패킷 프로세싱을 위한 새로운 명령어 셋에 관한 연구

정회원 정 원 영*, 이 정 희**, 이 용 석*

A Novel Instruction Set for Packet Processing of Network ASIP

Won-young Chung*, Jung-hee Lee**, Yong-surk Lee* *Regular Members*

요 약

본 논문에선 기계 기술 언어(machine descriptions language)인 LISA(Language for Instruction Set Architecture)를 통하여 시뮬레이션 모델로 설계한 새로운 네트워크 ASIP(Application Specific Instruction-set Processor)을 제안한다. 제안한 네트워크 ASIP은 라우터(router)에서 패킷 프로세싱을 담당하는 전용엔진을 목적으로 설계되었다. 이를 위해 MIPS(Microprocessor without Interlock Pipeline Stages) 아키텍처를 기반으로 한 일반적인 ASIP에 패킷을 빠른 속도로 처리하기 위해 필요한 새로운 명령어 셋을 추가하였다. 새로 추가된 명령어 셋은 "classification" 명령어 그룹과 "modification" 명령어 그룹으로 나눌 수 있으며, 각 그룹은 실행 단계(execution stage)에 위치한 각각의 기능 유닛(function unit)에 의해서 처리된다. 그리고 각각의 기능 유닛은 Verilog HDL을 통해 면적과 속도 측면에서 최적화하였으며, 이를 합성하여 면적과 동작 지연시간을 비교하였다. 또한 CKF(Compiler Known Function)을 이용하여 C 언어 레벨의 매크로 함수에 할당하였으며, 어플리케이션 프로그램에 대한 실행 싸이클을 비교 분석하여 성능 향상을 확인하였다.

Key Words : ASIP, LISA, CKF, Network router, Packet processing

ABSTRACT

In this paper, we propose a new network ASIP(Application Specific Instruction-set Processor) which was designed for simulation models by a machine descriptions language LISA(Language for Instruction Set Architecture). This network ASIP is aimed for an exclusive engine undertaking packet processing in a router. To achieve the purpose, we added a new necessary instruction set for processing a general ASIP based on MIPS(Microprocessor without Interlock Pipeline Stages) architecture in high speed. The new instructions can be divided into two groups: a classification instruction group and a modification instruction group, and each group is to be processed by its own functional unit in an execution stage. The functional unit was optimized for area and speed through Verilog HDL, and the result after synthesis was compared with the area and operation delay time. Moreownr, it was allocated to the Macro function ana low-level standardized programming language C using CKF(Compiler Known Function). Consequently, we verified performance improvement achieved by analysis and comparison of execution cycles of application programs.

Ⅰ. 서 론

현재 광대역통합망(BcN : Broadband convergence Network)에 대한 연구가 계속 진행 중에 있으며, 이로 인해 확장된 망을 사용하는 트래픽의 양이 증가하고 있다. 특히 차세대 통신망으로 가는 과도기

※ 본 연구는 지식경제부 및 정보통신연구진흥원의 IT산업원천기술개발사업의 일환으로 수행하였습니다. [2009-S-043-01, Scalable 마이크로 플로우 처리 기술 개발]

* 연세대학교 전기전자공학과 프로세서 연구실(wychung@mpu.yonsei.ac.kr) ** 한국전자통신연구원(ETRI) 차세대 이더넷 팀(hilee@etri.re.kr)

논문번호 : #KICS2009-03-122, 접수일자 : 2009년 3월 14일, 최종논문접수일자 : 2009년 9월 14일

적 현상으로 네트워크 전반에 걸친 Alk)IP convergence 현상으로 인해 서킷 망이 패킷 망으로 통합되는 추이를 보이면서 IP 패킷을 기반으로 하는 트래픽의 양적 증가가 가속화 되고 있다. 특히 디지털 어플리케이션의 발달로 인하여 IPTV, VoIP, VOD, 온라인 게임 등의 멀티미디어 속성을 가진 트래픽이 빠른 속도로 증가하고 있다. 이러한 멀티미디어 속성을 가진 트래픽의 증가로 처리해야 할 패킷 개수의 증가뿐만 아니라 그에 따른 페이로드(IPye ad) 양의 증가, 그리고 분석해야 할 패킷 헤더(IPcke인해 header)의 비트 수가 증가하고 있다. 이와 같은 IP 패킷의 수적인 증가뿐만 아니라, 네트워크 망의 물리적인 속도가 계속 빨라지고 있어 라우터 및 스위치에서의 폭주(bur에) 현상이 잦아지고, 이로 인하여 버려지는(discard) 패킷이 빈번히 발생하고 있다.

이와 같은 현상으로 이미 접속 노드(access node)에 위치한 라우터(router)에는 패킷 처리 속도를 높이기 위해 각 포트(port) 별로 여러 개의 프로세서를 병렬로 두어 처리량(throughput)을 늘리는 방법이 보편화적으로 사용되고 있다. 이와 같은 프로세서를 ASIP (Application Specific Instruction-set Processor)으로 개발할 경우 범용 프로세서(general processor)나 ASIC(Application Specific Integrated Circuit)으로 개발할 경우에 비해 명령어 확장성이 뛰어나 네트워크만의 특화된 성능을 맞출 수 있다. 또한 최적화 과정을 통해 동작 주파수를 높일 수 있으며 면적 및 소비전력 감소가 용이하다.

본 논문에서 네트워크에 특화된 네트워크 ASIP을 개발하기 위해 일반적인 명령어를 지원하는 ASIP을 기반으로 네트워크 처리를 가속화 할 수 있는 하드웨어 가속화 엔진을 추가한 네트워크 ASIP을 시뮬레이션 모델로 구현하였다. 하드웨어 가속화 엔진은 NX(Network-eXtension : extension instruction set for network)이라 명명한 추가된 명령어에 의해 동작한다. NX 명령어는 분류(classification) 명령어 그룹과 변형(modification) 명령어 그룹으로 이루어져 있다. 또한 컴파일러의 CKF(Compiler Known Function)를 이용하여 상위 레벨의 함수에 NX 명령어를 매핑(mapping)하였고, 실제 네트워크 테스트벤치에 대한 프로세서의 성능을 비교하였다.

논문의 순서는 다음과 같다. 2장에서는 패킷 통신 기반의 라우터에서 자주 사용되는 연산을 설명하고, 3장에서는 이러한 연산을 지원해주는 NX 명령어를 제안한다. 4장에서는 기본적으로 설계한 범용 ASIP에 대한 설명과 NX 기능 유닛이 추가된 네트워크 전용 ASIP에 대한 설명이 있다. 또한 제안한 명령어를 한 싸이클에 처리할 수 있도록 최적화된 기능 유닛을 설명한다. 5장에서는 CKF를 이용한 매핑을 설명한다. 6장에서는 설계한 범용 ASIP과 네트워크 ASIP에 대한 비교 분석 및 실제 네트워크 테스트벤치에 대한 프로세서의 성능을 비교한다. 마지막으로 6장에서는 결론을 맺는다.

Ⅱ. 네트워크 연산

패킷 통신(packet communication)을 기반으로 하는 라우터에서는 입력된 다량의 패킷을 파싱(parsing), 분류(classification), 변형(modification)과 같은 과정을 정해진 시간 내에 처리해야 한다. 이때 마스킹, 비교, 특정비트의 변환 등의 연산이 빈번하게 수행된다.

2.1 분류 연산

분류(classification) 단계에서는 파싱(parsing)에서 추출되어진 헤더의 정보 중 IPv4의 발신 주소, 수신 주소, 발신 포트, 수신 포트, 플래그의 5-튜플(tuple)이나 IPv6의 제어 플로우(control flow)를 확인하여 룰을 확인하게 된다. 이를 위하여 테이블 룩업(lookup)을 하여 들어온 패킷에 대해 어떻게 처리할지 결정하는데, 소프트웨어 처리 시에는 그림 1과 같이 LPM(Longest Prefix Match) 방법이 널리 사용되고 있다. 이는 테이블을 검색하여 일치하는 여러 개의 엔트리(entry) 중 가장 많이 일치하는 엔트리의 대응 값을 읽어 룰을 결정하게 된다. 그 룰에 따라 패킷을 어떻게 처리할지 판단하게 된다. 이러한 연산들은 대부분 두 데이터 혹은 두 데이터의 특정부분이 얼마나 일치하는지를 알기위해 XOR를 통해 비트별 비교 연산을 실시하여 MSB에서부터 동일한 개수를 세기위해 CLZ(Count Leading Zero)를 실시한다.

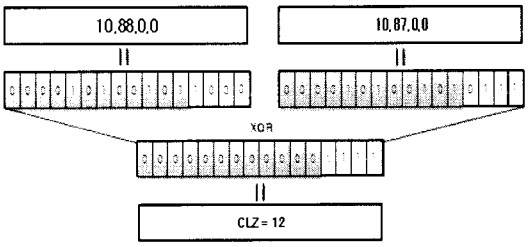


그림 1. LPM(Longest Prefix Match)의 예
Fig. 1. A example of LPM(Longest Prefix Match)

2.2 변형 연산

변형(modification)이란 라우터에서 패킷을 재가공하여 발송하는 단계에서 헤더의 일부 정보를 수정함을 의미하며, 이때 비트 스트림 연산이 빈번히 발생하게 된다. 주소를 대치하고 포트 정보를 수정하는 것은 마스크와 결합 연산으로 가능하지만, 특정 비트를 1이나 0으로 설정할 때, 특정 비트의 값을 인버팅해야 할 때, 또는 제로 패딩을 하기 위해선 비트 단위의 연산이 가능해야 한다.

Ⅲ. 제안하는 명령어

네트워크 트래픽 처리에 있어 빈번히 발생하는 연산을 지원하는 명령어로써 분류 단계를 위한 분류 명령어 그룹(classification instruction group)과 특정 비트의 속성을 변환시키는 변형 명령어 그룹(modification instruction group)을 제안하였다.

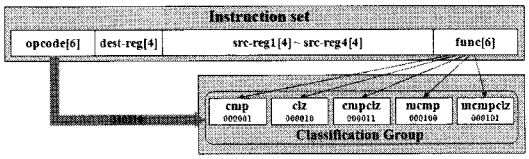
3.1 분류 명령어 그룹

분류 명령어 그룹은 총 5개의 명령어로 구성되어 있으며, 각 명령어에 대한 설명은 표 1에 명시되어 있다. 분류 명령어 셋의 구조는 그림 2에 도식화 되어 있다. CLZ 명령어는 소스 레지스터가 하나이며, CMP와 CMPCLZ 명령어는 두 개의 소스 레지스터를 갖으며, MCMP와 MCMPCLZ 명령어는 네 개의 소스 레지스터를 갖는다.

그림 3은 분류 명령어 그룹 중에서 가장 복잡도가 높은 MCMPCLZ의 동작 원리를 도식화 한 그림이다. mask1과 mask2는 각각 src_reg3, src_reg4에 해당한다. src_reg1과 src_reg3, src_reg2와 src_reg4를 비트별 AND연산을 한 후, 각각의 result1과 result2를 각 비트 별로 비교한 결과값을 MSB(Most Significant Bit)로부터 0의 개수를 카운트하여 출력하는 명령어이다.

표 1. classification 그룹을 구성하고 있는 명령어
Table 1. Instructions of a classification group

명령어	설 명
CMP	rs1의 값과 rs2의 각 비트를 비교하여 동일하면 rd = 0, 동일하지 않으면 rd = 1
CLZ	rs1의 값에서 1이 시작되지 전까지의 0의 개수를 카운트 (count leading zero)
CMPCLZ	cmp + clz
MCMP	소스에 해당하는 rs1과 rs2를 각각 마스크에 해당하는 rs3과 rs4 로 마스킹(masking) 한 후 cmp
MCMPCLZ	mcmp + clz



classification group	OP code	rd	rs1	rs2	rs3	rs4	function code
CMP	110110	o	o	o	x	x	000001
CLZ	110110	o	o	x	x	x	000010
CMPCLZ	110110	o	o	o	x	x	000011
MCMP	110110	o	o	o	o	o	000100
MCMPCLZ	110110	o	o	o	o	o	000101

그림 2. classification 명령어 셋 구조
Fig. 2. A structure of the classification instruction set

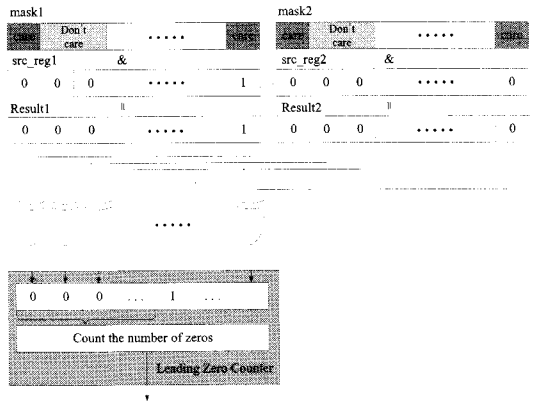


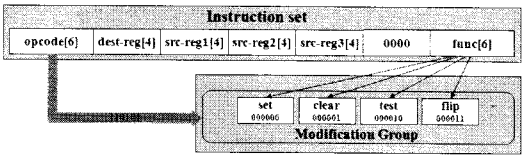
그림 3. MCMPCLZ 명령어의 동작 원리
Fig. 3. The operation of MCMPCLZ instructions

3.2 변형 명령어 그룹

변형 명령어 그룹은 표 2와 같이 총 4개의 명령어로 구성되어 있다. 그림 4는 변형 명령어 그룹의 명령어 셋의 구조를 보여준다. 명령어 셋은 32비트로 구성되어 있으며 6비트의 opcode, 각각 4비트의 목적지 레지스터(dest_reg), 소스 레지스터(src_reg) 1부터 3까지 총 4개의 레지스터로 구성되어 있다. 또한 LSB(Least Significant Bit)에서부터 6비트는

표 2. modification 그룹을 구성하고 있는 명령어
Table 2. Instructions of a modification group

명령어	설 명
SET	특정 비트들을 1로 세팅
CLEAR	특정 비트들을 0으로 클리어
TEST	특정 비트들의 각각의 비트를 비교하여 동일하면 rd = 0, 동일하지 않으면 rd =1
FLIP	특정 비트들의 각각의 비트들을 0은 1로, 1은 0으로 변환(flip)



modification group	OP code	rd	rs1	rs2	rs3	0	function code
SET	110110	o	o	o	o		000001
CLEAR	110110	o	o	o	o		000010
TEST	110110	o	o	o	o		000100
FLIP	110110	o	o	o	o		001000

그림 4. modification 명령어 셋 구조
Fig. 4. A structure of the modification instruction set

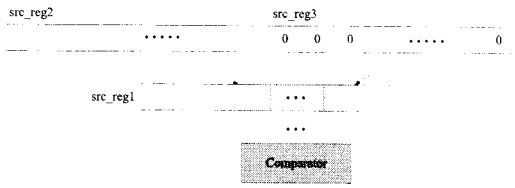


그림 5. TEST 명령어의 동작원리
Fig. 5. The operation of TEST instructions

기능 코드(function code)에 해당한다.

src_reg2는 특정 비트 영역의 시작 비트를 나타내며, src_reg3은 특정 비트 영역의 끝나는 비트를 나타내게 된다. 이 값들은 src_reg1의 값의 특정 비트 영역을 나타내며, 이는 기능 코드에 따라 연산결과가 MUX에서 선택되어 dest_reg에 값이 저장된다. 그림 5는 변형 명령어 셋 그룹 중에서 TEST 명령어의 동작 원리를 도식화 한 것이다.

Ⅳ. 제안하는 네트워크 ASIP

4.1 범용 ASIP

기본적인 명령어를 처리할 수 있는 범용 ASIP은 32비트 MIPS-DLX RISC(Reduced Instruction Set Computer) 아키텍처를 기반으로 그림 6과 같이 설계하였다. 또한 명령어 메모리(instruction memory)와 데이터 메모리(data memory)가 분리된 하버드 아키텍처(harvard architecture)구조로 설계하였으며 5단 파이프라인 구조이다(IF(Instruction Fetch), ID(Instruction Decode), EX(Execution), MEM(Memory access) and WB(Write-back)). EX 단계의 기능 유닛(function unit)은 ALU0, ALU1, SHFT, MULT로 구성되어 있으며 이를 통해 기본적인 산술 및 논리 명령어, 분기 명령어, Load & Store 명령어,

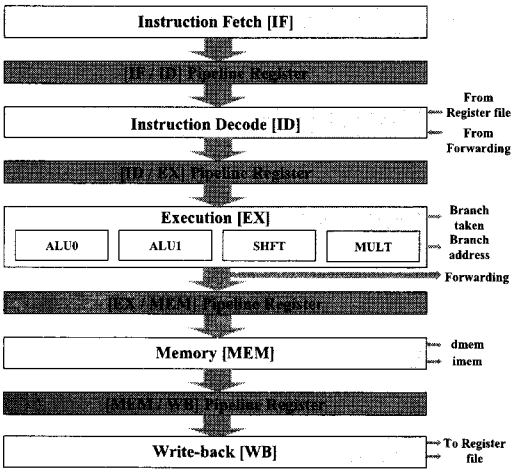


그림 6. 설계한 범용 ASIP
Fig. 6. The designed general-purpose ASIP

기타 명령어 등을 수행할 수 있다.

4.2 네트워크 ASIP

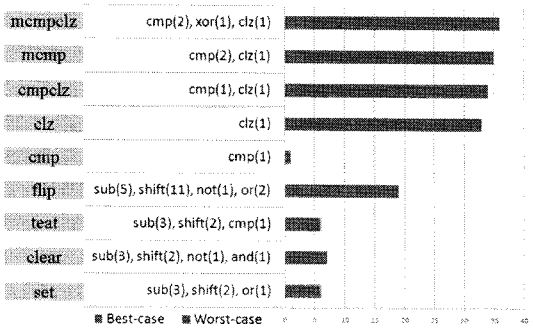
범용 ASIP에 패킷 프로세싱을 위해 NX 명령어를 추가하고, 이를 한 싸이클에 수행하기 위해 NX 기능 유닛을 EX 단에 추가한 네트워크 ASIP을 설계 하였다.

표 3은 NX 명령어를 일반적인 명령어로써 수행했을 경우 각 명령어 별로 수행되는 범용 명령어 및 총 수행 싸이클을 나타낸 것이다. 단, CMP와 CLZ 명령어와 같은 경우에는 일반적인 명령어가 아니며, CLZ의 경우 MSB로부터 한 비트씩 0과 비교하여 1이 나올 때까지의 0의 개수를 세는 명령어이기 때문에 최악의 조건(worst case)에선 32싸이클이 걸리게 된다.

제안한 NX 명령어를 지원하기 위해 LISA(Language

표 3. 일반적인 명령어로 NX 명령어를 수행하였을 경우의 수행 싸이클 수
Table 3. A number of cycles of NX instructions

NX 명령어	명령어(Cycle)	Total Cycle (Best case)
mcmpclz	cmp(2), xor(1), clz(1)	4
mcmp	cmp(2), xor(1)	3
cmpclz	cmp(1), clz(1)	2
clz	clz(1)	1
cmp	cmp(1)	1
flip	sub(5), shift(11), not(1), or(2)	19
test	sub(3), shift(2), cmp(1)	6
clear	sub(3), shift(2), not(1), and(1)	7
set	sub(3), shift(2), or(1)	6



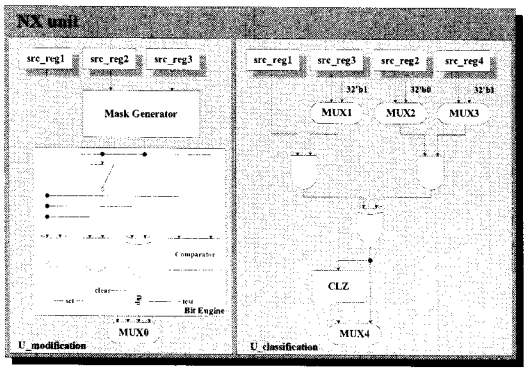


그림 7. U_classification과 U_modification
Fig. 7. U_classification & U_modification

for Instruction Set Architecture)를 사용하여 명령어 셋을 추가하였다. NX 명령어 셋이 추가된 네트워크 ASIP은 CoWare 디자인 컴파일러의 HDL 생성기를 사용하여 Verilog HDL로 생성하였다. NX 명령어는 파이프라인 5단계 중 ID 단계에서 opcode를 통해 디코드 되며, opcode가 '110110'이면 EX 단계의 U_classification 유닛으로, opcode가 '110101'이면 U_modification 유닛으로 데이터가 입력되어 기능 코드에 따라 각기 다른 처리 과정을 통해 결과값이 출력된다. 하지만 CoWare 디자인 컴파일러의 HDL 생성기를 통해 생성된 Verilog HDL 모듈은 최적화가 되지 않은 상태이기 때문에 면적의 효율 및 실행 속도가 떨어진다. 따라서 두 개의 기능 유닛의 성능향상을 위해 HDL 생성기를 통해 생성된 Verilog HDL을 새로이 작성(hand-written)한 Verilog HDL로 치환하여 최적화 하였다. 그림 7은 그림 6의 EX단계에 추가된 최적화된 U_classification과 U_modification의 기능 유닛을 도식한 것이다.

분류 명령어 그룹은 U_classification이라는 기능 유닛에서 처리된다. 기능 코드의 [2:1]에 해당하는 비트는 MUX2의 컨트롤 신호로 입력된다. 만약 [2:1]에 해당하는 비트가 모두 0이면 CLZ 관련 명령어를 의미하므로, MUX2는 0으로 이루어진 32bit 값을 출력한다. 만약 [2:1]에 해당하는 기능 코드가 00이 아닌 경우에는 src_reg2의 데이터가 출력된다. 기능 코드의 [2] bit은 MUX1과 MUX3의 컨트롤 신호로 사용되어, 1일 경우 MCMP 명령어나 MCMPCLZ 명령어에 해당하므로 각각 src_reg3 과 src_reg4의 값을 출력하며, 0일 경우 각각 32'b1 을 출력한다. 기능 코드의 [0] bit은 MUX4의 컨트롤 신호로 사용되어서, 1일 경우 CLZ 블록을 거친 값이 출력되며, 0일 경우 CLZ 블록을 거치지 않은 값이 출력된다. CLZ 블록은 Leading One Detector[5]로 설계하여 실행시간을 단축하였다.

변형 명령어 그룹은 U_modification이라는 기능 유닛에서 처리된다. src_reg2와 src_reg3이 가르치는 값은 Mask Generator에 입력되어 특정 비트 영역을 1, 나머지 부분은 0으로 된 32bit의 값이 출력된다. Mask Generator에서 출력된 값과 src_reg1의 값은 Bit Engine에 입력되어 변형 명령어에 해당하는 4가지 연산이 이루진다. 4가지 연산의 결과값은 MUX0에서 기능 코드의 하위 2비트를 컨트롤 신호로 받아 EX/MEM Pipeline Register로 전달된다. 이와 같은 최적화된 NX 유닛으로 원하는 대역폭 안에서 한 싸이클 동안 NX 명령어가 처리된다.

V. CKF

C레벨의 어플리케이션에 적용하기 위하여 컴파일러가 필요하게 된다. 하지만 범용 프로세서와 달리 ASIP은 일반적인 컴파일러로는 불충분하며 보다 특성화된 코드 생성과 최적화를 위해 새로운 컴파일러가 필요하게 된다. 이를 위해 확장 명령어 세트를 CKF(compiler known functions)로 지정하여 컴파일러를 생성하였다. 이는 일반적인 함수 호출을 하지 않고 CKF에 호출을 할당하여 로컬 함수 호출에 따른 오버헤드 없이 C레벨 매크로로서 사용되어 질 수 있다. 앞에서 설계된 NX 명령어와 이를 위한 기능 유닛을 상위 레벨의 어플리케이션에서 사용하기 위해선 컴파일러에 의한 연결이 필요하다.

C레벨에서 제안한 명령어에 해당하는 연산이 불가능한 것은 아니지만, 프로그래머 입장에선 사용이 용이하지 않으며 임베디드 시스템에서 중요시 되는 코드 사이즈가 상당히 커진다는 단점이 있다. 또한 더 큰 문제는 이들은 각각의 어셈블리로 표현될 수 밖에 없어 명령어 수가 증가한다. 예를 들어 LPM (longest prefix match) 연산을 C언어로 구현하려면

아래와 같이 표현 되어 질 수 있다.

```
unsigned int lpm(){
data = (A&B)^(C&D);
for(i=32; i>0; I--){
        tmp = (data >> 1);
        if(tmp == 0) count++;
        else break;
        } return count;
}
```

그러나 이러한 연산은 31번의 쉬프트 연산과 더불어 수많은 산술연산이 이루어지므로 프로세서 입장에서 큰 부담이 되며 성능 저하에 큰 영향을 미치게 된다. 특히 lookup 테이블의 검색 시 엔트리 수가 많은 경우 이를 검색하기 위하여 많은 시간이 소요되어지나, CKF를 통하여 아래와 같이 정의하고 명령어 셋에 할당해 놓으면 검색 작업을 가속화 할 수 있다.

unsigned int lpm(unsigned int A, unsigned int B, unsigned int C, unsigned int D);

A와 C는 rs1과 rs3에 해당하는 값으로 비교되어야 할 정보이고, B와 D는 rs2와 rs4에 해당되는 값으로 마스크이다. C함수에서 LPM은 'mcmpclz' 명령어로 할당되며, 마스킹한 후 두 개의 데이터를 비교하여 얼마나 일치하는지 카운트하여 결과 레지스터에 전달하는 구조이다. 또한, 변형 명령어 그룹의 set, clear, flip, test를 위하여 아래와 같이 정의하였다.

unsigned int prset(unsigned int A, unsigned int B, unsigned int C);
unsigned int prclear(unsigned int A, unsigned int B, unsigned int C);
unsigned int prflip(unsigned int A, unsigned int B, unsigned int C);
bool prtest(unsigned int A, unsigned int B, unsigned int C);

이는 변형 명령어 그룹의 set, clear, flip, test에 연결되며 A는 rs1에 해당하는 값으로 수정할 레지스터이며 B는 rs2에 해당하며 시작 지점이며 C는 rs3에 해당하며 종점에 해당한다. prset, prclear, prflip 은 패킷 일부를 수정하지만 prtest는 비교한 결과가 같은지 틀린지를 true, false 로 전달한다.

VI. 실험 및 결과

6.1 설계한 각 ASIP의 합성결과

설계한 총 3가지의 ASIP 모델의 합성결과이다. 첫 번째 모델은 NX 기능 유닛이 포함되지 않은 범용 ASIP이다. 두 번째 모델은 NX 기능 유닛이 포함되어 있으며 NX 기능 유닛을 최적화하지 않은 모델이다. 마지막 모델은 두 번째 모델을 최적화한 최종적으로 제안하는 모델이다. 각각의 ASIP 모델은 기본적으로 LISA를 통하여 모델링 되었으며 프로세서 디자이너(Processor Designer)를 통하여 시뮬레이터, 링커, C-컴파일러, 어셈블러, 디버거를 생성하였다.

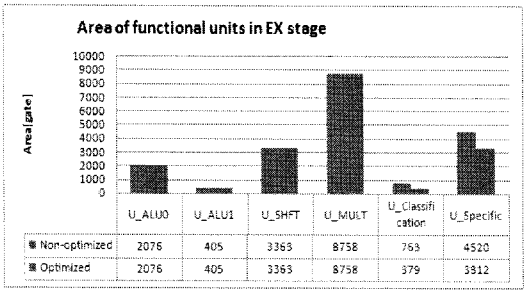
표 4는 시납시스 디자인 컴파일러(Synopsys Design Compiler)를 사용하여 동부 0.18 CMOS 라이브러리로 합성하였을 때 NAND(2x1) 게이트(9.9792) 기준의 전체 면적이다. 제약 조건(constrain)은 worst case로 세팅하여 합성하였다. 최적화하지 않은 두 번째 ASIP 모델의 경우 NX 기능 유닛을 추가하였으므로 범용 ASIP에 비하여 7.5% 면적이 증가하였으며, 최적화한 세 번째 ASIP 모델의 경우 범용 ASIP에 비해 4.7% 면적이 증가하였다.

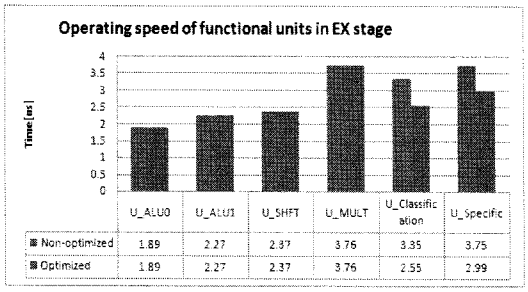
표 5는 EX 단계의 각 기능 유닛들의 면적과 동작 속도를 나타낸다. 5단 파이프라인에서 EX 단계가 다른 단계에 비해 가장 처리속도가 느리므로, EX 단계의 기능 유닛 중 처리속도가 가장 느린 유닛에 의해 전체 ASIP의 동작 주파수가 결정된다. 합성 결과 곱셈(Multiply) 연산을 담당하는 U_MULT가 처리속도가 가장 느리므로, U_MULT의 처리속도와 ID/EX 파이프라인 레지스터, EX/MEM 파이프라인 레지스터에서 소요되는 시간을 각각 합하면 전체 ASIP의 동작 주파수가 결정된다. 합성 결과 동작 주파수는 약 156.5Mhz (6.38ns)로 결정되었다. 동작 주파수 측면에서는 U_classification과 U_modification 기능 유닛을 최적화할 필요가 없지만, 곱셈 연산을 필요로 하지 않은 라우터에 장착된 경우 U_classification과 U_modification 기능 유닛의 수행 속도에 의해 전체 동작주파수를 결정되므로 두 기능 유닛을 최적

표 4. 각 모델의 총 면적 (Eq. NAND2x1)
Table 4. Total area of each models

	Basic	Non-optimized	Optimized
Total area[gata]	80840	86922	84670
Normalization	1	1.075	1.047

표 5. EX 단계의 각 유닛별 면적 및 수행 속도
Table 5. The area and the operation speed of each units on EX stage





화하였다. 최적화한 결과 동작 지연시간 측면에서는 각각 23.9%, 20.3% 성능향상이 있었으며, 면적 측면에서도 약 50.3%, 26.7%의 면적 감소를 보였다.

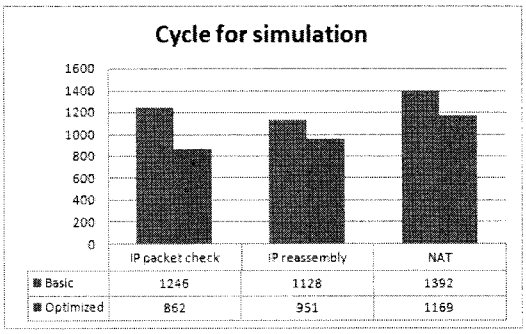
6.2 성능비교

설계된 네트워크 ASIP을 위한 전용 컴파일러를 만들기 위해 Coware사의 디자인 컴파일러를 이용하여 lcc 컴파일러를 생성하였으며, 추가된 기능 유닛과 명령어 그룹을 연결하기 위해 CKF를 정의한 후 컴파일러에 적용하였다. 이는 프로그래머 입장에서 하나의 C 매크로로 인식되어 하나의 명령어로 네트워크 패킷 연산이 가능하며 성능 면에서는 한 사이클 내에 처리가 가능하게 된다.

시뮬레이션을 위하여 EEMBC 테스트벤치의 IP packetcheck, IP reassembly, NAT, OSPFv2, QOS, Rounte, TCP 7개의 테스트 항목 중 분류(Classification)와 변형(Modification)이 빈번이 이루어지는 IP packetcheck와 IP reassembly, 그리고 NAT에 대하여 CKF를 적용 하였을 때와 CKF를 적용하지 않고 연산하였을 때를 C레벨에서 적용하였다. 이를 lcc를 이용하여 컴파일 한 후 디버거에서 전체 사이클 수와 동작 시간을 비교하였다.

IP packet check에선 데이터그램의 길이는 MTU (Maximum Transfer Unit)인 1500byte로 고정하고 버퍼 사이즈는 1Mbit로 하였으며 360개의 데이터그램에 대한 파싱(parsing)과 분류(classification)을 거

표 6. 실행 싸이클 비교
Table 6. The comparison of each execution cycle



쳐 egress하는데 걸리는 사이클 수를 비교하였다.

IP reassembly에선 최대 패킷 사이즈를 600byte로 분할하여 입력되어지는 패킷에 대하여 reassembly를 실행한다. 데이터그램 200개에 대한 실행 사이클 수를 비교하였다.

NAT에선 10.1.1.1로 입력되어지는 패킷에 대하여 룩업(lookup) 테이블을 검색하여 192.1.1.1로 변환하여 하위 네트워크로 전송하고 192.1.1.1로 들어온 패킷에 대하여는 lookup 테이블 검색 후 10.1.1.1로 변경하여 egress하는 작업을 실행한다. 100byte의 데이터그램에 대해 1000번 반복 수행 시 실행 사이클 수를 비교하였다.

위의 어플리케이션에서 알 수 있듯이 CKF 적용 시 C레벨에서의 코드 사이즈 감소뿐만 아니라 어셈블러 레벨에서의 전체 명령어 수를 감소시킬 수 있다. 이는 성능 향상 및 스위칭 동작을 줄여 소비전력 감소의 효과가 있다.

주어진 어플리케이션을 디버거에서 시뮬레이션 한 결과 표 6과 같이 IP packetcheck 실행 시 29.8%, IP reassembly 실행 시 15.7%, NAT의 경우 16.1% 개선됨을 확인하였다.

이를 종합해 보면, 네트워크를 위한 패킷 처리 시 제안하는 분류 명령어 그룹과 변형 명령어 그룹을 통해 약 15~30% 가속시킬 수 있다.

Ⅶ. 결 론

본 논문에선 LISA를 통하여 시뮬레이션 모델로 설계한 새로운 네트워크 ASIP을 제안한다. 제안한 네트워크 ASIP에 추가된 NX 명령어는 U_classification과 U_modification 기능 유닛에서 한 싸이클 안에 처리되며, 동부 0.18 CMOS 라이브러리로 합성한 결과 동작 지연시간은 각각 2.55ns, 2.99ns, 면적은

각각 379.666um^2, 3312.346um^{2}로 합성되었다. 또한 제안한 명령어를 C 레벨 네트워크 어플리케이션에서 테스트하기 위하여 CKF를 이용하여 매크로 함수로 할당하여 전용 컴파일러 lcc를 생성하고 이를 입력으로 한 실행시간 시뮬레이션에서 성능 향상을 확인하였다. 입력한 어플리케이션은 EEMBC 테스트 벤치의 IP packet check, IP reassembly, NAT 3 항목에 대한 시뮬레이션을 수행하였으며 수행결과 실행 사이클에서 각각 29.8%, 15.7%, 16.1% 개선됨을 확인하였다.

하나의 NX 명령어는 최대 32개의 일반적인 명령어로써 수행할 수 있는 연산을 하나의 명령어로 한 싸이클 안에 처리하므로 NX 명령어를 많이 사용할수록 전체적인 처리 속도를 높일 수 있다. 또한 하나의 패킷을 처리하기 위해서 범용 ASIP의 일반적인 명령어를 이용할 경우 여러 개의 명령어들을 통하여 처리되지만, 본 논문에서 제안한 네트워크 ASIP의 NX 명령어를 사용하면 상대적으로 적은 수의 명령어들을 통하여 처리할 수 있다. 따라서 패킷 프로세싱의 코드 크기를 줄일 수 있으며, 이로 인해 IRAM(Instruction RAM)을 효율적으로 사용할 수 있다.

IP 패킷을 기반으로 하는 트래픽의 양적 증가로 인해 처리해야 할 패킷이 증가하고 있다. 또한 멀티미디어 속성의 다양한 어플리케이션이 새롭게 개발되고 발전함에 따라 사용하는 프로토콜이 다양해지고 복잡해지고 있기 때문에 분석해야 한 패킷 헤더 필드가 커져서 라우터의 프로세싱 속도가 더욱 중요시 되고 있다. 따라서 최적화된 NX 기능 유닛은 전체적으로 약 4.7%의 면적 증가가 있지만, 성능면에서 약 15~30%의 처리속도향상이 있기 때문에 라우터에 내장 될 하드웨어 가속기로 적합하다 할 수 있다.

참 고 문 헌

[1] R. M. Hinden, "IP Next Generation Overview," *Commun. ACM*, vol. 39, no. 6, 1996, pp. 61 - 71.

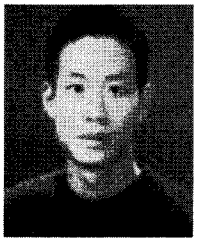
[2] Scott Weber and Liang Cheng, "A Survey of Anycast in IPv6 Networks," *Communications Magazine*, IEEE, Vol.42, Jan 2004, pp. 127-132.

[3] Ha-young Jeong, Hyoung-pyo Lee, and Yong-surk Lee, "A Low-cost Multimedia ASIP Architecture for .264/AVC," *The 22nd International Technical Conference on Circuits/Systems, Computers and Communications(ITC-CSCC 2007)*, Vol. 2, July 2007, pp. 777-778.

[4] Andreas Hoffmann, Tim Kogel, Achim Nohl, etc. "A Novel Methodology for the Design of Application-modification Instruction-Set Processors (ASIPs) Using a Machine Description Language," *IEEE Transactions on Computer-AIDED design of integrated circuits and systems*, Vol. 20, No. 11, November 2001.

[5] H. Suzuki, et. Al, "Leading-Zero Anticipatory Logic for High-Speed Floating Point Addition," *IEEE Journal of Solid-State Curcuits*, vol. 31, No. 8, August 1996.

정 원 영 (Won-young Chung) 정회원



2005년 8월 : 연세대학교 전기전자공학과 졸업

2005년 9월 ~ 현재 연세대학교 전기전자공학과 석박통합과정

<관심분야> 네트워크 프로세서, 컴퓨터 아키텍처

이 정 희 (Jung-hee Lee) 정회원

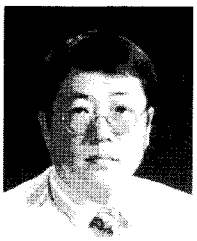
1984년 2월 경북대학교 전자공학과 학사

1990년 2월 경북대학교 전자공학과 대학원 석사

1984년 3월 ~ 현재 한국전자통신연구원 광대역통합망 연구원

<관심분야> 네트워크 자원관리, 인터넷 QoS

이 용 석 (Yong-Surk Lee) 정회원



1973년 2월 연세대학교 전기공학과 학사

1977년 2월 University of Michigan, Ann Arbor 석사

1981년 2월 University of Michigan, Ann Arbor 박사

1993년 ~ 현재 연세대학교 전기전자공학과 교수

<관심분야> 마이크로프로세서, 네트워크 프로세서, 암호화 프로세서, SoC