범용 멀티미디어 프로세서 구조 개발에 관한 연구

오명훈, 박성모
전남대학교 컴퓨터공학과
E-mail : mhoh@orion.chonnam.ac.kr

A Study on the Development of General-Purpose Multimedia Processor Architecture

Myeong-Hoon Oh, Seong-Mo Park
Dept. of Computer Engineering, Chonnam National Univ.

요약

멀티미디어 데이터를 아날로그 방식보다는 디지털 방식으로 처리하게 되면 여러 면에서 이득을 볼 수 있다. 멀티미디어 데이터를 디지털 방식으로 처리하는 방법 중 범용프로세서에서 멀티미디어 명령어에 의해 처리하게 되면 flexibility를 증가시키며 효율적으로 프로그래밍 할 수 있다.

본 논문에서는 범용 프로세서 안에서 멀티미디어 데이터를 효율적으로 처리할 수 있는 명령어 집합 구조와 이를 수행할 수 있는 프로세서의 구조를 제안하고 이를 HDL(Hardware Description Language)로 동작 레벨에서 기술하고 시뮬레이션 하였다.

제안된 멀티미디어 명령어는 특성에 따라 8개의 그룹에 총 55개의 명령어로 구성되며 64비트 데이터 안에서 각각 8비트의 8바이트, 16비트의 4하프워드, 32비트의 2워드의 부워드(subword) 데이터들을 병렬 처리한다. 모델링된 프로세서는 오픈아키텍처(Open Architecture)인 SPARC V.9의 정수연산장치(Integer Unit)에 기반을 두었으며 하바드 구조를 지닌 5단 파이프라인 RISC 형태이다.

1. 서론

과거에는 멀티미디어 정보들이 아날로그방식으로 처리되어왔다. 그러나 디지털 기술의 발달로 멀티미디어 정보들을 디지털 방식으로 처리하여 가격, 성능, 신뢰도 등 여러 면에서 이점이 생긴다. 이러한 오디오, 비디오, 3D 그래픽 등의 멀티미디어 정보를 디지털 방식으로 처리했던 초기의 방법은 데이터와 알고리즘에 따라 각각의 특성화된 특별목적의 칩과 메모리를 사용하는 것이었다.[1]

본 논문은 정보통신부의 정보통신분야 우수학교 지원사업에 의하여 수행된 결과임.

보다 발전되어 한가지 이상의 알고리즘에 대하여 처리를 가능케 하는 즉, 프로그램 가능한 DSP(Digital Signal Processor)가 등장하였다. DSP가 한 칩에서 여러 알고리즘을 수행할 수 있지만 역시 다양한 멀티미디어 데이터를 처리하기 위해서는 몇개의 DSP들이 필요하게 되고 각 DSP들의 상세한 구조를 알아야 프로그램이 가능하였다. 최근에는 한 개의 DSP에서 메모리와 프로세스 자원을 공유하여 여러 멀티미디어 데이터를 처리하는 새로운 형태의 미디어 코프로세서와 미디어 프로세서가 등장하였다. 미디어 코프로세서는 범용프로세서와 같이 동작하면서 동시에 다른 미디어 데이터 형태를 처리한다. 미디어 프로세서는 마찬가지로 다른 미디어 데이터 형태를 동시에 처리하지만 동작은 범용 프로세서와 유사하다.

이 두 프로세서 형태는 범용 프로세서에서 멀티미디어 데이터를 처리할 수 있게 할 수 있는 idea를 제공해 주었다. 즉 명령어 집합 구조에 의해 멀티미디어 데이터를 처리하면 flexibility를 증가시킬 수 있다.[2][3]

본 논문에서 범용 프로세서 안에서 멀티미디어 데이터를 처리할 수 있는 명령어 집합 구조를 고안하고 이를 처리할 수 있는 프로세서의 구조를 제안하였다. 이를 검증 위해 Verilog-HDL을 사용하여 모델링하고 시뮬레이션을 수행하였다.

2. 프로세서 구조

모델링된 프로세서는 오픈 아키텍처(Open Architecture)인 SPARC V.9의 IU(Integer Unit)를 기반으로 하였다.[4][5] 즉, SPARC V.9의 IU 명령어와 64비트 데이터를 처리하기 위한 구조를 기본으로 하였으며 시간적 제어방식의 RISC 파이프라인 구조로 명령어와 데이터 캐시를 따로 지녀 성능을 최할 수 있는 하바드 구조를 지원한다.

그림 1은 모델링된 프로세서의 데이터 패스이다. 명령어 캐시에서 패치된 명령어는 id[31:0]으로 오며

oprd 레지스터에 지정되고 시간 정적 제어 방식에 따라 페치된 명령어가 opex, opm, opw 레지스터로 매 단계마다 이동하고 각 단에서 제어 신호를 발생한다.

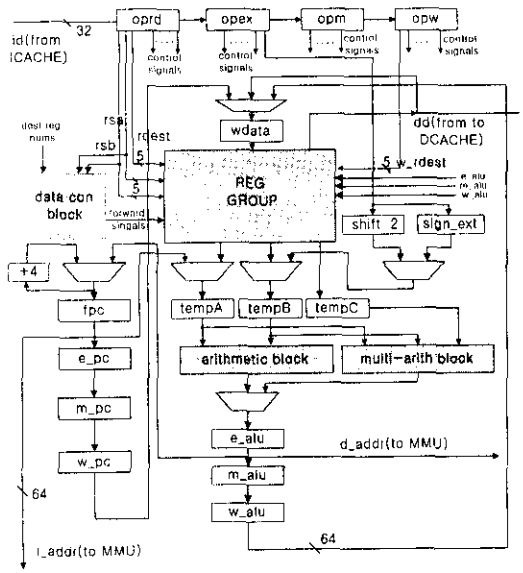

그림 1. 프로세서 데이터 패스

복성시간에서의 각 단계는 유일하므로 제어 신호는 겹치지 않으며 이 제어 신호는 데이터 패스를 제어하게 된다. REG GROUP에는 32개의 범용레지스터를 포함한 각종 특별 목적의 레지스터들이 있다. 그리고 data_con block에서의 제어 신호를 사용하여 읽기 포트에 값을 내보내는 제어 logic을 포함한다.

oprd 레지스터에서 REG GROUP로 5비트의 rsa와 rsb 신호는 소스 레지스터의 주소를 보내며 주소에 맞는 레지스터 값을 tempA와 tempB로 래치시킨다. 또한 멀티미디어 명령어 중 그 전 목적지 레지스터의 값을 소스 레지스터로 사용하는 명령어를 위해 목적지 레지스터의 주소 값인 rdest 신호를 발생한다. rdest 신호에 의해 tempC에 레지스터 값이 래치된다. 이 값들은 opex 레지스터의 제어 신호에 의해 ALU와 Shifter로 구성된 Arithmetic block에서 계산된다.

멀티미디어 명령어인 경우는 multi-arith block에서 처리되며 연산 결과 값은 파이프라인 되어 마지막의 w_alu 레지스터 값이 wdata 레지스터에 저장되고 그때 opw 레지스터에서 목적지 레지스터의 주소인 w_rdest를 보내 맞는 레지스터에 값이 저장된다.

data con block에는 forwarding과 interlock 제어 logic이 있다. 각 단에서 파이프라인된 목적지 레지스터의 주소 값과 현재 D 단계의 명령어의 소스 레지스터 주소 값인 rsa와 rsb신호를 입력으로 하여 forwarding 제어 신호를 REG GROUP으로 보낸다. 여기에는 E 단계에서 계산된 값을 각 단에서 저장한 e_alu, m_alu, w_alu값들이 들어오고 있으므로 data con block에서의 제어 신호에 의해 forwarding할 것인지 rsa와 rsb에 의해 디코드된 레지스터 값을 읽을 것인지를 결정하여 tempA와 tempB에 저장한다. load 명령어의 목적지 레지스터를 다음 명령어가 소스 레지스터로 사용하는 경우에 한 싸이클을 stall하기 위한 제어신호도 발생한다.

제어 전달 명령어가 페치된 PC 값을 저장하는 경우를 위해 PC값도 파이프라인된다. e_alu에서 fpc로 가는 값은 제어 전달 명령어의 목적지 주소를 나타낸다.

또한 d_addr은 데이터 주소 버스이므로 load나 store 명령어의 메모리 주소를 담고 있다. opex에서 나오는 신호 중에서 SETHI 명령어와 제어 전달 명령어의 목적지 주소를 계산하기 위한 left shift by 2와 sign_ext의 모듈이 있다.

fpc에서 i_addr로 나가는 값은 제어 전달 명령어가 새로운 메모리 영역에서 명령어를 패치하는 주소 값을 내보낸다.

wdata 레지스터에는 load 명령어에 의한 데이터 값, dd와 W 단계의 실과 값인 w_alu와 PC값을 저장한다. dd는 입출력 포트로 load 명령어처리 시는 입력으로 store 명령어 처리 시는 메모리에 저장할 값을 내보내는 출력 포트로 사용된다.

3. 멀티미디어 명령어 집합 구조

추가한 멀티미디어 명령어는 총 8개의 그룹에 55개의 명령어로 구분된다. 8개의 그룹은 각 subword끼리 덧셈과 뺄셈을 하는 add/sub 그룹, 비교하여 마스크를 생성하는 compare 명령어 그룹, 논리 연산을 하는 logical 명령어 그룹, 쉬프트 연산을 수행하는 shift 명령어 그룹, subword 데이터를 압축하고 확장하는 conversion 명령어 그룹, 데이터의 배열을 바꾸는 rearrange 명령어 그룹, 곱셈을 수행하는 multiply 명령어 그룹, 마지막으로 8개의 subword의 절대값을 계산하는 pixel distance 명령어 그룹 등이다.

멀티미디어 명령어는 레지스터 소스만을 사용하며 immediate 소스는 사용하지 않는다.

3.1 명령어 및 데이터 포맷

그림 2는 멀티미디어 명령어의 포맷을 나타내고 있다.

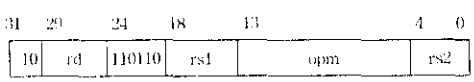

그림 2. 멀티미디어 명령어 포맷

명령어 포맷에서 [31:30]과 [24:19]까지의 필드 값은 "10110110"으로 하였는데 이는 SPARC V.9 명령어 집합에서 Implement dependant로 지정되어 있으므로 이 필드 값을 사용하였다.

rs1은 소스 1번째 레지스터 주소 값을 나타내고 rs2는 소스 2번째 레지스터 주소 값이다. 각 멀티미디어 명령어는 opm 필드 값에 의해 구분되어진다.

그림 3은 멀티미디어 명령어가 처리하는 데이터의 종류를 나타내고 있다. 총 데이터의 길이는 64비트이며 8개의 8비트 byte, 4개의 16비트 half word, 2개의 32비트 word데이터를 처리한다. 처리하는 데이터의 종류에 따라서 각 명령어 그룹의 명령어들이 구분되며 이는 opm 필드에 나타난다. opm의 하위 4비트를 byte는 "00"으로 halfword는 "01"로 word는 "11"로 할당하였다.

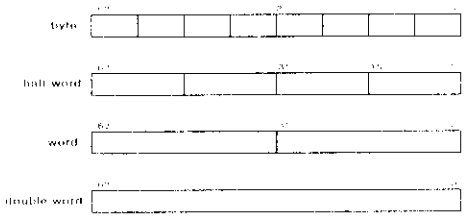

그림 3. 멀티미디어 데이터 포맷

3.2 멀티미디어 명령어 그룹

add와 sub 명령어는 각각의 byte, halfword, word의 데이터들을 덧셈과 뺄셈 연산을 한다. overflow, underflow를 무시하는 warp around방식이 연산과 overflow, underflow 발생 시에 특정 값으로 대치시키는 saturation방식의 연산을 구분하여 할 수 있다.

compare 명령어 그룹은 "greater than", "less than", "not equal", "equal"의 4개의 비교 연산을 수행한다. 소스 1 레지스터 값과 소스 2레지스터 값을 비교하여 결과 값이 참이 되면 목적지 레지스터의 각 subword의 비트들에 '1'을 저장하고 결과 값이 거짓인 경우는 '0'으로 채워진다. 이렇게 저장된 비트 마스크는 주로 분리연산 명령어의 소스 레지스터 값으로 사용된다.

logical 명령어 그룹은 "and", "or", "xor", "xnor", "andnot"의 연산을 수행하는 명령어들과 목적지 레지스터 값을 모두 '1'로 채워주는 명령어인 mone과 '0'으로 채워주는 mzero 명령어를 포함한다.

shift 명령어는 다른 일반 쉬프트 명령어와 같이 오른쪽, 왼쪽의 쉬프트와 logical, arithmetic 방식의 명령어로 구분된다. 처리 데이터 크기는 halfword, word, double word로 구분된다. 데이터의 변환과 정렬에 사용될 수 있으며 수행방식은 소스 1 레지스터의 각 subword를 소스 2 레지스터의 하위 4비트, 5비트, 6비트의 값에 따라서 쉬프트하게 된다.

conversion 명령어 그룹은 멀티미디어 데이터 처리 시에 데이터의 크기가 변하는 intermediate form을 요구하는 처리가 종종 발생하는데 이 때에 사용되는 명령어 그룹이다. 16비트의 halfword의 subword를 8비트의 byte단위로, 32비트의 word를 8비트의 byte로, 32비트의 word를 16비트의 halfword로 변환하는 명령어를 포함한다. 반대로 8비트의 byte단위와 subword를 16비트의 halfword로, 64비트의 doubleword로 확장시키는 명령어도 포함된다.

rearrange 명령어 그룹은 모두 4개의 명령어로 구성되어 있다. 모두 64비트의 소스레지스터의 하위 32비트 데이터 안에서 8비트 4개의 subword와 16비트 2개의 subword의 자리를 재배치한다.

multiply 명령어 그룹은 모두 6개의 명령어로 구성되어 있다. 4개의 8비트 unsigned 데이터와 16비트 signed 데이터와의 곱셈을 하여 상위 16비트 값을 취하는 명령어, 두 소스 레지스터의 4개의 16비트 subword끼리 곱셈을 하여 16비트 데이터 값을 생성하는데 필요한 명령어, 두 소스 레지스터의 2개의 16비트 subword끼리 곱셈을 하여 32비트 데이터 값을 생성하는데 필요한 명령어들을 포함한다. 또한 곱셈 후에 덧셈을 연산하는 명령어도 있다.

pixel distance 명령어는 두 64비트 데이터 안의 8개의 8비트 subword끼리의 절대값을 계산하여 더하게 된다. 즉 소스 1 레지스터의 각 subword와 소스 2 레지스터의 subword끼리 절대값을 취한 후 목적지 레지스터 값과 계산된 절대값들을 더하게 된다. 이 명령어는 MPEG 2의 motion estimation에 사용된다.

표 1은 각 멀티미디어 명령어 그룹의 opm필드의 상위 4비트를 나타낸다.

표 1. 멀티미디어 명령어 그룹 코드 assign

명령어 그룹	opm[7:4]	명령어 그룹	opm[7:4]
add/sub	0101	conversion	0011
compare	0010	rearrange data	1000
logical	0110	multiply	1001
shift	0111	pixel distance	0011

4. 명령어 적용 예

제안된 멀티미디어 명령어를 멀티미디어 데이터 처리 시에 사용되는 간단한 연산에 적용하였다.

4.1 벡터 내적

```
short inmat_a[N], inmat_b[N][N], outmat[N], i, j;
for(i=0; i<N; i++){
  for(j=0; j<N; j++){
    outmat[i] += inmat_a[j] * inmat_b[j][i];---①
  }
}
```

프로그램 1. 벡터 내적

프로그램 1의 코드는 두 행렬의 내적을 수행하는 프

로그램의 원부이다. intmat_a는 첫 행렬의 배열이름이며 inmat_b는 두 번째 행렬의 배열이다. outmat은 저장한 배열 값의 이름이다. 'N'은 행렬의 크기를 나타낸다. 만약 N이 4일 때 대부분의 수행시간은 (1)에서 소비되며 이 부분을 멀티미디어 명령어를 사용하지 않을 때 4^2 = 16개의 곱셈명령어와 4*(4-1) = 12개의 덧셈 명령어가 필요하다. 이를 제안한 멀티미디어 명령어 집합에 적용하면 4개의 mmuladdh 명령어와 2개의 maddw 명령어로 구현가능하다.

4.2 행렬 변환

```
int mat[N][N], temp[N][N], i, j;
for(i=0; i<N; i++){
  for(j=0; j<N; j++){
    mat[i][j] = temp[j][i]; ------------(2)
  }
}
```

프로그램 2. 행렬 변환

프로그램 2는 크기가 N*N인 정방행렬의 전치행렬(Transpose matrix)을 구현한 C 코드이다. mat는 입력 행렬이며 temp는 스왑(swap)에 필요한 임시 행렬이다. temp 배열에는 입력행렬의 값이 똑같이 들어있어야 한다. 수행시 대부분의 시간은 (2)의 연산부분에서 소요된다. N이 4일 때 멀티미디어 명령어를 사용하지 않으면 2*(4-2)+1 = 5개의 스왑 연산을 하므로 3*(2*(4-2)+1) = 15개의 mov 명령어가 필요하다. 제안한 멀티미디어 명령어 집합에 적용시키면 8개의 mmix 명령어로 수행할 수 있다.

5. 시뮬레이션

제안된 멀티미디어 명령어를 검증하기 위해서 어셈블리 라인에서 명령어들의 스티뮬러스 벡터(stimulus vector) 값을 생성하였다. 먼저 모든 멀티미디어 명령어들을 검증하기 위하여 각 명령어 그룹에서 몇 개씩 선택하여 미리 적절한 값들을 레지스터에 치리할 레지스터들을 설정하고 기계어 코드를 핵사 값으로 생성하였다. 다음으로 4장에서 적용한 명령어들의 스티뮬러스 벡터 값들은 2개의 부 프로그램으로 구성하였다.

그림 4는 시뮬레이션 환경을 나타낸다. 거술된 프로세서 모듈이 processor core에 해당되고 멀티미디어 명령어를 포함한 2개의 부 프로그램이 각각 multimedia inst cache1과 multimedia inst cache2에 저장된다. 멀티미디어 명령어가 아닌 일반 명령어는 general inst cache에 저장하였다. 각각의 명령어 cache와 데이터 cache, 클럭 및 리셋 발생 부분 역시 Verilog-HDL로 동작 수준에서 기술하였으며 CADENCE의 simwave 툴을 사용하였다.

4장의 프로그램을 적용하여 시뮬레이션한 결과 성능이 프로그램 1에서는 대략 4배, 프로그램 2에서는 2배가 향상됨을 알 수 있었다.

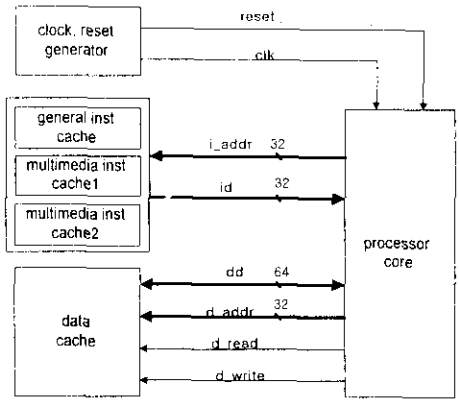

그림 4. 시뮬레이션 환경

결론

본 논문에서는 범용 프로세서 안에서 멀티미디어 정보를 효율적으로 처리하기 위해 멀티미디어 명령어 집합 구조와 이를 처리하기 위한 프로세서의 구조를 제안하였다.

제안한 멀티미디어 명령어는 연산 종류에 따라 8개의 그룹에 총 55개의 명령어로 구성되며 처리하는 데이터는 64비트 데이터 안에서 8비트의 8개 바이트, 16비트의 4개 하프워드, 32비트의 2개 워드이며 모두 각각의 subword를 병렬적으로 처리하는 특성을 지닌다.

프로세서는 SPARC V.9를 기반으로 하여 명령어 캐시와 데이타 캐시를 따로 사용하는 하바드 구조를 채택하고 시간 정적 제어 방식의 5단 파이프라인 RISC 구조를 갖는다.

이를 Verilog-HDL로 모델링하고 CADENCE 사의 simwave 툴을 사용하여 간단한 멀티미디어 응용 프로그램을 C 코드로 작성하여 이를 기술된 프로세서에 적용시켜 시뮬레이션을 수행하였다.

[참고문헌]

[1] Ruby B. Lee and Michael D. Smith, "Media Processing : A New Design Target", IEEE Micro, pp.6-9, Aug, 1996.

[2] M.Tremblay, J.M.O'Connor, V.Narayanan and L.He, "VIS Speeds New Media Processing", IEEE Micro, pp.10-20, Aug, 1996.

[3] Alex Peleg and Uri Weiser, "MMX Technology Extension to the Intel Architecture", IEEE Micro, pp.42-50, Aug, 1996.

[4] "The SPARC Architecture Manual Version 9", SPARC International, Inc., 1994.

[5] "UltraSPARC User's Manual", SUN microsystems, July, 1997.