

단일 칩 다중프로세서의 설계

이상원, 김영우, 오형철¹, 김수원, 한우종², 윤석한²

고려대학교 전자공학과 ASIC 연구실

¹고려대학교 정보공학과 병렬연산연구실

²한국전자통신연구원 컴퓨터시스템연구부

Tel. 02-923-2081, Fax. 02-928-1216, Email. swlee@asic.korea.ac.kr

Design of an On-Chip Multiprocessor

Sang-Won Lee, Young-Woo Kim, ¹Hyeong-Cheol Oh,

Soo-Won Kim, ²Woo-Jong Hahn and ²Suk-Han Yoon

ASIC Laboratory, Department of Electronic Engineering, Korea University

¹Parallel Computation Lab., Department of Information Engineering, Korea University

²Computer System Research Department, ETRI

Tel. 02-923-2081, Fax. 02-928-1216, Email. swlee@asic.korea.ac.kr

ABSTRACT

This research aims at developing a single chip multiprocessor for high-performance computer system. Our design approach is to design a relatively small and simple processor unit and to integrate multiple copies of the unit in an efficient way. The proposed multiprocessor is composed of four CPUs and one graphic coprocessor. The four CPUs share the graphic coprocessor and each CPU implements the 64-bit SPARC-V9 instruction set architecture.

This paper gives an overview of the proposed microarchitecture and discusses the considerations made in the course of the design.

1. 서론

반도체 공정 기술의 발달로 VLSI의 기본 단위의 크기가 지속적으로 줄면서 동일 면적에 집적할 수 있는 트랜지스터의 수는 꾸준히 증가하고 있다. 현재의 기술 추세로 볼 때 약 10억 개의 트랜지스터를 단일 칩에 집적하는 것이 곧 현실화 될 전망이다. 이러한 고 집적 트랜지스터들을 효과적으로 이용할 수 있는 병렬

처리 방법이 요구 되고 있다.

오늘날 대부분의 프로세서들은 슈퍼스칼라 구조로 명령어 수준의 병렬성(Instruction Level Parallelism)을 극대화하여 처리하고 있으며 성능을 향상시키기 위한 여러 가지의 복잡한 명령어 처리 과정을 포함하고 있다. 그러나 연구 결과에 따르면 4개의 명령어를 한 사이클에 처리하는 4-way 슈퍼스칼라 이상의 구조에서는 추가적인 성능향상이 매우 적으며 평균적으로 2개 이하의 명령어를 처리하는 것으로 나타나고 있다[2][3]. 이러한 배경에서 최근 스레드 수준의 병렬성(Thread Level Parallelism)에 대한 관심이 고조되고 있다.

본 논문에서 제안하는 단일 칩 다중프로세서는 스레드 수준의 병렬성을 이용한 구조로 설계가 비교적 간단하다는 장점을 가지며 프로세서 코어 별로 설계가 분리되어 있으므로 10억 개 이상의 트랜지스터로 구성되는 고집적 칩 구현에 적합하다. 본 논문에서는 RAPTOR로 명명된 단일 칩 다중프로세서의 구조와 설계시 고려사항에 대하여 논한다.

2. RAPTOR의 구조

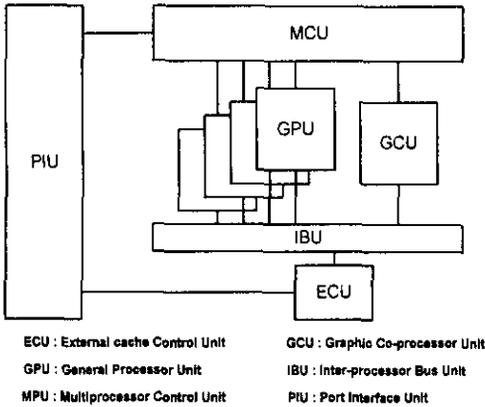
RAPTOR는 독립된 4개의 프로세서 유닛(General Processor Unit : GPU)과 한 개의 그래픽 연산 유닛(Graphic Co-processor Unit : GCU)으로 구성되는 단

일 칩 프로세서로서 주요 특징은 다음과 같다.

- 각각의 프로세서 유닛은 SPARC V9 명령어 셋 구조를 가지며 64비트 정수 및 부동 소수점 연산을 한다.
- 4개의 독립된 프로세서 유닛이 그래픽 연산 유닛을 공유하여 그래픽 연산을 처리한다.
- 크기가 비교적 작은 내장 1차 캐쉬와 대용량의 외부 2차 캐쉬로 구성된 다중 캐쉬 구조를 갖는다.
- 2차 캐쉬 제어기를 내장하며 여러개의 RAPTOR 프로세서를 이용한 시스템 구현을 지원한다.

2.1 RAPTOR의 블록도

RAPTOR의 주요 기능 블록들을 [그림 1]에 보았다.



[그림 1] RAPTOR의 블록도

각각의 GPU는 윈도우 형태의 레지스터 파일과 다수의 명령어 처리 유닛을 가지고 있으며, 하버드 구조의 1차 캐쉬를 내장하고 있다. 2차 캐쉬는 RAPTOR의 외부에 존재하며, 프로세서 내부의 ECU에 의하여 제어된다. 1차 캐쉬와 2차 캐쉬는 모두 Write-Back 프로토콜로 운영하며 데이터 전송을 가능한 한 최소화한다. 메모리 인터페이스는 PIU를 통하여 이루어지며 IBU를 통하여 각각의 프로세서에 전달된다. RAPTOR를 구성하는 각 블록의 기능들은 다음과 같다.

- 프로세서 유닛(General Processor Unit : GPU)
프로세서 유닛은 RAPTOR의 명령어 셋 중 그래픽 명령어를 제외한 모든 명령어를 수행하는 유닛이다. SPARC V9 명령어 셋 구조를 구현하며 각각의 프로세서 유닛은 8개의 윈도우 레지스터 파일로 구성

되었고 4레벨 트랩처리를 한다. 단일 파이프라인으로 구성 되었으며 비 순차적 명령어 수행 및 프리사이즈 트랩(Precise Trap)을 지원한다. 그래픽 명령어의 디코딩은 프로세서 유닛에서 이루어진다.

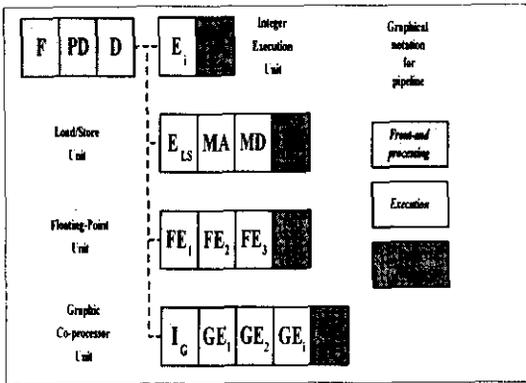
- 그래픽 연산 처리 유닛
(Graphic Co-processor Unit : GCU)
그래픽 연산 처리 유닛은 RAPTOR의 명령어 셋 중 그래픽 명령어를 수행하는 유닛이다. 4개의 프로세서 유닛은 그래픽 연산 처리 유닛을 공유하며 각각 처리할 그래픽 명령어를 그래픽 연산 처리 유닛으로 전달한다. 그래픽 연산 처리 유닛으로 전송된 그래픽 명령어는 중재와 스케줄링을 통하여 결정된 처리 순서대로 차례로 수행되며 연산 결과는 명령어 처리를 의뢰한 프로세서 유닛으로 보내게 된다.
- 멀티프로세서 제어 유닛
(Multiprocessor Control Unit : MCU)
멀티프로세서 제어 유닛은 인터럽트의 분배, 프로세서들의 초기화, 동기화 및 오류 처리 등의 다중프로세서 환경에서 요구되는 동작을 제어한다.
- 외부 캐쉬 제어 유닛
(External cache Control Unit : ECU)
외부 캐쉬 제어 유닛은 4개의 프로세서 유닛에 의해서 공유되는 2차 캐쉬 메모리를 제어하여 2차 캐쉬로부터 전송되는 데이터의 처리 및 캐쉬 일관성(Coherency)을 유지하는 기능을 한다.
- 내부 버스 유닛(Internal-processor Bus Unit : IBU)
내부 버스 유닛은 메모리 또는 2차 캐쉬로부터 전송된 데이터를 프로세서의 내부에 있는 4개의 프로세서 유닛에 전송하는 프로토콜 및 프로세서 내부의 1차 캐쉬의 일관성 유지 프로토콜을 정의한다.
- 포트 인터페이스 유닛(Port Interface Unit : PIU)
포트 인터페이스 유닛은 메모리로부터의 데이터 전송 및 그 밖의 외부 인터페이스를 처리하는 기능을 한다.

2.2 파이프라인 스테이지

RAPTOR의 파이프라인 스테이지를 [그림 2]에 보았다. RAPTOR의 파이프라인은 연산 유닛에 따라 다양한 파이프라인 단계를 가지며 각 연산 유닛에서 처리된 연산이 이슈된 순서에 관계없이 종료될 수 있는 구조이다. 비 순차적으로 수행이 완료된 연산 결과는 프로세서 유닛 내부의 리오더 버퍼(Reorder Buffer)에

의해서 순차적으로 레지스터 파일에 저장된다. 이렇게 비 순차적 연산 종료를 하드웨어적으로 처리함으로써 그래픽 연산 처리 유닛의 공유가 효과적으로 이루어진다.

RAPTOR의 파이프라인은 크게 전처리단계(Front-End Processing), 실행단계(Execution), 저장단계(Retire or Write)의 세 가지 단계로 나누어 볼 수 있다. 전처리 단계에서는 명령어의 페치(Fetch)와 디코드(Decode) 그리고 레지스터 파일에 대한 액세스가 수행되며 실행단계에서는 각 기능 유닛에 따른 명령어의 연산이 수행된다. 마지막 단계인 저장 단계에서는 비 순차적으로 종료되는 명령어들의 연산 결과 값들이 리오더 버퍼로 저장된다.

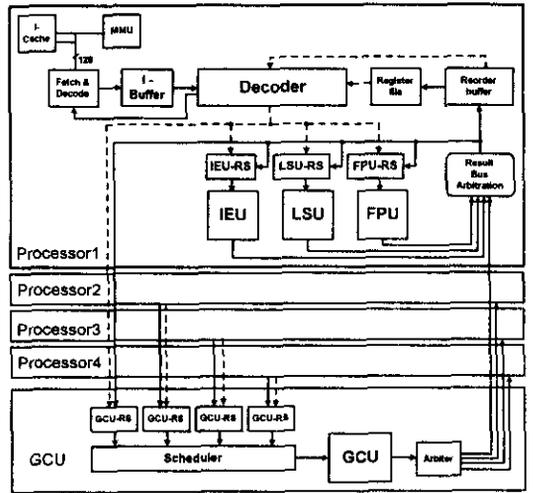


- MA Access TLB, Check hit/miss
- PD Pre-decode instructions
- MD Access Cache
- D Decode instruction, Register file access, Dependency check, Issue decoded instruction to reservation station
- FE₁ Execution in FPU
- FE₂ Execution in FPU
- FE₃ Execution in FPU
- IC Execution in GCU
- GE₁ Execution in GCU
- GE₂ Execution in GCU
- GE₃ Execution in GCU
- W Write results to reorder buffer, Report exception
- F Fetch instruction from I-cache
- E₁ Execution in integer ALU in IEU
- E_{LS} Execution in LSU, Calculate address for load/store

[그림 2] RAPTOR의 파이프라인 스테이지

2.3 명령어 처리 유닛

RAPTOR의 프로세서 유닛(GPU)은 정수 연산 처리 유닛(Integer Execution Unit : IEU), 부동 소수점 연산 처리 유닛(Floating Point Unit : FPU), 로드/스토어 유닛(Load/Store Unit : LSU)의 세 가지 연산 유닛으로 구성되고 그래픽 연산 처리 유닛을 포함하여 모두 4개의 연산 유닛에서 명령어의 처리가 이루어진다. [그림 3]은 명령어 처리 유닛의 블록도이다.



[그림 3] 프로세서 유닛의 블록도

2.4 프로세서 인터페이스

RAPTOR의 구조는 4개의 프로세서 유닛이 데이터 전송 버스를 공유하는 구조로서, 공유되는 버스의 전송 능력이 전체 시스템 성능에 매우 큰 영향을 미칠 수 있는 구조이다. 본 연구에서는 공유 버스를 설계하는데 있어 전송의 지연시간(Latency)과 전송율(Throughput) 두 가지 측면을 모두 고려하였으며 2차 캐쉬가 충분히 크고 2차 캐쉬로부터의 전송 시간이 짧은 가정 하에 RAPTOR 구조에 적합한 버스 프로토콜을 설계하였다.

내부 버스는 4개의 프로세서와 외부에서의 스누우프(Snoop) 요구에 의해서 점유된다. 내부 버스의 사용 요구는 전송의 종류, 대상 등에 따라 우선 순위가 결정되는데 동시에 발생된 여러 점유 요구 중에서 외부 스누우프 요구, Lock 사이클 등이 우선 순위를 가진다. 이렇게 우선 순위를 줌으로써 전송의 효율을 높이고 Deadlock 등의 발생을 방지한다.

2.5 캐쉬 구조 및 일관성 프로토콜

RAPTOR 는 각각 16 킬로바이트 크기를 가지는 1 차 데이터/명령어 캐쉬와 4 메가바이트의 크기를 가지는 외부 2차 캐쉬의 다중 캐쉬 구조를 가지며 1차, 2차 캐쉬는 모두 직접 사상(Direct-mapped) 방식으로 설계 되었다. 직접 사상 방식은 셋-어소시어티브(Set-Associative) 방식에 비하여 블록교체

(Replacement) 알고리즘이 필요 없고, 히트(Hit) 결정이 빠른점 등의 장점이 있으나 히트율이 다소 떨어질 수 있다. 그럼에도 불구하고 설계가 용이하고, 2차 캐쉬의 크기가 충분히 크고 전송이 빠를 경우 1차 캐쉬의 미스 페널티(Penalty)가 작다는 점 등을 고려하여 본 연구에서는 직접사상 방식을 채택 하였다.

RAPTOR의 캐쉬의 일관성 프로토콜(Cache Coherency Protocol)을 결정하는데 있어 가장 중요하게 고려한 사항은 내부 버스의 전송을 가능한 최소화하는 것이다. 이러한 방안으로 1차 캐쉬로의 불필요한 스누우프를 최소화하는 것이 중요한데 1차 캐쉬의 내용을 2차 캐쉬가 모두 포함하는 캐쉬 인크루전(Inclusion)과 Write Once 프로토콜 등을 이용하여 불필요한 스누우프를 최소화하였다.

3. 설계 및 검증

RAPTOR의 설계는 탑-다운(Top-Down) 방식으로 진행되었다. 설계사양에 따라 탑 블록이 정의되면 기능별로 세분화하여 기능별 유닛들과 이들 유닛간의 인터페이스 신호가 정의되었다. 정의된 유닛의 기능은 HDL (Hardware Description Language)을 이용하여 기술되고 각 유닛 별 기능 검증이 진행되었다. 유닛별 기능 검증이 끝나면 각 유닛들이 통합되고 통합 시뮬레이션을 하는 과정으로 전체 시스템의 동작이 검증된다.

RAPTOR는 Verilog HDL로 RTL 수준으로 기술되었으며 자기 진단 방법(self-checking method)과 기준 모델 비교 방법(reference model comparison method)으로 검증되었다. 자기 진단 방법을 위해서 구조 검증 프로그램들이 개발되었고, 기준 모델 비교 방법을 위해서 RapSim으로 명명된 구조 시뮬레이터가 개발되었다.

4. 결론 및 향후계획

본 논문에서는 단일 칩 다중프로세서 구조의 RAPTOR를 제안하고 설계 고려 사항을 논하였다. 제안된 구조는 비교적 단순한 구조의 프로세서 4개를 결합하여 성능을 향상시키면서 한편으로 구현이 용이한 프로세서 구조이다. 이러한 구조의 프로세서는 멀티쓰레딩(Multithreading) 프로그램 처리에 적합한 장점을 가진다. 제안된 프로세서는 Verilog HDL로 설계

되었으며 다양한 기능 검증 방법으로 검증되었다.

설계된 프로세서는 논리 합성 프로그램을 이용하여 논리회로로 합성하여 구현하는 것을 계획하고 있고 한편으로 적절한 벤치마크 프로그램을 사용하여 성능을 평가하는 방법을 모색하고 있다.

5. 참고문헌

1. 박경, 최성훈, 한우종, 윤석한, "다중 처리 마이크로 프로세서 설계", 1996년도 대한전자공학회 추계학술대회 발표논문집 제 19 권 제 2 호, pp.717 - 720, 1996. 11.
2. Janet Wilson, "Challenges and Trends in Processor Design," IEEE Computer, January 1998, pp.39-50.
3. Lance Hammond, Basem A. Nayfeh and Kunle Olukotun, "A Single-Chip Multiprocessor," IEEE Computer, September 1997.
4. Saman P. Amarshinghe et al., "Multiprocessors From a Software Perspective," IEEE Micro, Vol.8, No.6, pp.52-61, June 1996.
5. L. Gwennap, "Microprocessor head toward MP on a chip," Microprocessor report, Vol.8, No.6, pp.18-21, May 1994
6. David A. Patterson, and John L. Hennessy, *Computer Architecture A Quantitative Approach*, 2nd Ed, Morgan Kaufmann Publishers, Inc, 1996.