특 집

임베디드 시스템을 위한 ASIP 설계 방법론

백 윤 흥*

● 목 차 ●

1. 서 론

어떤 application을 위한 임베디드 시스템을 설계하는 것은 성능, 가격, 전력, 크기등과 같은 상당히 많은 요소들을 고려하여 미리 결정되어 있는 각각의 제약조건을 만족시키는 가장 좋은 것을 찾아내는 것이다. 사람들의 생활수준이 높아짐에 따라 새롭고 복잡한 application이 등장하고 요구조건 역시 높아짐에 따라 이러한 application을 위한 임베디드 시스템의 복잡도 역시 매우 높아지게 되었고 앞으로 이런 추세는 더욱 가속화되리라 예상한다. 이러 다양한 application의 specification에 요구되는 성능, 가격 등의 조건을 충족하면서 설계되는 임베디드 프로세서를 ASIP (application specific instruction-set processor)라고 부르며, 이러한 ASIP의 설계자는 상황에 따라 고칠 수 있는 매우 유연한 형태의 구현 모델을 필요로 한다[1].

반도체 공정기술의 발전에 따라 주어진 칩 크기에 보다 많은 기능을 구현할 수 있게 되었다. 결과적으로 어떤 application을 위한 시스템이 하나의 칩으로 구현되는 System-On-Chip이 가능하게 되었다. 그러나 설계 기술이나 방법론이 공정상의 발전을 따라가지 못하여 많은 칩 area를 낭비하고 있다. 이러한 한계를 극복하고자 다양한 설계방법론이 제시되고 있는데 co-design[2]과 design 재사용[3]이 대표적인 예이다. 현재의 추세는 새로운 하드웨어의 개발은 많은 시간과 높은 비용을 요구하므로 소프트웨어를 기반으로 한 임베디드 시스템 개발이 더욱 활발해지고 있다. 이것이 가능한 이유는 ASIP의 성능의 향상과 공정기술의 발전에 따른 size와 cost의 향상에 있다.

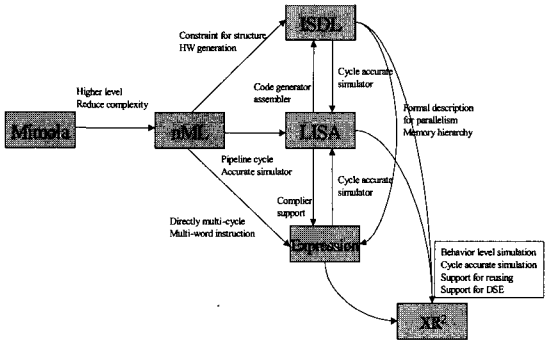
본 논문에서는 위에서 설명한 바와 같이 임베디드 시스템에서 중요한 역할을 차지하는 소프트웨어 부분인 ASIP의 개발을 위해 유용한 개발환경에 대해 설명할 것이다. 다양한 application을 위한 ASIP의 용이한 개발을 위하여 ASIP를 기술할 수 있는 언어와, 이러한 언어로 기술된 ASIP 모델로부터 자동으로 시뮬레이터와 컴파일러가 합성되는 방법론에 대해 소개하고자 한다. 다음 장에서는 관련연구에 대해서 간략히 보도록 한다. 3장에서는 READ(REusable Architecture Description language)[4]로부터 발전된 XR^2(eXtensible, Reconfigurable, Reusable)의 전체적인 개발 환경에 대해서 설명될 것이다. 4장과 5장에서는 top-down디자인 환경의

* KAIST 전자전산학과 조교수

두 중요한 단계인 HighXR²와 LowXR²에 대해서 각각 설명될 것이다. 6장에서는 결론을 내리고 향후 과제에 대해 설명함으로써 본 논문을 마치고자 한다.

2. 관련 연구

지난 몇 년 동안 ASIP 구조를 기술하기 위한 여러 가지 기술언어가 제안되었다. MIMOLA[5]는 프로세서를 net list로 기술하고 이러한 하위 레벨 프로세서기술로부터 명령어 집합을 이끌어내고 코드 생성기를 만들어낸다. 그러나 복잡한 명령어의 경우는 제대로 이끌어 내기 힘들고 결과적으로 코드 생성기의 성능은 떨어지게 된다. nML[6]은 프로세서의 명령어 집합과 어드레싱 모드집합을 기술하여 컴파일러를 생성하고 명령어 수준의 시뮬레이션을 수행하게 된다. 그 이후에 파이프라인 수준의 프로세서 모델링을 수행하여 cycle-accurate한 시뮬레이션을 하게 된다. EXPRESSION [7], LISA[8] 그리고 ISDL[9]은 cycle-accurate한 시뮬레이터와 최적화 컴파일러를 지원할 정도의 low level에서의 프로세서 모델링을 지원한다. 그러나 명령어 수준의 시뮬레이터는 지원하지 않기 때문에 초기 명령어 집합을 만들어내는 데에도 큰 노력이 들게 된다. 특히 ISDL의 경우에는 타겟을 VLIW구조를 갖는 프로세서만으로 제한을 둔다. 이러한 프로세서 기술



(그림 1) 프로세서 기술언어들의 특징

언어들의 표현력에 대한 관계는 (그림 1)에 잘 나타나 있다. 우리의 프로세서 기술 언어인 XR2는 위와 같은 명령어 수준, cycle-accurate한 수준의 프로세서 모델링과 그에 따른 retargetable 시뮬레이터와 컴파일러를 지원한다. 또한 빠른 파이프라인 구조의 탐색을 위해 프로세서의 새로운 추상레벨을 제안하고 있다.

3. 프로세서 기술 언어와 설계 방법론

프로세서 기술을 통하여 효과적으로 ASIP를 개

<표 1> 시뮬레이션의 목적과 관계된 ASIP의 특징 및 설계 단계와의 관계

Objective/feature	Required Semantics	Design stage	Meaning
Instruction correctness	Per instruction behavior	HighXR²	Instruction set architecture
Addr. mode correctness	Per addressing mode behavior		
Compiled code size	ISA (not pipeline)		
Execution profile	ISA (not pipeline)		
Total cycle count	Delay model for pipeline architecture	Token-level LowXR²	Pipeline architecture VLIW Superscalar
Rough area estimation	Area estimated value for resource		
Rough power estimation	Power estimated value for resource		
Pipeline arch. decision	Delay model for pipeline architecture		
Resource usage	Relation between instruction/addr model and resources in pipeline architecture		
Cycle-true snapshot	Cycle accurate pipeline model	Cycle-accurate LowXR²	
Exact area estimation	HDL models	MicroXR²	Implementation
Exact power estimation	HDL models		

발하기 위해서는 적절한 설계 단계로 이루어진 top-down 설계 환경이 필요하다. 이러한 설계 환경을 구축하기 위하여 <표 1>에서는 프로세서를 모델링 및 시뮬레이션 하는 목적과 이러한 성능 지수를 측정하기 위하여 필요한 ASIP의 특징들을 볼 수 있다.

가령 어떤 application의 컴파일된 코드 크기를 얻어내기 위해서는 프로세서의 컴파일러를 구축할 수 있는 ISA (instruction set architecture)가 필요하다. 이러한 프로세서의 특징들을 상위 추상레벨부터 하위 추상레벨까지 나열한 후 특징에 따라 잘 나누어 설계자가 의도하는 성능 지수를 보다 효율적으로 얻을 수 있도록 top-down설계 환경을 구축하여야 한다. <표 1>의 3번째와 4번째 열에서 이러한 관계를 보이고 있다. 각 설계 단계에 대해서 다음 두 장에서 조금 더 자세하게 보도록 하자.

4. HighXR^2

ASIP를 설계할 때 application을 잘 수행할 수 있는 명령어 집합과 어드레싱 모드를 결정하는 것은 컴파일러 개발자, 시뮬레이터 개발자는 물론 소프트웨어를 설계하는 사람 모두에게 매우 중요하다. 이 설계 단계에서는 결정하는 것은 다음과 같은 프로세서의 high level ISA이다.

HiISA = < IS, AM, ST, RIA, RAS > where

IS^{Σ} : A structured set for the instructions of target architecture

AM^{Σ} : A set of addressing modes of target architecture

ST : A set of the storages of target architecture

$R_{IA} \subseteq IS \times AMn$: relation between instruction and addressing modes.

$R_{AS} \subseteq AM \times ST$: relation between addressing modes and storage elements

HighXR^{2}는 위와 같은 semantics를 갖는 형태의 언어이다. 이 언어로 기술된 프로세서 모델로부터 시뮬레이터와 컴파일러가 자동 생성된다. 각각에 대해서 알아보도록 하자.

4.1 HighXR^{2}를 위한 Retargetable 시뮬레이터

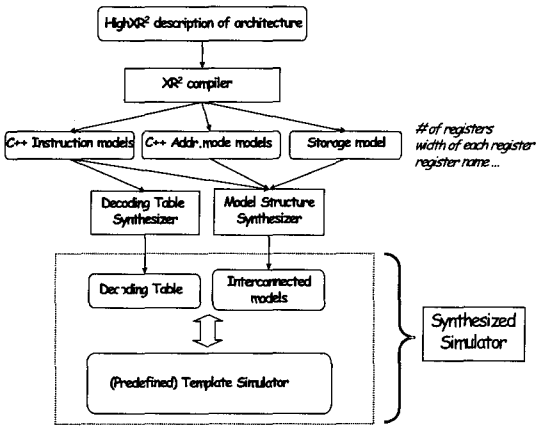
Retargetable 시뮬레이터는 HighXR2로 기술된 프로세서 모델로부터 자동 생성된다. 어셈블리 프로그램을 입력으로 받아들여 명령어 수준의 시뮬레이션을 수행함으로써 application이 정확한지와 어떤 명령어가 많이 사용되었는지 알 수 있게 된다.

(그림 2)는 retargeting하는 방법을 보여주고 있다. 시뮬레이션 엔진은 fetcher, decoder와 같은 control path, 그리고 프로그램/데이타 메모리, 레지스터와 같은 파라미터화된 storage element들로 이루어져 있다. HiISA 모델 생성기는 명령어, 어드레싱 모드 그리고 storage에 대한 C++ code를 생성해준다. Model-structure 생성기는 이러한 C++모델을 가지고 HighXR2에서 기술된 대로 전체 구조를 생성해낸다. 그리고 decoding table생성기는 시뮬레이터가 주어진 mnemonic이나 op code에 대해 적당한 명령어를 찾아주도록 decoding table을 생성해준다.

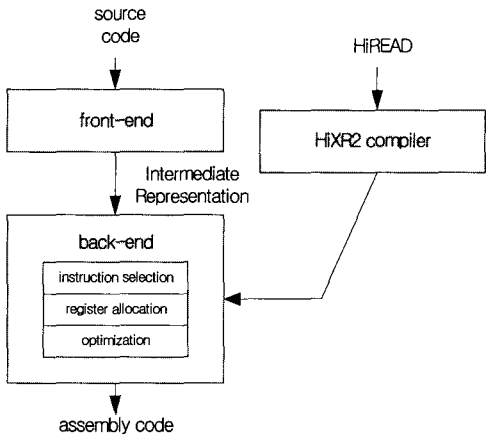
4.2 HighXR^{2}를 위한 Retargetable 컴파일러

Retargetable 컴파일러는 HighXR^2 기술로부터 해당 프로세서를 위한 컴파일러를 자동 생성한다. (그림 3)은 retargetable 컴파일러의 framework을 보여주고 있으며, 컴파일러의 기계 종속적인 부분들은 HighXR2 컴파일러라 불리는 컴파일러 생성자가 HighXR2 기술로부터 자동적으로 retargeting하게 된다.

(그림 3)에서처럼 컴파일러에서 기계 종속적인 부분들은 back-end에 있으며, back-end는 크게 코드 생성과정인 명령어 선택과 레지스터 할당과 여러



(그림 2) 시뮬레이터 생성기 구조



(그림 3) Retargetable 컴파일러 Framework

가지 최적화 과정들로 구성되어 있다. HighXR2 컴파일러는 HighXR^{2}로부터 ISA와 그 behavior를 알기 때문에, 명령어 선택과 레지스터 할당을 해당 프로세서에 맞게 retargeting하게 된다. 최적화 과정은 크게 기계 종속적인 것과 비종속적인 것으로 나뉘는데, HighXR^{2}에서는 명령어에 대한 latency나 pipeline에 대한 정보가 없기 때문에 기계 비종속적 최적화 과정만 수행하고 기계 종속적인 최적화 과정은 LowXR^{2}가 있어야 이루어질 수 있다. 예외적으로 기계 종속적인 최적화 과정이 HighXR^2 기술로부터 얻어지는 경우도 있는데, 이는 매우 기술하기 어려운 복잡한 명령어는 parameterization에 의해서 기술되기 때문에, 이와 같은 명령어를 선택하기 위한 기계 종속적인 최적화 과정은 HighXR^{2}의 기술로부터 얻어지게 된다. 대표적인 예로 zero-overhead loop을 위한 명령어를 들 수 있다.

5. LowXR^2

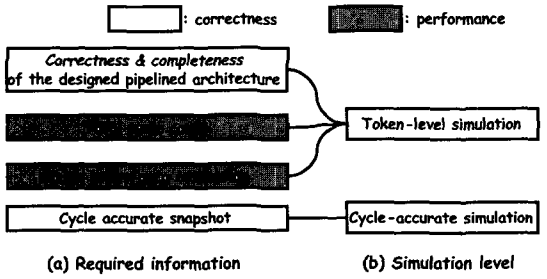
LowXR^{2}는 HiISA에서 결정한 명령어 집합과 어드레싱 모드 집합에 대해 프로세서의 파이프라인 구조, VLIW 및 superscalar와 같은 data path를 기술하는 언어이다. 이 추상레벨에서의 시뮬레이션은 다음과 같은 두 가지 목적을 가지고 있다.

- 프로세서 구조의 설계 공간이 매우 넓기 때문에 빠른 architecture evaluation을 할 수 있어야 한다.
- 프로그램과 프로세서의 디버깅을 위하여 cycle-accurate한 시뮬레이션을 할 수 있어야 한다. Storage element들의 cycle-accurate한 정보는 프로세서를 HDL로 구현할 때 참조 정보가 된다.
- 컴파일러에게 data path에 대한 정보를 제공하여, 컴파일러가 기계 종속적인 최적화를 하도록 한다.

그러나 첫 번째 목적과 두 번째 목적은 시뮬레이션 속도에 있어서 상충된다. 즉 매 싸이클마다 정확한 정보를 얻어내기 위한 시뮬레이션은 시간이 지나치게 오래 걸리게 된다. 이러한 이유로 (그림 4)과 같이 이 단계의 모델링 수준을 두 수준으로 다시 나누었다. Token-level modeling/simulation은 빠른 시뮬레이션 속도를 특징으로 한다. 가정하고 있는 모델은 각 스테이지의 latency정보가 있는 파이프라인 구조이다. 매 클럭마다 모든 스테이지를 보고 내부에 명령어를 갖고 있는 스테이지에 대해 다음과 같은 동작을 취해준다.

현재 스테이지에 들어온 명령어가 그 스테이지

에 정의된 result latency이상 머물러 있었고 다음 스테이지에 입력으로 들어간 명령어가 들어가고 나서 그 스테이지에 정의된 issue latency이상의 시간이 흘렀으면 다음 스테이지로 명령어는 진행된다.



(그림 4) 파이프라인 구조 설계 단계

각 명령어에 정의되어 있는 파이프라인의 latency정보에 의해서만 명령어가 토큰이 전해지는 것과 같은 방법으로 시뮬레이션된다. 이 경우 아무런 메모리 값의 계산이 이루어지지 않기 때문에 상당한 시뮬레이션 속도 향상을 가져오게 된다. 이 때 사용하는 application은 벤치마크 코드가 아니라 HiXR^2 시뮬레이터에서 시뮬레이션되어 명령어 sequence가 정해진 형태를 사용한다. 그 이유는 token-level LowXR^2 시뮬레이션에서는 값에 대한 계산이 이루어지지 않기 때문에 branch와 같은 경우 target을 결정할 수 없기 때문이다.

컴파일러에서는 LowXR^{2}에서 기술되는 micro-architecture를 통하여 기계 종속적인 최적화를 할 수 있게 된다. 즉, data-path와 각 명령어의 cycle-accurate한 정보를 얻을 수 있기 때문에, 어떠한 최적화 과정을 적용하기 전에 효과적인지 아닌지에 대한 판단을 할 수 있으며, pipeline 기술로부터 resource model을 얻을 수 있고, HighXR2로부터 명령어 형식에 대한 정보를 얻을 수 있기 때문에 ILP를 위한 명령어 scheduler도 자동적으로 생성할 수 있다.

6. 결론 및 향후 과제

본 논문에서는 임베디드 시스템에서 소프트웨어에 해당하는 프로세서를 설계하는 방법론에 대해서 살펴보았다. 이 환경은 다음과 같은 장점을 갖는다.

첫 번째로 프로세서를 기술할 수 있는 언어를 계층적으로 두어 상위 설계 단계에서 결정한 것을 가지고 하위 설계 단계에서 refine하게 된다. 가령 HighXR2로 기술된 명령어 집합과 어드레싱 모드 집합을 가지고 다양한 파이프라인 구조에 대해 모델링하고 시뮬레이션함으로써 보다 빠른 프로세서 설계를 가능하게 해준다.

다음으로 각 단계에서는 공통적인 프로세서 기술을 가지고 시뮬레이터/컴파일러를 구축하기 때문에 기존에 공통된 specification만을 갖고 서로 시뮬레이터와 컴파일러를 독립적으로 구축하던 방법에 비해 에러가 많이 줄어들게 된다.

마지막으로 다른 방법론에서는 지원하지 않는 token-level 모델링/시뮬레이션을 지원함으로써 보다 빠른 architecture evaluation을 가능하게 해 준다는 점이다. 이것이 유용한 이유는 application이 커짐에 따라, 그리고 설계하고자 하는 프로세서가 복잡해짐에 따라 architecture의 설계 공간이 커지게 되고 그 중에 최적의 설계를 찾기 위해서는 많은 architecture에 대한 모델링과 시뮬레이션이 필요하게 되기 때문이다.

본 연구는 현재 우리 KAIST 연구진에의해 계속 진행 중이다. Token-level LowXR^{2}를 이용하여 프로세서의 구조를 결정한 후 cycle-true 정보를 얻어내기 위한 Cycle-true LowXR^{2}를 정의하고 관련 소프트웨어를 구축하고 있다. 이 모델을 reference model로 하여 MicroXR^{2}에서 프로세서를 구현하는 방법론을 제시할 계획이다. 또한 LowXR^{2}를 이용하여 기계 종속적인 최적화 과정을 자동적으로 수행하는 것에 대한 연구와 함께 디지털 신호 처리기

(Digital Signal Processor)와 같은 복잡한 임베디드 프로세서에서 자동화된 컴파일러가 어떻게 최적화된 코드를 얻을 것인가에 대한 연구도 수행중이다. 보다 이 연구에 대한 최근 결과 및 논문은 웹사이트 http://soar.kaist.ac.kr 에서 볼 수 있다.

참고문헌

[1] Alberto Sangionvanni-Vincentelli, Grant Marthin, "A Vision for Embedded Software," Proceedings of the international conference on Compilers, architecture, and synthesis for embedded systems, Nov. 2001

[2] P.Subrahmanayam, Hardware-Software Codesign : Cautious Optimism for the Future, IEEE Computer, 26(1), pp84-85, 1993

[3] Thomas, T. Technology for IP reuse and Portability, IEEE Design and Test of Computers, Volume 16, Issue 4, pp7-13, Oct 1999

[4] Young Geol Kim, Tag Gon Kim, "A Design and Tool Reuse Methodology for Rapid Prototyping of Application Specific Instruction Set Processors," Rapid System, Prototyping, pp46-51, 1999

[5] R.Leupers and P.Marwedel, "Retargetable code generation based on structural processor description," Design Automation for Embedded Systems, 3(1), 1998

[6] A.Faulth, J.Van Praet, M.Freericks, "Describing instruction set processors using nML," Proceedings of European Design and Test Conference, pp503-507, Paris France, 1995

[7] Ashok Halambi et al., "EXPRESSION, A Language for Architecture Exploration through Compiler/Simulator Retargetability," Proceedings of Design, Automation and Test in Europe Conference and Exhibition, pp.485-490, 1999

[8] V.Zivojnovic et al., "LISA Machine Description Language and Generic Machine Model for HW/SW Co-design," IEEE Workshop on VLSI Signal Processing, 1996

[9] G.Hadjiyiannis et al., "ISDL: An Instruction Set Description Language for Retargetability," Proceeding of DAC, 1997

저자약력

백 윤 흥

1988년 서울대학교 컴퓨터공학과 (공학사)
1991년 서울대학교 컴퓨터공학과 (공학석사)
1997년 University of Illinois at Urbana-Champaign, Computer Science Department (Ph.D)
1997년-1999년 New Jersey Insititue of Technology, Computer Science & Information Department, Assistant Professor
1999년-현재 KAIST 전자전산학과 조교수
관심분야 : 컴파일러, 임베디드시스템 소프트웨어, 프로그래밍 언어
e-mail : ypaek@soar.kaist.ac.kr