, P. Grun, V. Ganesh, A. Khare, N. Dutt, and A. Nicolau, "EXPRESSION : a Language for Architecture Exploration through Compiler/Simulator Retargetability", in Proc. DATE, pp. 485-490 Mar. 1999.

[21] P. Grun, A. Halambi, A. Khare, V. Ganesh, N. Dutt, and A. Nicolau, "EXPRESSION : An ADL for System Level Design Exploration", Technical Report TR 98-29, University Of California, Irvine, 1998.

[22] A. Hoffmann, H. Meyr, and R. Leupers, Architecture Exploration for Embedded Processors with LISA, Kluwer Academic Publishers, 2002.

홍 성 민 (Sung-Min Hong) 준회원



2005년 2월 서강대학교 전자공학과 졸업

2007년 2월 서강대학교 전자공학과 석사학위 취득

<관심분야> CAD 시스템

박 창 수 (Chang-Soo Park) 준회원

2002년 2월 서강대학교 전자공학과 졸업

2004년 8월 서강대학교 전자공학과 석사학위 취득

2005년 3월~현재 서강대학교 전자공학과 대학원 CAD & Embedded Systems 연구실 박사과정

<관심분야> SoC 설계, Embedded System Design

황 선 영 (Sun-Young Hwang) 정회원

1976년 2월 서울 대학교 전자공학과 졸업

1978년 2월 한국 과학원 전기 및 전자공학과 공학석사 취득

1986년 10월 미국 Stanford 대학 전자공학 박사학위 취득

1976년~1981년 삼성 반도체 주식회사 연구원, 팀장

1986년~1989년 Stanford 대학 Center for Integrated System 연구소 책임 연구원

Fairchild Semiconductor Palo Alto Reaserch Center 기술 자문

1989년~1992년 삼성전자(주) 반도체 기술 자문

1989년 3월~현재 서강대학교 전자공학과 교수

<관심분야> SoC 설계 및 framework 구성, CAD 시스템, Computer Architecture 및 Embedded System Design 등