임베디드 프로세서 코어 자동생성 시스템의 구축

준회원 조 재 범*, 정회원 유 용 호*, 황 선 영*

Construction of an Automatic Generation System of Embedded Processor Cores

Jae-Bum Cho*, Yong-Ho You*, Sun-Young Hwang* ***Regualar Members***

요 약

본 논문은 SMDL을 이용하여 임베디드 프로세서 코어를 자동 생성해 주는 임베디드 코어 자동 생성 시스템의 구조와 동작에 대해 설명하고 있다. 이러한 SMDL 기술을 통해 제안된 시스템에서는 파이프라인 구조의 데이터패스와 컨트롤 유닛으로 구성된 메모리 모듈을 가진 프로세서 코어를 생성하게 된다. 생성된 코어는 메모리 억세스를 정상적으로 수행할 수 있도록 멀티 싸이클 인스트럭션을 지원하고, 파이프라인 프로세서 상에서 생길 수 있는 파이프라인 해저드를 처리할 수 있다. 실험 결과를 통해서 생성된 코어의 정확성을 확인할 수 있다.

Key Words : MDL, Automatic Generation, Embedded Core, ASIP

ABSTRACT

This paper presents the structure and function of the system which automatically generates embedded processor cores using the SMDL. Accepting processor description in the SDML, the proposed system generates the processor core, consisting of the pipelined datapath and memory modules together with their control unit. The generated cores support muti-cycle instructions for proper handling of memory accesses, and resolve pipeline hazards encountered in the pipelined processors. Experimental results show the functional accuracy of the generated cores.

Ⅰ. 서 론

최근 SoC 설계는 반도체 공정의 발전으로 하나 혹은 이상의 범용 임베디드 프로세서를 포함하여 설계의 유연성과 재사용율을 높이고 있다. 그러나 임베디드 시스템에 사용되는 SoC는 제한된 소비전력과 비용으로 최대의 성능이 요구되어 기존의 고정된 범용 프로세서로는 그 욕구를 만족시키기가 점점 어려워지고 있다. 이러한 이유로 프로세서의 유연성과 소비전력, 성능 사이의 trade-off를 고려하여 어플리케이션의 최적화를 위해, 고정된 범용 임베디드 프로세서 대신 ASIP (Application Specific Instruction set Processors)이 SoC 에 사용되고 있다[1, 2]. ASIP에는 프로세서의 기본 코어와 IS(Instruction Set)를 제공하고 설계자가 파라미터를 조절하여 원하는 ASIP을 얻는 파라미터 기반의 ASIP 설계 방식[3]과, Machine Description[4]을 이용하여 설계자가 IS 전체를 설계하는 MDL(Machine Description Language) 기반의 설계 방식[4-8]등이 제안되고 있다. 파라미터 기반 설계 방식은 ASIP 설계 자동화 시스템 내에 기본적인 프로세서 모델(Architecture Skeleton)을 가지고 설계자가 프로세서 모델의 파라미터만을 변경하여 타겟 ASIP을 얻기 때문에 해당 ASIP 설계 자동화 시스템이 제공하는 프

* 서강대학교 전자공학과 CAD & Embedded System 연구실 (hwang@ccs.sogang.ac.kr)
논문번호 : KICS2004-08-144, 접수일자 : 2004년 8월 10일
※본 연구는 대학 IT 연구센터 (연세대 ITRC) 육성 지원 사업의 연구결과로 수행되었습니다.

로세서의 범주를 벗어나는 다양한 머신의 구조를 구현할 수가 없다는 단점을 가지고 있지만, 타겟 프로세서와 소프트웨어 툴 셋을 쉽게 구현할 수 있다는 장점을 가지고 있다. MDL 기반의 설계 방식은 설계자가 프로세서의 모든 IS에 대해서 직접 기술하는 방식으로 다양한 머신 구조를 구현할 수 있는 장점을 가지고 있지만, 기존의 연구들은 다음과 같은 단점을 보이고 있다. 베를린 대학에서 개발한 nML[5]은 타겟 프로세서에 대한 기술을 통하여 간단한 컨트롤러를 생성할 수 있지만, 파이프라인에 대한 기술을 지원하지 않는다. 따라서 파이프라인된 프로세서에 대한 생성에 어려움이 있다. 독일의 Aachen 대학에서 개발한 LISA[6]는 nML의 단점을 개선하여 파이프라인 기술이 가능하도록 하였다. 그러나, 프로세서 코어 생성 측면에서 데이터패스와 디코딩 정보만을 추출하여 고정된 파이프라인 컨트롤러에 맵핑하는 방식을 취하기 때문에 다양한 머신의 구조를 생성할 수 없다. 또한 설계자가 해저드 검출 및 처리에 대한 정보를 자세히 기술하도록 하여 프로세서 개발 초기부터 타겟 프로세서에 대해 구체적이고 정확한 기술이 요구된다. 미국의 UCI에서 개발한 EXPRESSION[7] 시스템은 MDL을 근간으로 프로세서 코어를 생성하는 기존의 ASIP 설계 자동화 시스템과는 용도와 구조가 상이한 시스템이다. 기존에 존재하는 범용 프로세서 IP 정보를 이용하여 프로세서 라이브러리를 구축하고 이 라이브러리에 있는 프로세서를 EXPRESSION으로 기술하면 EXPRESSION 시스템은 소프트웨어 툴 셋을 자동 생성하는 방식을 취하고 있다. 유저는 구축된 라이브러리 정보와 생성된 툴 셋을 이용하여 타겟 SoC 시스템을 검증하고 시뮬레이션할 수 있다. 따라서 이 시스템은 다양한 프로세서에 대한 생성에 적합하지 않다. 일본 오사카 대학에서는 Micro Operation Description[8]을 통해 프로세서 코어를 자동 생성하는 연구를 수행하였다. 그러나 Micro Operation Description의 특성상 추상화 수준이 낮은 RTL 수준의 기술이 필요하기 때문에 유저는 프로세서 내부의 서브 컴퍼넌트들 간의 연결 정보를 세부적으로 기술해야 한다. 또한 그들이 제안한 시스템은 블록 데이터 전송 인스트럭션 등을 위한 멀티 사이클 인스트럭션과 파이프라인 해저드 처리를 고려하지 않았다.

이와 같이 최근 임베디드 시스템에 사용되는 SoC의 설계 동향을 분석해 보면 다양한 타겟 어플리케이션에 최적화된 SoC 설계를 위해서 각각의 어플리케이션에 적합한 프로세서와 그 프로세서의 소프트웨어 툴 셋을 자동 생성할 수 있는 MDL기반의 'ASIP 설계 자동화 시스템'이 필요하다는 것을 알 수 있다. 그러나 ASIP 설계 자동화 시스템을 실용화하기 위해서는 기존의 MDL기반의 ASIP 설계 자동화 시스템이 보이고 있는 단점들의 극복이 반드시 선행 되어야 한다. 이를 위해 본 논문에서는 기존의 MDL기반의 ASIP 자동생성 연구들의 단점들을 개선한 ASIP 자동 생성 시스템인 '임베디드 프로세서 코어 생성기(Embedded Core Generator)'를 제안한다. 제안된 임베디드 프로세서 코어 생성기는 대상 ASIP을 기술한 SMDL 기술을 입력으로 받아 타겟 ASIP을 자동생성해 주는 시스템으로서, 생성된 타겟 ASIP은 메모리 엑세스 등의 효율적인 처리를 위해 다중 사이클 인스트럭션을 지원하며 별도의 파이프라인 해저드 정보의 기술 없이 해저드 감지 및 처리가 가능하다. 또한 제안된 시스템은 본 연구실이 구축한 ASIP 설계 자동화 시스템의 한 구성요소로서 다른 구성요소인 SRCC (Sogang Retargetable C Compiler), ISSGen(Instruction Set Simulator Generator)과 연동하여 임베디드 SoC에 사용되는 ASIP의 설계, 생성, 검증, 프로파일링 등에 사용될 수 있다.

본 논문의 Ⅱ에서는 본 연구실에서 구축한 SMDL (Sogang Machine Description Language) 언어와 SMDL을 이용한 ASIP 설계 자동화 시스템인 SMDL 시스템을 보이고, Ⅲ에서는 본 논문이 제안하는 임베디드 코어 생성기의 구조와 임베디드 프로세서 코어의 자동생성 과정을 설명하였다. Ⅳ에서는 제안된 시스템을 평가하기 위해 상용화된 임베디드 프로세서들의 SMDL 기술을 제안된 시스템에 입력하여 프로세서 코어를 생성한 결과를 보였다.

Ⅱ. SMDL 시스템과 SMDL 언어

SMDL 시스템은 SoC 설계 자동화를 위해 구축한 시스템으로서, 타겟 프로세서를 자동생성하는 임베디드 코어 생성기와 임베디드 코어 생성기에서 생성된 프로세서에서 수행 가능한 최적화된 어셈블리 코드를 생성해주는 리타겟터블 컴파일러인 SRCC, 타겟 프로세어의 ISS(Instruction Set Simulator)를 자동생성하는 RISGen(Retargetable Instruction set Simulator Generator)으로 구성된다. 설계자가 이 시스템에 타겟 프로세서의 SMDL기술과 어플리케이션 프로그램을 입력하면 어플리케이션에 최

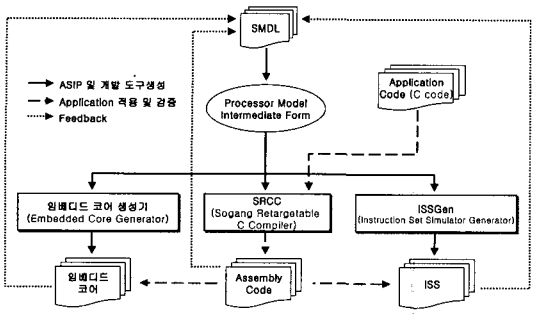

그림 1. SMDL 시스템 개관

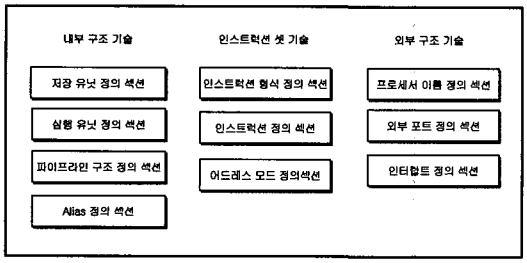

그림 2. SMDL 언어의 구조

적화된 프로세서 코어, 컴파일러, ISS가 생성된다. 그림 1은 구축된 SMDL 시스템의 개관을 보인다.

SMDL 언어는 타겟 아키텍처의 인스트럭션에 대한 정보와 저장 모듈, 실행 유닛 등의 리소스 구조에 대한 정보와 파이프라인의 구조를 기술할 수 있는 C언어의 형태를 띠고 있는 머신 기술 언어이다. SMDL은 크게 내부 구조 기술, 인스트럭션 셋 기술, 외부 구조 기술로 구분된다. 그림 2에 SMDL 언어의 구조를 나타낸다.

SMDL의 정확성과 유용성은 DSP56000[9], ARM7 [10, 11], ARM9[10, 12], DLX[13], MIPS R3000 [14]등의 프로세서들을 제안된 시스템에서 생성하고 그 동작을 시뮬레이션을 통해 검증하였다.

Ⅲ. 제안된 임베디드 코어 생성기

제안된 임베디드 코어 생성기는 타겟 프로세서의 SMDL 기술을 입력받아 프로세서 생성에 적합한 중간형태를 생성하는 전 처리 과정, 생성된 중간형으로부터 타겟 프로세서 모델을 생성하는 코어 모듈 생성 과정, 생성된 모델을 합성 가능한 HDL 코드로 변환하는 HDL 코드 생성 과정으로 이루어진다. 그림 3에 임베디드 코어 생성기의 개관을 보인다.

3.1 전 처리 과정

전 처리 과정은 SMDL로 기술된 타겟 프로세서

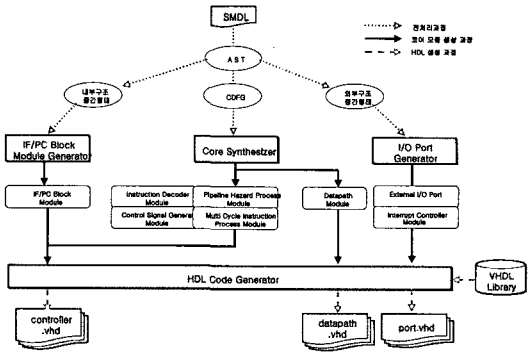

그림 3. 임베디드 코어 생성기 시스템의 개관

기술을 이용하여 프로세서 생성에 적합한 중간형태로 변환하는 과정으로 타겟 프로세서에서 사용되는 연산자와 저장유닛의 기능과 사용되는 스테이지에 대한 정보를 가진 '내부 구조 중간형태', 인스트럭션 셋 기술로부터 추출되며 타겟 프로세서에서 사용되는 리소스(연산자, 저장유닛) 간의 연결정보를 갖는 'CDFG (Control Data Flow Graph)', 외부 구조 기술로부터 추출되며 타겟 프로세서와 외부의 디바이스와 인터페이스를 위한 외부 포트의 정보를 가진 ' 외부 구조 중간형태'의 3가지 중간형태를 만든다.

3.2 코어 모듈 생성 과정

코어 모듈 생성 과정은 전 처리 과정에서 만들어진 중간형태를 이용하여 실질적인 프로세서 모델을 만드는 과정이다. 전처리 과정에서 생성된 CDFG 내부 구조 중간형태 중 파이프라인 스테이지 리소스 정보를 제약조건으로 하여 파이프라인 스케줄링을 한다. 이를 토대로 타겟 프로세서의 데이터패스 모델과 CDFG의 각 노드들이 실행될 스테이지와 동작에 대한 정보를 가진 ACT(Active Component Table)[15]을 구성한다. 이후 생성된 정보(외부 구조 정보, 내부 구조 정보, CDFG, ACT)를 토대로 타겟 프로세서가 생성되는데 필요한 부 모듈들을 생성해 낸다. 코어 모듈 생성과정에 의해 생성되는 타겟 프로세서 모델의 블록 다이어그램을 그림 4에 보인다.

다음은 각각의 부 모듈들의 생성 방법을 설명하였다.

3.2.1 데이터 패스 생성.

전 처리 과정에서 구성된 CDFG 상에서 단일 인스트럭션의 파이프라인 동작 때문에 발생될 수 있는 리소스 충돌(resource conflicts)이나 멀티 사이클 연산자가 있을 경우에는 필요한 만큼의 사이클을

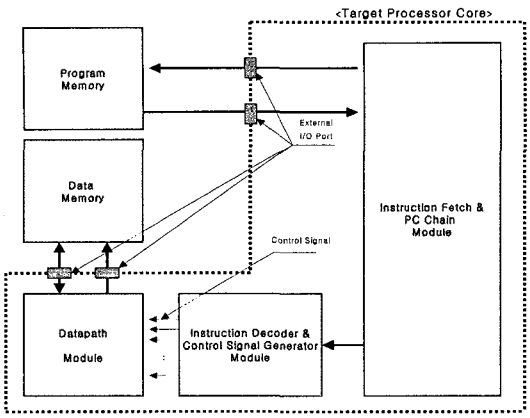

그림 4. 생성된 타겟 프로세서의 블록 다이어그램

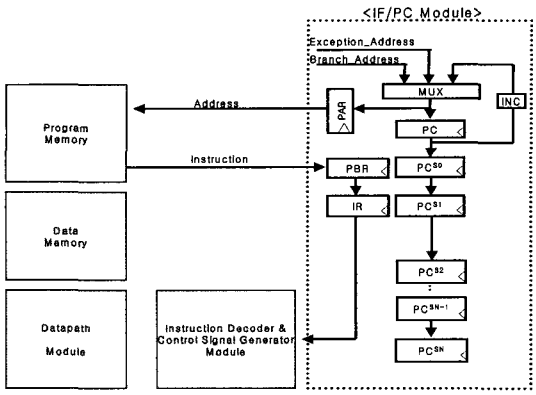

그림 5. IF/PC 모듈의 구조

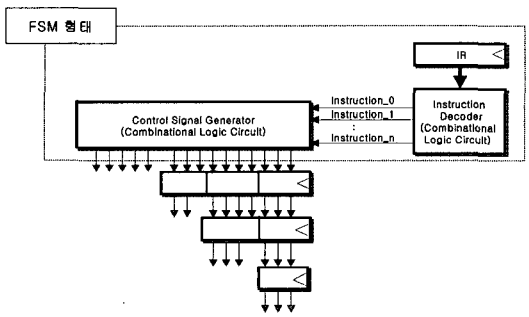

그림 6. 인스트럭션 디코더와 컨트롤 신호 발생기

카운트한 후 이 정보를 토대로 ACT의 컨트롤 스텝을 확장한다. 이후 ACT의 서로 다른 스테이지에 존재하는 연산자 간의 연결에 파이프라인 레지스터를 삽입한다.[8] 구성된 각 인스트럭션의 CDFG들은 하나의 CDFG로 조합되어 타겟 프로세서의 데이터패스 모듈을 생성하게 되는데, 다음과 같은 최적화 과정을 통하여 이루어진다.

a. 리소스 할당 과정 : 모든 CDFG의 노드들을 각 스테이지에 맞게 배열하고 중복되는 레지스터와 연산자를 제거한다.
b. 연결구조 추출 및 최적화 과정 : 인스트럭션 CDFG로부터 노드 간의 연결정보를 추출한다. 이후에 버스충돌을 피하면서 최소 연결을 가진 전체적인 구조를 추출한다.
c. 버스 할당 및 ACT 업데이트 과정 : 연결구조 구성 과정에서 다른 시점-동일 종점 연결구조에 mux를 삽입하고, 이 과정을 통해 추가된 mux를 ACT에 업데이트한다. [16]

3.2.2 IF/PC 모듈 생성

IF/PC 모듈 생성기는 전 처리 과정에서 생성한 구조 정보를 이용하여 PBR(Program Buffer Register), PAR (Program Address Register), PC 체인 구조[17]를 가진 IF/PC 모듈을 생성한다. 그림 5는 N-스테이지 파이프라인 구조를 가진 IF/PC 블록을 보인다. 그림 5에서 IR은 인스트럭션 레지스터를, PCS0~N는 각 스테이지별 PC 체인을 나타내낸다.

3.2.3 인스트럭션 디코더와 컨트롤 신호 발생기 생성

연산자, mux, 저장유닛들의 노드 정보를 참조하여 생성된 데이터패스가 필요로 하는 제어 신호들의 정보를 추출하고, ACT로부터 인스트럭션마다 노드들이 취해야 하는 동작과 타이밍에 대한 정보를 추출한다. 추출된 정보들을 토대로 PLA 형태의 디코딩 신호 테이블(DST : Decoding Signal Table)을 구성한다. 이 테이블의 입력은 인스트럭션의 'binformat' 정보이며 출력은 각 스테이지 별로 발생되어야 할 컨트롤 신호가 된다.

구성된 DST를 이용하면 그림 6과 같은 FSM 형태의 인스트럭션 디코더와 컨트롤 신호 발생기를 생성할 수 있다. 인스트럭션 디코더는 인스트럭션 셋의 종류에 따라 상태가 바뀌고, 바뀐 상태를 컨트롤 신호 발생기에 전달하게 된다. 컨트롤 신호 발생기는 인스트럭션 디코더의 각 상태, 즉 펫치된 인스트럭션이 각 스테이지마다 발생해야할 컨트롤 신호들을 발생하게 된다.

3.2.4 컨트롤러 생성

제안된 시스템이 생성하는 컨트롤러는 코어 모듈 생성 과정을 통해 생성된 타겟 프로세서의 IF/PC 모듈과 인스트럭션 디코더/컨트롤 신호 발생기를 합쳐서 기본 구조를 구성한다. 그림 7은 제안된 시스템이 생성하는 컨트롤러의 기본 구조를 보인다.

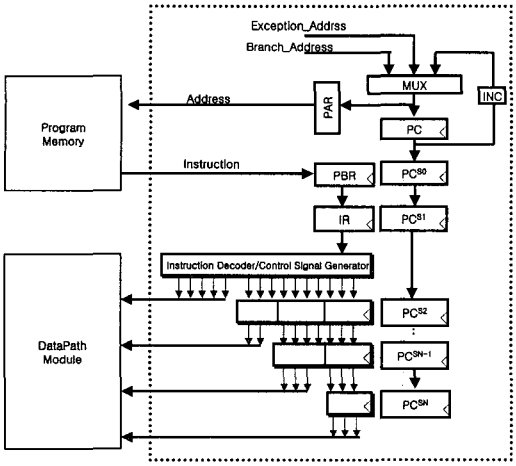

그림 7. 제안된 시스템이 생성한 컨트롤러의 기본 구조

3.2.5 멀티 사이클 인스트럭션 처리 모듈 생성

멀티 사이클 인스트럭션 처리 모듈은 리소스 충돌로 발생되는 사이클 익스텐션이나 멀티 사이클 연산자처럼 단일 사이클 명령어로는 수행할 수 없는 인스트럭션들을 지원하기 위해 제안된 시스템이 생성하는 컨트롤러의 한 부분이다. 멀티 사이클 인스트럭션 처리 모듈의 생성방법은 다음과 같다. 멀티 사이클 인스트럭션의 ACT를 분석하여 처리에 필요한 사이클 정보를 멀티 사이클 룩 업 테이블에 저장한다. 이 테이블은 멀티 사이클 인스트럭션이 입력되면 입력된 인스트럭션 처리에 필요한 사이클 수를 사이클 카운터에 업데이트한다. 멀티 사이클 인스트럭션이 펫치된 다음, 계산된 사이클수로부터 컨트롤 신호 발생기는 IR과 사이클 카운터를 이용하여 컨트롤 신호를 발생하게 된다. 이때 멀티 사이클 명령어 처리 중에는 IF 스테이지를 스톨시켜야 하기 때문에 사이클 카운터는 자신이 0이 아닌 동안 계속해서 IF 스테이지를 스톨하도록 스톨 신호 발생 로직에 요청한다. 그림 8은 멀티 사이클 명령어 처리 모듈의 구조를 보인다

3.2.6 파이프라인 해저드 처리 모듈 생성

파이프라인 해저드 처리 모듈은 타겟 프로세서에서 발생하는 파이프라인 해저드를 검출하고 처리하는 모듈이다. 이 모듈들은 별도의 SMDL 기술 없이 구축된 시스템이 ACT와 CDFG를 분석하여 자동 생성한다. 다음은 파이프라인 해저드 처리 모듈의 형태와 자동 생성 방법에 대해 설명한다.

(1) 데이터 해저드 처리

각 인스트럭션의 ACT의 레지스터 관련 부분 중

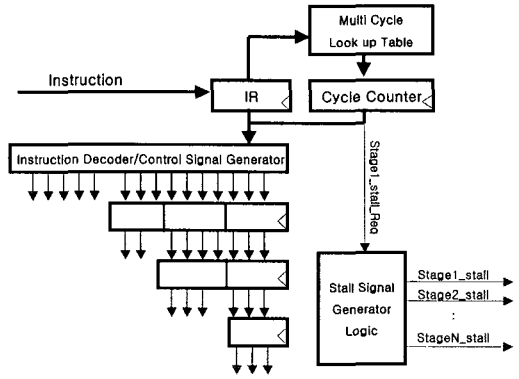

그림 8. 멀티 사이클 명령어 처리 모듈의 구조

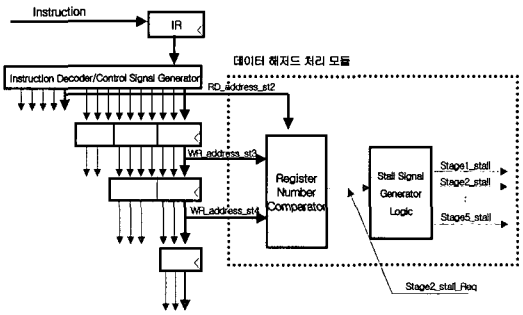

그림 9. 생성된 데이터 해저드 처리 모듈

레지스터 읽기와 쓰기가 활성화되는 부분을 분석하고, 이 정보를 이용하여 읽기 레지스터 어드레스 신호(RD_address)와 쓰기 레지스터 어드레스 신호(WR_address)를 비교하는 비교기를 생성한다. 생성된 비교기는 읽기 레지스터 어드레스와 쓰기 레지스터 어드레스가 같은 경우 레지스터 읽기 스테이지를 스톨하도록 스톨 신호 발생기에 요청한다. 그림 9는 5단 파이프라인 구조의 타겟 프로세서가 스테이지 2에서 읽기를 하고 스테이지 5에서 쓰기를 할 경우 생성되는 데이터 해저드 처리 모듈을 보인다.

(2) 컨트롤 해저드 처리

제안된 시스템에서 컨트롤 해저드 처리는 분기 명령어 입력 시 컨트롤 해저드가 발생하지 않을 때까지 펫치 스테이지(스테이지 1)를 스톨시키는 방식을 취하고 있다. 이때 펫치 스테이지의 스톨 기간(사이클 수)은 분기 인스트럭션의 ACT를 분석하여 타겟 점프 주소가 PC에 업데이트되는 스테이지를 카운트하면 알아낼 수 있다. 컨트롤 해저드 처리에 필요한 스톨 사이클 수는 식 (1)과 같으며, 알아낸 스톨 사이클 수를 사이클 익스텐션 룩 업 테이블에 추가한다.

스톨사이클수 = PC가 업데이트되는 스테이지의 수-1 (1)

분기 인스트럭션이 입력되면 사이클 카운터 스테이지 1에 카운트된 사이클 넘버가 입력이 되고, 분기 인스트럭션이 스테이지 2로 넘어가게 되면 카운트된 사이클 만큼 따라오는 스테이지를 스톨시킨다. 이 때문에 컨트롤 해저드 처리를 위한 추가 모듈은 필요가 없다.

3.2.7 외부 입출력 포트 생성

외부 포트 정의 기술로부터 생성되는 외부 입출력 포트는 외부 구조 정보 중간형태 중 유저 정의 포트의 이름, 속성 그리고 비트 폭을 이용하여 유저가 기술한 타겟 프로세서의 외부 포트를 생성한다. 포트 기술 없이 생성되는 외부 입출력 포트는 타겟 프로세서가 동작하기 위해 반드시 필요한 기본적인 포트나 외부 디바이스와 동기를 맞추는데 필요한 포트들을 유저 기술 없이 자동생성한다. 그림 10에 외부 포트 기술 및 생성된 포트의 예를 보인다.

3.3 HDL 코드 생성 과정

코어 모듈 생성과정에서 생성된 모듈들을 HDL 라이브러리에 있는 객체로 매칭을 수행하고, 필요한 HDL 코드를 추가하여 합성 가능한 코드를 생성한다. 본 논문에서는 대상 언어를 VHDL로 하였다. VHDL 코드 생성 방법은 다음과 같다.

3.3.1 데이터패스 기술

생성된 데이터패스를 VHDL 기술로 변환하여 구성한다. 내부 버스는 signal statement로, 상수는 constant statement로 구성한다. mux 및 연산자는 비트의 수를 파라미터로 하여 VHDL 라이브러리 중에서 연산자와 연결구조 라이브러리를 호출한다. 레지스터는 데이터 비트와 어드레스 비트의 수를 파라미터로하여 메모리 라이브러리를 호출하여 구성한다.

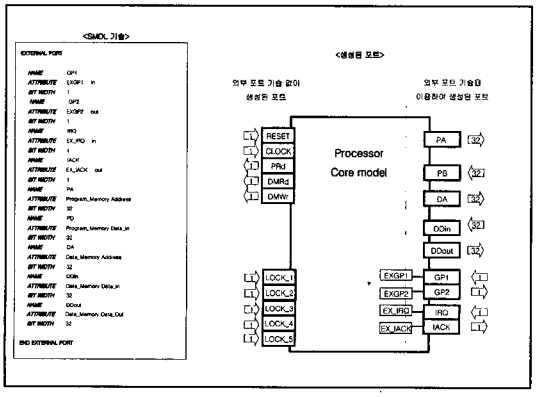

그림 10. 외부 포트 기술 및 생성된 포트의 예

3.3.2 컨트롤러 기술

IF/PC 블록은 내부 구조 정보의 파이프라인 정보와 외부 구조 정보의 메모리 입출력 포트 정보를 파라미터로 IF/PC 라이브러리를 호출한다. 인스트럭션 디코더는 VHDL의 type문을 이용하여 인스트럭션과 어드레스 모드의 이름과 'binformat'을 정의한 코드를 만들고, constant 문과 case문을 이용하여 만든 인스트럭션 디코더의 코드에서 인스트럭션과 어드레스 모드 정의 코드를 호출한다. 컨트롤 신호 발생기는 DST를 이용하여 코드를 만들어 낸다. 인스트럭션 디코더 코드와 컨트롤 신호 생성기는 컨트롤러 코드에서 IR과 함께 호출되며 컨트롤러 코드의 코딩 형태는 FSM 형태를 가진다. 데이터 해저드 처리 모듈은 해저드 처리 라이브러리를 호출하여 사용한다. 멀티 사이클 처리 모듈에서 사이클 카운터는 해저드 처리 라이브러리에서 최대 사이클 카운터 수를 파라미터로 호출한다. 멀티 사이클 룩업 테이블은 type문과 case문을 이용하여 메모리 코드 형태로 구성한다.

3.3.3 외부 포트 기술

생성된 코드의 최고 상위 계층에서 구성되며, 패스 생성된 외부 포트 모델과 매치되는 VHDL 인터페이스 라이브러리의 포트 객체를 호출한다. 생성된 외부 포트 모델의 포트를 이용하여 타겟 프로세서의 최고 상위 계층 코드의 entity 부분을 구성한다.

Ⅳ. 실험 결과

실험 과정은 다양한 타겟 프로세서의 생성을 검증하기 위해 상용화된 임베디드 프로세서 코어인 ARM7[10, 11], ARM9[10, 12], MIPS R3000[14]을 타겟 프로세서로 선정하고, SMDL의 '내부 구조 기술' 부분과 '외부 구조 기술' 부분을 기술하였다. 또한 타겟 프로세서의 파이프라인 동작 검증을 위하여 타겟 프로세서의 인스트럭션 중 인스트럭션 타입 별로 인스트럭션을 하나씩 선정하여 '인스트럭션 셋 기술' 부분을 기술하였다. 기술된 타겟 프로세서 기술들을 구축된 시스템에 입력하여 각각의 타겟 프로세서 코어들을 생성한 후, 각각의 인스트럭션 사이클과 파이프라인 동작을 실제 프로세서와

표 1. ARM7의 인스트럭션 사이클 비교 결과.

인스트럭션	인스트럭션/해저드 처리 사이클 (단위 : 클럭)		비고 (인스트럭션 타입)
	실제 프로세서	생성된 모델	
ADD	3	3	산술연산
LDR	5	3 + 2 익스텐션	로드
STR	4	3 + 1 익스텐션	스토어
B	5	3 + 2 stall	분기

비교하였다. 메모리 액세스 시간은 1 사이클, 캐시는 항상 히트 된다고 가정하였다.

ARM7 모델 생성 실험에서는 실제 프로세서와 마찬가지로 파이프라인은 3단, 클럭은 2-phase non-overlapping, 해저드는 하드웨어로 처리하도록 SMDL을 기술하였다. 표 1에 ARM7과 생성된 모델의 인스트럭션 사이클을 비교한 결과를 보인다.

생성된 모델에서 LDR과 STR 인스트럭션은 버스 및 리소스 충돌로 인하여 각각 2, 1 사이클이 익스텐션 되어 실제 프로세서와 같은 인스트럭션 사이클을 갖는다. 데이터 해저드에 관련하여 ARM7의 레지스터 read와 write 동작은 스테이지 3에서만 이루어진다. 이로 인해 생성된 모델에서 데이터 해저드는 발생하지 않았으며, 데이터 해저드 처리 로직 역시 생성되지 않는다. 생성된 모델에서 컨트롤 해저드를 발생시키는 B 인스트럭션은 3개의 인스트럭션 사이클을 필요로 하고, 추가로 2 사이클의 stall이 일어나게 되어 실제 프로세서와 같은 5 사이클이 요구된다. 결과적으로 생성된 ARM7 모델은 실제 프로세서와 같은 인스트럭션/해저드 처리 사이클이 필요하다.

ARM9 모델을 생성하는 실험에서는 실제 프로세서와 마찬가지로 하바드 아키텍처에 파이프라인은 5단, 클럭은 2-phase non-overlapping, 해저드는 하드웨어로 처리하도록 SMDL을 기술하였다. 표 2에 ARM9과 생성된 모델의 인스트럭션 사이클을 비교한 결과를 보인다.

표 2. ARM9의 인스트럭션 사이클 비교 결과

인스트럭션	인스트럭션/해저드 처리 사이클 (단위 : 클럭)		비고 (인스트럭션 타입)
	실제 프로세서	생성된 모델	
ADD	5	5	산술연산
LDR	5	5	로드
STR	5	5	스토어
B	7	5 + 2 stall	분기

생성된 모델은 5단 파이프라인에 인스트럭션 버스와 데이터 버스가 분리되었으며 리소스 충돌은 발생하지 않았다. 이로 인해 사이클 익스텐션은 일어나지 않으며 스테이지 1을 제외한 나머지 stall 카운터 로직이 생성되지 않았다. 데이터 해저드 처리에 있어서 생성된 모델은 최대 3 사이클의 stall이 발생했지만 실제 프로세서는 내부 포워딩 로직 때문에 최대 1 사이클의 stall만이 발생한다. 분기 인스트럭션의 경우 생성된 모델은 5 사이클의 인스트럭션 사이클에 추가로 2 사이클의 stall이 발생한다. 실제 프로세서는 7개의 인스트럭션 사이클 필요하다. 결과적으로 생성된 모델은 실제 프로세서와 같은 인스트럭션 사이클과 컨트롤 해저드 처리를 위한 추가 사이클이 필요하며, 데이터 해저드의 경우만 실제 프로세서보다 2 사이클이 더 소요된다.

MIPS R3000 모델을 생성하는 실험에서는 실제 프로세서와 마찬가지로 하버드 아키텍처에 파이프라인은 5단으로, 클럭은 2-phase non-overlapping 클럭을 사용하였으며, 분기 주소 계산은 스테이지 2에서 수행하고, 해저드는 소프트웨어로 처리하도록 SMDL을 기술하였다. 표 3에 MIPS R3000과 생성된 모델의 인스트럭션 사이클을 비교한 결과를 보인다.

생성된 모델은 5단 파이프라인에 인스트럭션 버스와 데이터 버스가 분리되었으며 리소스 충돌은 발생하지 않았다. 데이터 해저드 처리에 있어서 생성된 모델은 최대 2개의 NOP이 필요하지만 실제 프로세서는 내부 포워딩 로직 때문에 최대 1개의 NOP이 필요하다. 분기 인스트럭션의 경우 생성된 모델은 실제 프로세서와 같은 1개의 NOP이 필요하다. 이는 SMDL 점프 주소 계산 및 PC 업데이트가 스테이지 2에서 일어나도록 기술하였기 때문이다. 결과적으로 생성된 모델은 실제 프로세서와 같은 인스트럭션 사이클과 컨트롤 해저드 처리를 위한 추가 NOP 인스트럭션이 필요하며, 데이터 해저드의 경우만 실제 프로세서보다 많은 NOP이 필요하다.

표 3. MIPS R3000의 인스트럭션 사이클 비교 결과

인스트럭션	인스트럭션/해저드 처리 사이클 (단위 : 클럭)		비고 (인스트럭션 타입)
	실제 프로세서	생성된 모델	
ADD	5	5	산술연산
LDR	5	5	로드
STR	5	5	스토어
J	5	5	분기

V. 결론 및 추후 과제

본 논문은 SMDL을 이용하여 임베디드 프로세서 코어를 자동생성하는 시스템인 임베디드 코어 생성기를 제안하였다. 제안된 시스템이 생성한 프로세서 코어 내부에는 파이프라인 해저드 처리 로직과 멀티 사이클 인스트럭션 처리 로직이 생성될 수 있으며, 생성된 해저드 처리 로직은 외부 포트와 연결되어 생성된 프로세서 코어와 다양한 메모리 소자들 간의 인터페이스에 활용될 수 있다. 실험은 제안된 시스템의 정확성과 효용성을 검증하기 위해 ARM7, ARM9, MIPS R3000을 SMDL로 기술한 뒤, 제안된 시스템을 사용하여 각각의 타겟 프로세서 모델을 생성한 후 실제 프로세서와 비교하였다.

추후 과제로는 제안된 시스템이 생성하는 프로세서의 성능 및 기능 향상 연구가 필요하다. 생성된 프로세서의 성능을 향상하기 위해서는 인터널 포워딩과 분기 예측이 지원되어야 하며, 기능 향상을 위해서는 발전된 머신 구조인 VLIW 머신 구조 지원에 대한 연구가 필요하다.

참 고 문 헌

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

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

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

[4] P. Marwedel and G. Goossens, "Code Generation for Embedded Processors," *Kluwer Academic Publishers*, pp. 138-152, 1995.

[5] 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.

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

[7] 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.

[8] 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, March 2000.

[9] Motorola, "DSP56000 : 24-BIT Digital Signal Processor Family Manual", 1995.

[10] S. Furber "ARM System-on-chip Architecture", *Addison-Wesley*, 2000.

[11] ARM, "ARM7TDMI Technical Reference Manual (rev 3)", 2001.

[12] ARM, "ARM922T Technical Reference Manual (rev 0)", 2001.

[13] J. Hennessy and D. Patterson, "Computer Architecture : A Quantitative Approach", *Morgan Kaufmann Publishers Inc.*, 1990.

[14] G. Kane, MIPS RISC Architecture, *Prentice -Hall*, 1998.

[15] H. Lee and S. Hwang "Design of a High-Level Synthesis System for Automatic Generation of Pipelined Datapath", *Journal of KITE, Vol. 31-A, No. 4*, pp. 53-67, March 1994.

[16] Y. Kim, H. Lee, and S. Hwang, "An Interconnect Allocation Algorithm for Performance-driven Datapath Synthesis", *Journal of Circuits, Systems, and Computers, Vol. 7, No. 4.* pp. 403-423 Sept. 1996.

[17] P. Kogge "The Architecture of Pipelined Computers", *Hemisphere Pub. Co.*, 1981.

조 재 범 (Jae-Bum Cho) 준회원

2000년 2월 단국대학교 전자공학과 졸업
2005년 2월 서강대학교 전자공학과(공학석사)
2005년 3월~현재 다물 멀티미디어 근무중
<관심분야> CPU 및 SoC 설계

유 용 호 (Yong-Ho You) 정회원

1990년 2월 광운대학교 전자통신공학과 졸업
1994년 2월 광운대학교 전자통신공학과(석사)
1993년 1월~2000년 7월 LG전자 우면동연구소 선임연구원
2000년 7월~2003년 4월 Infinior Microsystem 수석연구원
2002년 3월~현재 서강대학교 전자공학과 박사과정
2004년 11월~현재 Mimow Tech. Vancouver Canada. 수석연구원
<관심분야> 마이크로프로세서 구조, 컴파일러 자동생성

황 선 영 (Sun-Young Hwang) 정회원

1976년 2월 서울 대학교 전자공학과 졸업
1978년 2월 한국과학원 전기 및 전자공학과(공학석사)
1986년 10월 미국 Stanford 대학 전자공학(박사)
1976년~1981년 삼성반도체 주식회사 연구원, 팀장
1986년~1989년 Stanford 대학 Center for Integrated System 연구소 책임 연구원 Fairchild Semiconductor Palo Alto Reaserch Center 기술자문
1989년~1992년 삼성전자(주) 반도체 기술 자문
2002년 4월~2004년 2월 서강대학교 정보통신대학원장
1989년 3월~현재 서강대학교 전자공학과 교수
<관심분야> 컴퓨터 시스템. SoC 설계 및 framework 구성, Embedded System 설계 등