논문 2008-45SD-9-8

저비용 내장형 멀티미디어 프로세서를 위한 분할 레지스터 접근 구조

(A Partial Access Mechanism on a Register for Low-cost Embedded Multimedia ASIP)

조 민 영*, 정 하 영*, 이 용 석**

(Minyoung Joe, Hayoung Jeong, and Yong Surk Lee)

요 약

본 논문은 저비용 내장형 멀티미디어 프로세서를 위한 레지스터 분할 접근 구조를 제안한다. 저비용 내장형 시스템에서 SIMD 명령어 지원은 SIMD 지원 레지스터 파일과 실행유닛들의 추가에 따른 비용의 증가 때문에 적용이 어렵다. 제안한 구조는 하드웨어의 부담을 최소화하면서 SIMD 연산 수행을 지원하여 전체적인 성능을 향상 시킬 수 있는 구조다. ASIP을 설계하여 제안한 구조를 적용시켰으며 DSP 벤치마크에서 명령어 적용에 따른 실행 사이클의 변화를 비교하였다. 설계한 ASIP을 TSMC 0.25㎛ 공정으로 합성하여 제안한 구조 적용에 따른 면적 증가 및 전체적인 성능 향상을 분석하였다. 실험 결과 제안한 구조는 성능은 약 38% 향상되었고, 면적은 13.4% 증가하였다.

Abstract

In this paper, we propose a partial access mechanism for low cost multimedia processors. Due to the cost increase of adding the SIMD register files and the execution blocks, we experience difficulties applying the SIMD instructions to low cost multimedia embedded processors. The proposed mechanism has the advantages of decreasing the cost burden of the additional hardware and enhancing total performance of the SIMD operation. We designed the ASIP in which the mechanism is applied and compared the latency of the SIMD operation regarding the use of instruction sets in the DSP benchmark. Then, we analyzed the total performance enhancement and the reduction in area burden by synthesizing the ASIP using 0.25um TSMC CMOS technology. As a result, there are approximately a 38% of performance increase and a 13.4% of area increase according to the proposed mechanism simulation.

Keywords: 임베디드 프로세서, ASIP, SIMD, 선택적 접근

I. 서 론

최근 휴대용 멀티미디어 기기들의 사용이 증가하면서 고성능 내장형 프로세서에 대한 필요성이 높아지고 있다. 그러나 ASIC(Application Specific Integrated Circuit)은 그 복잡도에 따라 개발기간의 장기화와 설계와 변경의 어려움 때문에 디지털 기기들의 급격한 발달 속도를 따라가는데 어려움이 있다. 범용 프로세서나 DSP(Digital Signal Processor)는 그 응용범위가 넓은 장점이 있지만 ASIC보다 상대적으로 느리기 때문에 특정 어플리케이션에서의 성능을 만족시키지 못하는 경우가 있다. 이에 반해 어플리케이션 전용의 명령어 셋과 데이터 패스를 가진 ASIP(Application Specific Instruction set Processor)은 다양한 응용과 특정 알고리즘을 구현하는데 적합하다는 장점이 있어 멀티미디어 기기들에 널리 쓰이고 있다.

* 학생회원, ** 평생회원, 연세대학교 전기전자공학과
(Department of Electrical and Electronic Engineering, Yonsei University)
접수일자: 2008년6월3일, 수정완료일: 2008년9월1일

멀티미디어 어플리케이션에서는 데이터들의 병렬성 때문에 SIMD(Single Instruction Multiple Data) 명령어가 성능을 가속하기 위해 자주 사용되는데 Intel Pentium, AMD Athlon, SUN SPARC와 같은 상용화된 프로세서들은 이미 각각 MMX[1], SSE[2], 3DNow![3], VIS[4] 등과 같은 SIMD 명령어 셋을 갖추고 있다. 그러나 SIMD 명령어 셋을 구현하기 위해서는 128bit SIMD 연산기나 128bit SIMD 레지스터 파일 같은 별도의 하드웨어 자원들을 필요로 하기 때문에 하드웨어의 추가 비용이 존재한다.

최근 임베디드 프로세서의 발달에 따라 임베디드 프로세서에서도 SIMD 지원을 하기 시작하였는데, ARM v6 명령어 셋은 SIMD 명령어가 포함되어 있으며 NEON이라는 별도의 SIMD 확장 명령어 셋 유닛을 통하여 고성능 SIMD 연산을 지원하고 있다. 하지만 저비용의 임베디드 기기에서는 ARM과 같이 SIMD 지원을 위한 별도의 연산 유닛을 사용할 수 없다. 따라서 저비용 임베디드 기기에 적합한 SIMD 유닛의 연구가 필요하다.

임베디드 시스템에서 SIMD 연산 유닛 확장에 따른 부담을 최소화하고, 저비용 SIMD를 구현하기 위한 또 다른 방법으로는 소프트웨어 SIMD 방식이 있다.[5] 이 방식은 범용 레지스터 위에 소프트웨어적 방법으로 몇 개의 서브 워드(sub-word)들을 위치시켜 한 번에 연산함으로써 SIMD 연산을 지원하지 않는 하드웨어에서 SIMD 연산과 비슷한 동작과 성능을 이끌어 내고자 하는 방식을 말한다. 하지만 소프트웨어로 처리하기 위해서는 서브워드간의 데이터 독립성 때문에 서브워드마다 guard bit을 둬야 하므로 32bit 방식에서는 최대 3개의 8bit 서브워드 만을 처리할 수 있다. 만약 이러한 동작을 하드웨어로 지원이 가능하다면, 기존의 소프트웨어 SIMD 방식의 성능을 좀 더 개선할 수 있을 것이다. 하지만 소프트웨어 SIMD 방식과 같이 여러 서브워드를 하나의 SIMD 레지스터로 압축하기 위해서는 많은 명령어가 필요하고 기존의 명령어만으로는 한계가 있다. 기존의 레지스터는 워드(word)단위의 접근만이 가능하므로 하나의 워드 레지스터에 여러 개의 서브워드 데이타를 담기 위해서는 여러 번의 쉬프트(shift)와 비트 단위 논리합(or) 명령어 조합이 필요하다. 하지만 레지스터에 서브워드단위로 부분적인 접근이 가능하다면 묶기(pack), 풀기(unpack)를 위한 오버헤드를 줄일 수 있을 것이다. 따라서 이러한 레지스터의 일부만을 접근할 수 있는 프로세서 구조와 명령어를 추가한다면 프로세서 성능을 향상 시킬 수 있을 것이다.

본 논문은 저비용 멀티미디어 ASIP을 위한 분할 접근 구조을 제안한다. 논문의 나머지 부분은 다음과 같이 구성되어 있다. 본론 I 장에서는 멀티미디어 ASIP과 저비용 32bit SIMD 유닛의 구조를 살펴본 후 문제점을 설명한다. 본론 II장에서는 제안한 레지스터 분할 접근 구조을 설명한다. 실험에서는 제안한 레지스터 분할 접근 구조의 성능 시뮬레이션 결과를 보이고 하드웨어 모델에 따른 결과를 비교하고 분석한다. 마지막으로 본 논문의 결론을 맺는다.

II. 본 론

1. 멀티미디어 ASIP 구조

본 논문에서 사용한 멀티미디어 ASIP은 MIPS-DLX 파이프라인에 기반한 5단 scalar 파이프라인 구조를 가지고 있다. 또한 저비용 SIMD 연산을 구현하기 위하여 개선한 32bit SIMD 연산 유닛을 사용하였다.

그림 1은 개선한 저비용 SIMD 덧셈기의 구조를 보여주고 있다. 이 덧셈기는 SIMD 서브워드 크기 블록마다 게이트를 달아 각 덧셈 블록의 연산결과가 다른 블록으로 전달되는 것을 막을 수 있는 구조로 되어 있으며, 신호에 따라 연산의 단위를 8bit, 16bit, 32bit로 설정할 수 있다. Group CLA(Carry Lookahead Adder) 구조와 같이 고속 연산을 위해 블록별 연산 구조를 가

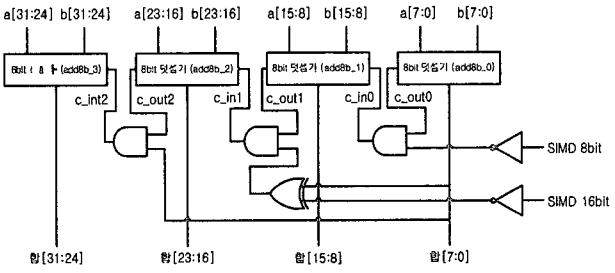

그림 1. 저비용 SIMD 덧셈기
Fig. 1. Low-cost SIMD adder implementation

표 1. 기본 SIMD 명령어
Table 1. Basic SIMD instruction.

Instruction	Description
sadd8 / sadd16	4쌍의 8bit 서브워드 간의 덧셈 연산 2쌍의 16bit 서브워드 간의 덧셈 연산
ssub8 / ssub16	4쌍의 8bit 서브워드의 뺄셈 연산 2쌍의 16bit 서브워드의 뺄셈 연산
smul16	2쌍의 16bit 서브워드 간의 곱셈 연산

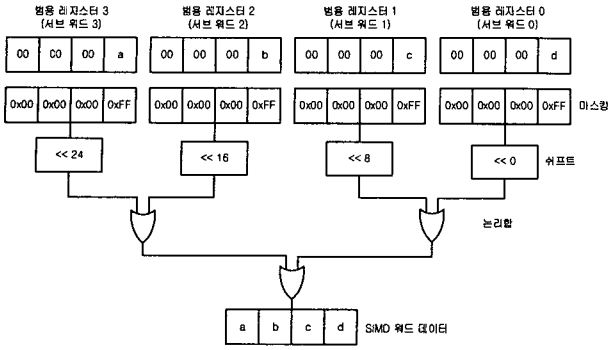

그림 2. 일반 명령어를 사용한 레지스터 packing
Fig. 2. Register packing with normal instruction.

진 덧셈기에서는 제안한 구조를 쉽게 적용할 수 있다. 이러한 구조는 서브워드간 상호 간섭 때문에 guard bit을 둬야 하는 소프트웨어 SIMD에 비해서 동일 클럭 사이클에서 더 많은 서브워드 데이타를 연산할 수 있게 한다.

표 1은 본 논문에서 추가한 SIMD 명령어를 보여준다. 덧셈과 뺄셈 같은 기능은 위에서 설명한 저비용 SIMD 덧셈기로 구현할 수 있고 두개의 16bit 곱셈기를 사용하여 동시에 2개의 16bit 곱셈 처리를 지원한다. 이러한 SIMD 산술 연산 명령어들을 사용하여 최소한의 비용으로 데이터 처리 속도를 높일 수 있다. 하지만 여러 서브워드들을 SIMD 레지스터에 묶을 때 추가적인 쉬프트와 논리합 연산이 필요하다. 그림 2에서 볼 수 있듯이 4개의 8bit 서브워드들이 하나의 SIMD 레지스터에 로드될 때, 3번의 쉬프트 연산과 3번의 논리합 연산이 사용된다. 만약 부호 있는 숫자를 다루게 된다면 여기에 3번의 추가 연산이 필요하다. 따라서 만약 SIMD 데이터를 준비하는 이러한 추가적인 연산들이 실제 SIMD 산술 연산에 비례하여 오버헤드가 크다면 소프트웨어 SIMD는 오히려 성능이 감소할 수 있다. 이러한 데이타 병렬처리화 오버헤드를 줄이기 위해서는 하드웨어와 컴파일러의 지원이 필수적이다.

2. 분할 접근 구조

대부분의 RISC 프로세서는 워드 단위로만 레지스터 파일에 접근 할 수 있다. 이런 구조적 한계 때문에 컴파일러에서도 SIMD 데이타에 접근할 때 오직 워드 단위로만 접근할 수 있다. SIMD 연산에서는 서브워드간 간섭이 없어야 하는데 일반적인 RISC 프로세서와 같이 워드단위로만 접근할 수 있다면 사칙연산들은 수행할 때 서브워드들끼리 자리올림(carry)/빌려옴(borrow)과 같은 상호작용을 할 수밖에 없다.

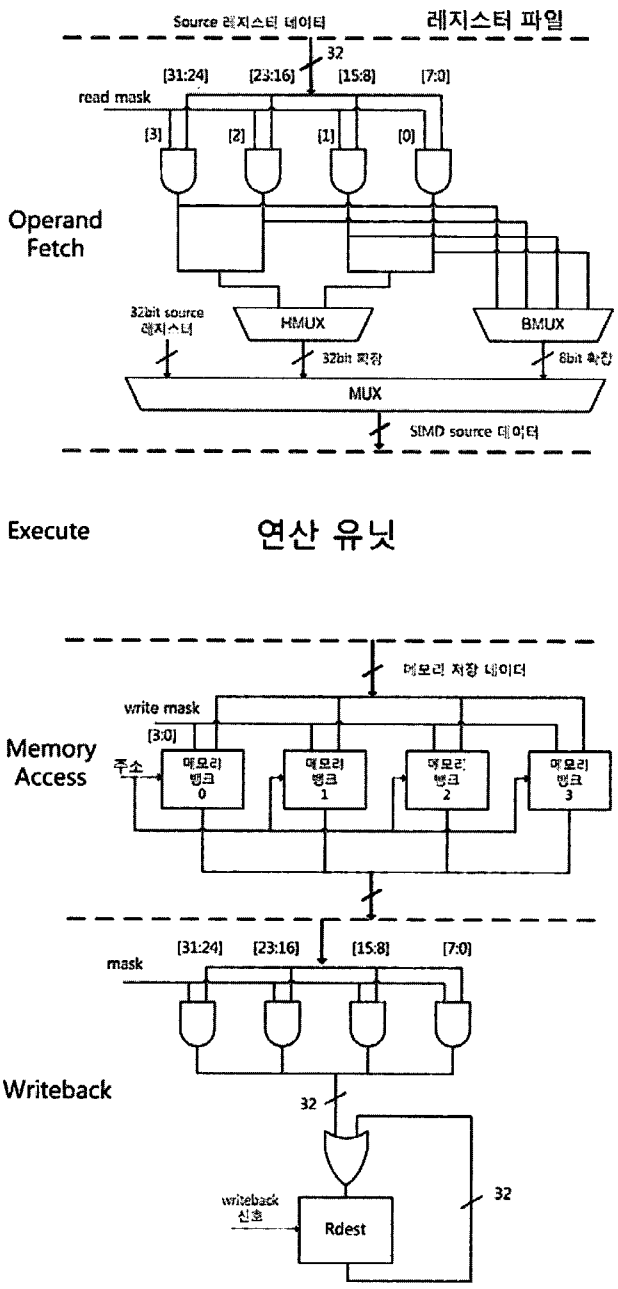

그림 3. 분할 레지스터 접근 구조
Fig. 3. Partial access mechanism.

그림 3은 각 서브워드간 이러한 간섭을 없애주기 위해 제안하는 블록 다이어그램이다. 제안한 구조처럼 각 레지스터에 서브워드 단위로 접근할 수 있다면, 소프트웨어 SIMD처럼 SIMD 데이타를 준비하기 위한 복잡한 연산을 줄일 수 있다.

제안한 분할 접근 구조에서는 덮개(mask)를 사용하여 byte 단위로 레지스터 일부만을 연산의 source 또는 destination 으로 사용할 수 있다. 이러한 구조를 사용하면 분할 저장 또한 구현이 가능하는데, 분할 저장은 SUN VIS에서 제안된 기능으로 SIMD 연산결과를

저장할 때 사용한다. 이 명령어를 사용함으로써 SIMD 연산 후 데이타를 저장할 때, 오버헤드를 감소시킬 수 있다.

분할 저장 명령어는 SIMD 데이터가 들어있는 레지스터에서 서브워드 단위로 데이터를 추출하여 읽고 메모리에 저장할 수 있어야 한다. 이를 구현하기 위해서는 우선 특정 서브워드를 추출할 수 있어야 하는데, 이를 위해 그림 3에서 제안한 블록 다이어그램을 살펴보면 레지스터에서 데이터를 추출하는 Fetch단과 결과를 다시 메모리에 저장하는 Memory-Access단을 수정하여 분할 저장을 가능하게 하였다.

Fetch단에서 추출된 서브워드는 각각 읽기나 쓰기를 독립적으로 할 수 있는 메모리 뱅크에 저장되는데, 이것은 멀티 뱅크 메모리이기 때문에 해당 메모리 뱅크를 가리키기 위한 레지스터 4bit mask(Write mask[3:0])가 추가적으로 필요하다.

제안한 구조에서 전체적인 SIMD 동작은 다음과 같다. 먼저 fetch단에서 source 레지스터는 바이트당 mask(rs_mask)가 할당되어 있다. 이렇게 선택된 필드는 MUX_array에 의해서 정해진 위치에 이동하는데 이렇게 해서 범용 레지스터에서 서브워드가 원하는 byte-offset에 위치 시킬 수 있다. 그리고 정렬된 데이터는 execute단과 memory단으로 보내져 SIMD 산술연산을 수행한다.

Writeback단에서 레지스터 파일로 수행결과를 기록해야 하는 데이터는 저장하기 위해서는 서브워드를 다시 정렬되어야 한다. 이런 점 때문에 rs_mask와 마찬가지로 저장할 서브워드 데이타의 위치와 사이즈를 rd_mask에 기록하고, MUX_array로 서브워드들을 재정렬하고 이 값은 레지스터 파일의 write 단자로 보내진다. 마지막으로 정렬된 서브워드는 rd_mask에 따라 각 레지스터 뱅크에 저장된다.

본 논문에서는 제안한 레지스터 접근 구조로 ASIP을 재구성하고 이를 지원하는 세 가지 명령어를 제안했다. 표 2는 제안한 분할 접근 명령어들을 보여준다. 이러한 명령어들은 분할 접근 구조를 이용하여 레지스터를 부분적으로 접근 가능하다.

분할 로드(Partial Load) : 이 명령어는 메모리에서 서브워드 데이터를 읽어와 그것을 레지스터로 이동시킨다. 일반적인 SIMD 구조에서 메모리 내에 서브워드들은 연속적으로 배치되어 있기 때문에 대부분 burst 모드로 읽지만, 만약 데이터가 정렬되어 있지 않다면 서브워드를 읽어오기 위해 메모리에 추가적으로 접근이 필요하다.[6] 그러나 이 명령어는 메모리에서 데이터를 읽어와 rd_mask에 따라 바로 타겟 레지스터의 byte-offset 만큼 위치한 곳에 서브워드를 정렬하기 때문에 별도의 작업이 필요없다.

분할 저장(Partial Store) : 이 명령어는 한 레지스터에서 rs_mask에 설정된 특정 서브워드를 메모리에 저장한다. 일반적으로 RISC 프로세서에서는 워드레지스터에서 서브워드를 추출하기 위해 masking과 쉬프트연산이 필요하지만, 제안한 구조에서는 한 사이클 안에 특정 서브워드를 추출하고 정렬할 수 있다.

분할 이동(Partial Move) : 이 명령어는 source 레지스터에서 특정 서브워드가 destination 레지스터로 저장될 때 byte-offset만큼 위치시킨 곳에 저장해준다. 이 명령어는 프로세서에서 SIMD 데이타를 다루는데 사용하게 된다.

III. 실 험

앞서 제안한 구조는 LISA 2.0을 사용하여 구현하였다. LISA 모델을 통하여 멀티미디어 ASIP을 설계한 다음 프로세서의 정확한 사이클 수준의(cycle-accurate)

표 2. 제안한 명령어 셋
Table 2. proposed instruction for partial access.

명령어	Syntax	동작 설명
partial load (plb, plh)	plb rd:mask, offset[rs]	메모리의 rs+offset 주소에서 읽어온 byte 또는 half 워드의 데이타를 레지스터 일부에 저장한다
partial store (psb, psh)	psb rd:mask, offset[rs]	레지스터의 일부를 메모리의 rs+offset 주소에 저장한다
partial move (pmovb, pmovh)	pmovb rd:mask, rs:mask	source 레지스터의 일부를 destination 레지스터의 일부분으로 이동한다.

시뮬레이터와 C 컴파일러, 어셈블러 등의 개발 환경을 생성하였다.[7~9] 제안한 구조의 성능 향상을 측정하기 위해 5가지 시뮬레이션을 하였는데, DSP프로그램에서 자주 사용되는 벡터 덧셈, 벡터 곱셈, 내적 연산, 컨볼루션 그리고 IDCT(inverse discrete cosine transform)를 선택하여 그 결과를 살펴보았다.

선택된 benchmark들은 C를 사용해 코딩되었으며 설계한 ASIP을 위하여 retargetable 컴파일러인 CoSy를 사용하였다. 제안된 명령어들은 CKF(compiler known function)를 사용해 구현하였다.

non-SIMD 명령어들을 사용한 모델(w/o SIMD inst)과 산술 명령어 셋만을 추가한 모델(SIMD core)과 제안한 명령어들을 추가한 모델을 사이클을 기준으로 비교하였고, 명령어들을 추가함으로 생기는 오버헤드를 측정하여 앞선 두 개의 비교군과 비교하였다.

그림 4는 시뮬레이션의 실행사이클 횟수를 보여준다. 시뮬레이션 결과 산술 SIMD 명령어 셋만을 사용한 모델(SIMD core)이 SIMD 명령어를 사용하지 않은 모델(w/o SIMD inst.)보다 사이클 횟수가 많게 나타났는데 이 이유는 산술 SIMD 연산을 사용하여 SIMD 연산이 가능하다는 점에서는 성능의 향상이 있었으나 일반 데이터를 SIMD 데이터로 만드는 묶기(pack). 풀기(unpack) 동작과정에서 오버헤드가 발생하여 전체적인 성능에서는 오히려 감소하였기 때문이다.

IDCT를 비롯하여, 전체적인 시뮬레이션 결과에서 제안한 구조의 성능이 높게 측정되었는데, IDCT의 경우 SIMD 연산을 위해 데이터를 준비하는 묶기, 풀기 동작 부분이 전체적인 기능 연산과정에서의 비중이 적기 때문에 제안한 구조의 성능이 더욱 높게 나올 수 있었다.

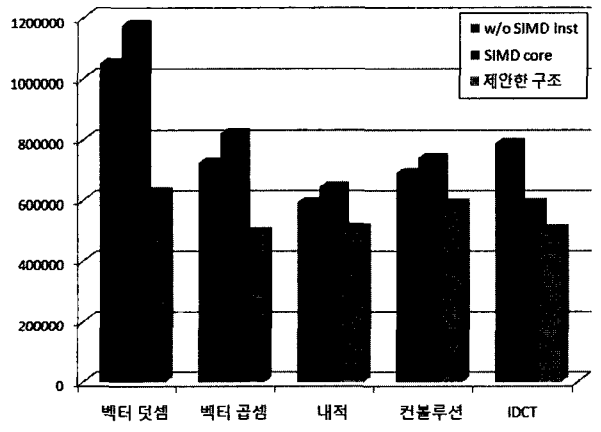

그림 4. Benchmarks 실행 사이클 비교
Fig. 4. Speedup of benchmarks on proposed architecture.

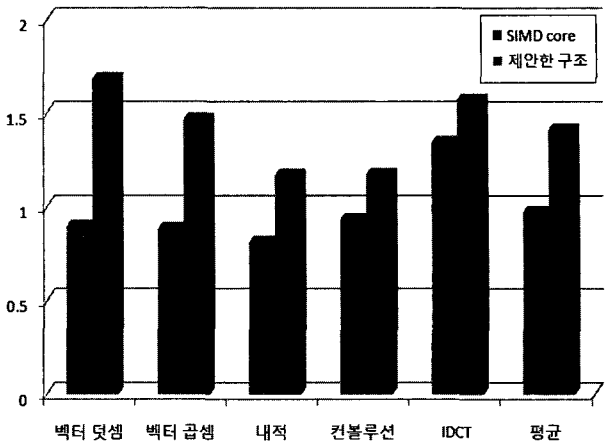

그림 5. Benchmarks 성능비 비교
Fig. 5. Speedup of benchmarks.

그림 5는 성능향상의 상대 비율을 나타낸다. 그림 5에 따르면 벡터 덧셈, 벡터 곱셈, 내적 연산, 컨볼루션에서 SIMD core 모델이 non-SIMD 모델보다 오히려 성능이 10% 낮게 나타났는데 이 이유는 데이타를 병렬데이타화하는 동작들에서 큰 오버헤드가 발생하여 전체적인 성능에서는 오히려 감소하였기 때문이다.

제안한 구조는 벡터 덧셈에서 최고 68%의 성능향상을 비롯하여 평균적으로 38%의 성능 향상을 보였고 시뮬레이션 결과에 기반하여, 제안한 구조가 SIMD 레지스터 묶기, 풀기 연산의 개선으로 성능이 향상되었음을 확인할 수 있다.

표 3. 합성 결과 (면적)
Table 3. Hardware area of three models

	면적(NAND2)	상대 비교
w/o SIMD	50902.24	0.88
w SIMD	57572.24	1.00
Proposed	65344.24	1.33

표 4. 합성 결과 (동작 주파수)
Table 4. Maximum clock frequency of three models.

	주파수 (Mhz)	상대 비교
w/o SIMD	132.34	1.01
w SIMD	131.59	1.00
Proposed	126.65	0.96

기준 ASIP과 제안한 명령어 셋을 적용한 ASIP은 모두 LISA 언어를 사용하여 설계해서, CoWare 사의 Processor Designer 를 이용하여 합성 가능한 HDL 모델을 생성하였다. 표 3과 4는 생성된 HDL을 TSMC 0.25㎛ 공정으로 합성하여 면적 및 동작 주파수를 비교한 결과를 나타내었다.

합성 결과 제안한 구조는 13.4%의 면적 증가와 3.8%의 동작 속도의 감소가 있어, 기준 프로세서의 동작속도에서 속도의 96.2%라는 속도 감소가 있었다. 하지만 이러한 속도 감소를 감안하더라도 제안한 구조에서 SIMD 연산에 따른 성능향상은 기준 ASIP에 비해 전체적으로 36%의 성능이 향상되었다.

Ⅳ. 결 론

기존의 저비용 멀티미디어 ASIP을 위한 소프트웨어 SIMD 방식은 하드웨어 확장없이 범용 레지스터를 SIMD 레지스터로 사용할 수 있다는 장점이 있다. 그러나 기존의 소프트웨어 SIMD 방식의 가장 큰 문제는 데이타 병렬처리화 오버헤드로 인한 큰 성능 하락이 있다는 것이다. 본 논문에서는 저비용 멀티미디어 ASIP을 위한 분할 레지스터 접근 구조를 제안한다. 이 방법은 레지스터의 일부만을 read/write할 수 있기 때문에 하나의 범용 레지스터를 SIMD 레지스터처럼 사용할 수 있게 한다. 본 논문에서 제안한 분할 접근 구조를 사용하면 데이타 병렬처리화 오버헤드를 감소시키면서 SIMD 유닛의 성능을 증가시킬 수 있다. 따라서 별도의 SIMD 실행 유닛을 사용하지 않기 때문에 하드웨어 구현 비용이 줄일 수 있으므로, 임베디드 기기에서 멀티미디어 응용 프로그램을 지원할 경우 유용한 방법이라고 할 수 있다.

참 고 문 헌

[1] A. Peleg, U. Weiser, I. Center, and I. Haifa, "MMX technology extension to the Intel architecture," Micro, IEEE, vol.16, no. 4, pp.42-50, 1996.

[2] S. Raman, V. Pentkovski, and J. Keshava, "Implementing streaming SIMD extensions on the Perntium III processor," Micro, IEEE, vol. 20, no. 4, pp.47-57, 2000.

[3] S. Oberman, G. Faver, F. Weber, A. Inc, and C. Sunnyvale, "AMD 3DNow! technology: architecture and implementations," Micro, IEEE, vol.19, no.2, pp.37-48, 1999.

[4] M.Tremblay,J.Narayanan,andV.He, "VIS speeds new media processing," Micro, IEEE, vol.16, no.4, pp.10-20, 1996.

[5] S. Kraemer, R. Leupers, G. Ascheid, and H. Meyr, "Soft SIMD-Exploiting Subword Parallelism Using Source Code Transformations," Design Automation and Test in Europe Conference and Exhibition, pp.1-6, 2007.

[6] Zhou, E. Q. Li and Y.-K. Chen, "Implementation of H.264 Decoder on General-Purpose Processors with Media Instructions", in Proc. of SPIE Conf. on Image and Video Communications and Processing, vol. 5022, pp.224-235, Jan. 2003.

[7] A. Hoffmann, H. Meyr, andR. Leupers, "Architecture Exploration for Embedded Processors with LISA", Springer, 2002.

[8] A. Hoffmann, "A Novel Methodology for the Design of Application Specific Integrated Processors (ASIP) Using a Machine Description Language," IEEE Trans. on Computer Aided Design, Month 2001

[9] A. Hoffmann, "A Methodology for the Design of Application Specific Instruction Set Processors (ASIP) Using the Machine Description Language LISA," International Conference on Computer-Aided Design, pp.625-630, San Jose, CA, USA, Nov. 2001

저 자 소 개

조 민 영(학생회원)
2006년 연세대학교 전기전자 공학과 학사 졸업.
2006년~현재 연세대학교 전기전자공학과 석사과정.
<주관심분야 : 마이크로 프로세서, SoC >

정 하 영(학생회원)
2003년 중앙대학교 전기공학부 학사 졸업.
2005년 연세대학교 전기전자 공학과 석사 졸업.
2005년~현재 연세대학교 전기전자공학과 박사과정
<주관심분야 : 마이크로 프로세서, ASIC, SoC>

이 용 석(평생회원)
1973년 연세대학교 전자공학과 학사 졸업.
1977년 University of Michigan Electrical Engineering 석사 졸업.
1981년 University of Michigan Electrical Engineering 박사 졸업.
1993년~현재 연세대학교 전기전자공학과 교수.
<주관심분야 : 마이크로프로세서 설계, VLSI 설계, DSP 프로세서 설계, 고성능 연산기 설계>