한국지능시스템학회 논문지 2008, Vol. 18, No. 3, pp. 323-328

다중 명령어 처리 DSP 설계

A Design of Superscalar Digital Signal Processor

박성욱
Sung-Wook Park

삼성전자 DM연구소

요 약

본 논문에서는 연산 중심의 DSP 작업에 대한 성능을 유지하면서 제어 작업을 효과적으로 수행할 수 있는 프로세서 구조를 제안하고 구현하였다. 전통적으로 DSP작업은 직렬 연결된 연산기로 구현되지만, 제안한 프로세서에서는 곱셈기, 2개의 ALU, 읽기/쓰기 유닛 등 4개의 실행 유닛이 병렬로 배치되어 있고 수퍼스칼라 방식으로 제어되므로 동시에 처리된다. 제안된 프로세서를 사용하여 AC-3 오디오 복호화기를 구현하여 성능이 37.8% 향상됨을 확인하였다. 이와 같은 연구는 기존의 고성능 DSP를 사용할 수 없는 저가격의 가전기기용 부품제작에 활용이 가능하다.

키워드 : 다중명령어처리, 프로세서, 제어 작업, DSP 작업

ABSTRACT

This paper presents a Digital Signal Processor achieving high through-put for both decision intensive and computation intensive tasks. The proposed processor employees a multiplier, two ALU and load/store Unit as operational units. Those four units are controlled and works parallel by superscalar control scheme, which is different from prior DSP architecture. The performance evaluation was done by implementing AC-3 decoding algorithm and 37.8% improvement was achieved. This study is valuable especially for the consumer electronics applications, which require very low cost.

Key Words : Superscalar, signal processor, decision intensive task, computation intensive task

Ⅰ. 서 론

미국과 유럽 디지털 방송의 오디오 표준인 AC-30와 MPEG-20는 높은 압축율을 얻기 위해 여러 가지 기법들을 조합하여 사용하고 있다. 이들 기법들은 요구하는 연산의 속성을 기반으로 다음의 3가지로 분류할 수 있다. 먼저 비트열 분석, 동작 모드 설정, 외부 시스템과의 통신 등과 같은 제어작업(control-intensive task), 두번째로 필터링, 행렬 곱셈, 변환(transform) 등 연산이 많은(computation intensive) DSP (Digital Signal Processing) 작업, 그리고 마지막으로 심리음향 모델을 이용한 마스킹 곡선 계산, 크기인자 추출, 테이블 참조 등의 혼합 작업을 들 수 있다.

제어 작업은 연산량은 적으나 조건 판별과 단순 연산을 반복적으로 수행한다. 이때 처리과정과 반복 횟수가 작업수행 시 입력되는 신호에 따라 실시간으로 결정되는 속성을 가지고 있다. DSP 작업은 입력 데이터의 속성과는 상관없는 반복작업이며 대부분 곱셈 및 누산(Multiplication and ACcumulation)로 구성되어 있고 그 횟수도 미리 정해져 있다. 마지막인 혼합작업은 선형 근사화된 초월함수 계산과 같이 제어작업과 DSP작업이 혼합되어 있다.

오디오 압축 알고리듬을 분석해보면 제어작업과 혼합작업이 60~70% 가량이며, DSP 작업은 30%을 차지한다.0 그러므로 전통적인 DSP를 오디오 디코딩에 사용하는 것은 적합하지 않다. 전통적인 DSP는 DSP 작업에 최적화 되어 있어 곱셈기, ALU, Shifter 를 직렬로 연결하여 사용하고 있지만, 제어 작업과 혼합작업은 MAC연산이 상대적으로 적으므로 직렬 연결된 연산기 중 하나만 사용할 수 밖에 없기 때문이다.

이와 같은 기존 DSP의 문제점을 개선하기 위한 연구들이 수행 되어 왔다000. 그러나 이들 연구들이 다중 코어를 활용하여 제어 작업과 DSP작업에서 각각 고성능을 가지도록 개선하였지만, 한번에 하나의 코어를 사용함에 따라 연산 유닛(functional Unit)의 활용도가 떨어지는 문제점을 가지고 있다. 본 논문에서는 오디오 알고리듬에 대하여 제어작업 및 DSP작업에서 고성능을 가지면서도 연산유닛을 활용도를 제고할 수 있는 프로세서 구조를 제안하였다.

접수일자 : 2007 년 10 월 3 일
완료일자 : 2008 년 5 월 9 일

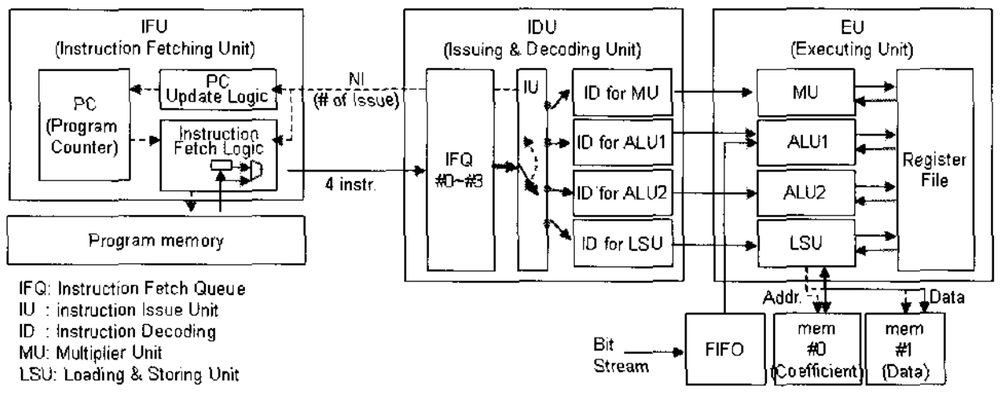

그림 1. 프로세서 내부 구조

Fig. 1 Processor Architecture

Ⅱ. 구조 설계

프로세서에서 명령어 처리과정은 크게 3단계로 분류할 수 있다. 명령어를 메모리로부터 읽어오는 과정(명령어 읽기:Instruction Fetch), 명령어를 해석하는 과정(명령어 해석:Instruction Decoding) 그리고 해석된 명령어를 수행하는 과정(명령어 실행:Instruction Execution)이 그것이다. 이러한 명령어 처리 단계는 순차적으로 진행되어야 하며 각 단계는 최소한 1개의 시스템 사이클을 요구한다. 그러므로 1개의 명령어를 완료하기 위해서는 최소한 3 사이클이 소요된다. 명령어 완료 시간을 단축시키는 방법으로써 명령어 처리의 각 단계를 중첩 시키는 파이프라인처리 기법이 주로 사용된다.

파이프라인처리 기법은 명령어 처리 효율을 높이기 위해 각 단계가 동일한 소요시간을 가지도록 설계되는 것을 전제로 한다. 이러한 관점에서 기존 DSP를 보자면, 명령어 읽기와 명령어 해석에 비하여 명령어 실행 단계의 소요시간이 직렬로 연결된 곱셈기, 누산기, Shifter 등 여러 연산 유닛에 의해서 상대적으로 길다. 본 논문에서는 이러한 실행 단계의 소요시간을 단축하기 위해서 기존 DSP가 가지는 연산유닛 중 곱셈기와 ALU를 병렬로 배치하고, 명령어 세트를 단순화하여 하나의 명령어가 하나의 연산유닛을 제어하도록 하였다. 이를 통하여 신호가 통과해야할 critical path가 짧아져 신호의 setup 시간이 줄어드는 장점을 얻게 된다. 또한 하바드 구조를 구성하기 위해 사용되는 메모리 제어유닛의 ALU 2개를 병렬연산이 가능하도록 분리함으로써 프로세서 전체적인 처리능력을 향상할 수 있도록 하였다. 이렇게 병렬 배치된 4개의 연산기가 동시에 동작할 수 있도록 제어하기 위하여, 4개의 명령어를 동시에 처리할 수 있는 4-way 수퍼스칼라 구조를 도입하였다.

그림 1은 제안된 DSP의 전체 구조이다. 명령어 페치 유닛(Instruction Fetching Unit), 이슈/디코딩 유닛(instruction Issuing & Decoding Unit) 그리고 실행 유닛(Executing Unit)로 구성된다. 각기 1 시스템 사이클에 동작하며 레지스터 파일을 통하여 파이프라인이 가능하도록 구성하였다.

1. 페치 유닛 (Instruction Fetching Unit)

페치 유닛은 내부 프로그램 카운터(PC)를 가지고 있어 현재 읽어와야 할 명령어의 주소를 저장해 놓는 다. Instruction Fetch Logic은 PC가 지시하는 번지부터 4개의 명령어를 읽어 내부 레지스터와 이슈/디코딩 유닛의 명령어 레지스터인 IFRQ0~IFQ3에 저장한다. Instruction Fetch Logic의 내부 레지스터는 명령어가 PC가 4의 배수가 아닌 경우 4개의 명령어를 이슈/디코딩 유닛에 전달하기 위해서 사용된다. 그리고 PC update logic은 순차적인 진행시의 n번째 사이클에 PC 값을 다음 식을 기반으로 구한다.

$$PC(0) = 0$$

$$PC(n) = PC(n-1) + NI \qquad \text{식(1)}$$

여기서 NI은 이슈/디코딩 유닛에서 주어지는 값으로 n 번째 사이클에서의 이슈된 명령어 개수이다. 프로그램의 흐름을 변경하는 명령어인 CALL, RTN(Return), JUMP, BR(Branch), 그리고 LOOP 와 같은 명령어가 이슈/디코딩 유닛에서 처리되는 경우는 예외적으로 PC 값은 명령어가 지정하는 값으로 정해진다. 이후에 다시 순차적인 진행이 이루어지면 식(1)에 따라 PC값이 변경된다.

2. 이슈/디코딩 유닛(instruction Issuing & Decoding Unit)

복수개의 연산기를 동시 제어하는 수퍼스칼라 프로세서를 구현하기 위해서는 명령어 이슈과정이 추가구현되어야 한다. 명령어 이슈는 프로그램 메모리의 일부 영역에 대해 원도우를 취한 후 그 명령어 윈도우(instruction window)내에 동시에 실행할 명령어를 찾아내는 작업으로, 이 작업은 명령어 상호간 자원 충돌(Resource Conflict) 데이터 의존성 (Data Dependency)이 발생 여부를 조사하는 것이다. 그러나 이 과정은 많은 실리콘 면적과 처리시간을 요구하는 어려움이 있다.

명령어 이슈 방법에는 순차적 이슈(In-Order Issue)와 비순차적 이슈(Out-of-Order)의 2가지가 이다. 비순차적 이슈란 현재 명령어가 앞의 명령어와 충돌이나 의존성이 발생하면 그 다음 명령어들이 대해서 계속 동시 수행이 가능한 명령어를 골라내 실행할 수 있도Rate)가 높아지는 장점이 있다. 하지만 그 만큼 복잡하고 많은 로직을 요구한다. 반면 순차적 이슈는 어떤 명령어와 다음 명령어 사이에 충돌이나 의존성이 하나라도 존재하면 이슈를 중단한다. 따라서 이슈율이 낮지만 구현이 간단한 장점이 있다. 그러나 순차적인 이슈라 할지라도 명령어를 프로그램 메모리에 싣기 전에 컴파일 과정에서 이러한 충돌이나 의존성이 최소화 되도록 명령어를 미리 재구성한다면 비순차적 이슈와 동일한 이슈율을 가지도록 할 수 있다. 그러므로 본 논문에서는 컴파일러와 순차적 이슈 방법을 채용하여 비교적 간단한 방법으로 고성능을 얻을 수 있도록 하였다.

이슈/디코딩 유닛은 IFQ0~IFQ3에 저장된 명령어에 대하여 자원 충돌 및 데이터 의존성을 검사한다. 그리고 충돌이나 의존성이 발생하면 검사를 중단하고, 검사 중단 이전에 이슈된 명령어만을 각 연산 유닛용 명령어 해석기에 보낸다. 자원의 충돌은 명령어의 명령어 코드(Op-code)로 식별하며, 데이터 의존성은 현재 명령어가 이전 명령어의 결과 레지스터를 소스로 사용하는 경우를 점검함으로써 식별한다.

그림 2은 64-tap FIR Filter의 보기에서 명령어 이슈 과정의 일 예를 보여 준다. 명령어 #7 (mult acc0, r0, r1)은 데이터 의존성에 의해서 cycle #2에서 이슈되지 못하고 cycle #3에서 이슈된다. 이때, cycle #2에서 데이터 의존성과 자원 충돌이 없는 명령어 #10을 수행할 가능성을 모색할 수도 있으나, 순차적인 이슈 방법을 따르는 프로세서이므로 cycle #2에서 수행되지 않았다. 프로그램의 흐름을 제어하는 CALL, RTN(Return), JUMP, BR(Branch), 그리고 LOOP는 cycle#4와 같이 단독으로 이슈된다.

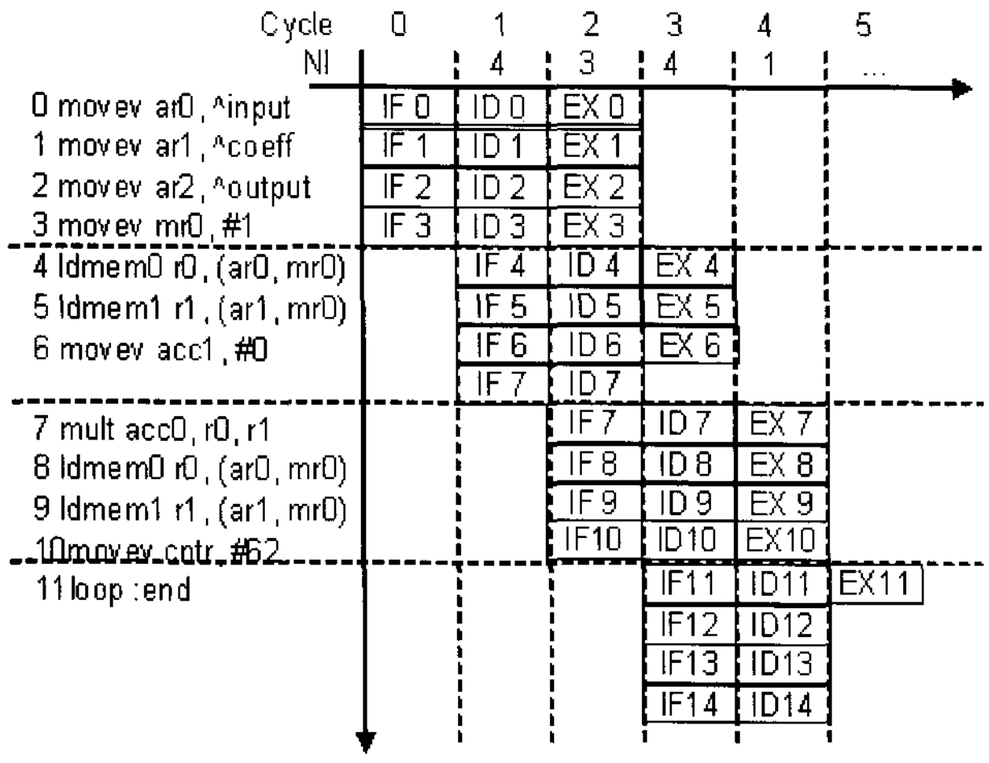

그림 2. 명령어 이슈 및 PC(Program Counter) 갱신 과정의 보기

Fig. 2 Example of Instruction Issue and PC(Program Counter) Update

3. 실행 유닛 (Execution Unit)

실행 유닛은 레지스터 파일(register file), 곱셈 유닛(MU: Multiplier Unit)과 ALU1, 그리고 하버드 아키텍처 지원을 위한 메모리 어드레스 생성 유닛을 분리한 ALU2와 읽기/쓰기 유닛(LSU: Load/Storage Unit)으로 구성된다.

3-1. 레지스터 파일

레지스터 파일은 13개의 단정도(16bit) 범용 레지스터(R0-R12)와 4개의 배정도(32-bit) 레지스터 (DWR0-DWR3: Double Word Register), 4개의 40bit 누산용 레지스터(ACC0-ACC3: Accumulator), 4개의 14bit 주소 레지스터(AR0-AR3: Address Register), 그리고 2개의 변경 레지스터 (MR0-MR1: Modify Register)로 구성된다.

배정도 레지스터는 2개의 단정도 레지스터로 사용될 수 있도록 설계하여, 1개의 32bit 값을 저장하거나 2개의 16bit값을 저장할 수 있다. 이러한 구조를 사용하여 서브 워드 프로세싱(sub-word processing)이 가능하다. 즉 1개의 데이터를 몇 개의 필드로 분리하여 각각이 독립적인 데이터를 의미하도록 하면 병렬로 배치된 여러 개의 연산기가 이들을 동시에 처리할 수 있도록 하여 FFT와 같은 복소수를 사용하는 응용에 활용가능하다. 만일 연산 유닛의 입력이 지정된 레지스터 보다 많은 비트를 요구할 경우 부호확장이 이루어진다.

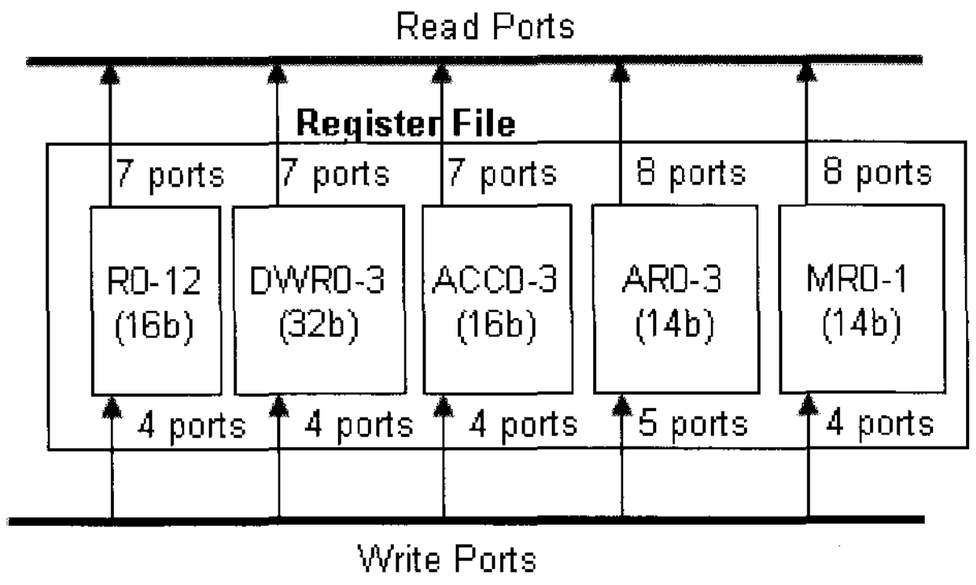

그림 3. 레지스터 파일 구조

Fig. 3 Architecture of Register File

주소 레지스터와 변경 레지스터를 제외한 나머지 레지스터는 4개의 쓰기 포트와 7개의 읽기 포트를 가지고 있다. 주소 레지스터는 8개의 읽기 포트와 5개의 쓰기 포트를 가지고 있으며, 변경 레지스터는 8개의 읽기 포트와 4개의 쓰기 포트를 가진다.

3-2. 곱셈 유닛(Multiplier Unit)과 ALU1

곱셈 유닛은 MULT 명령어로 2개의 16bit 입력을 받아 곱한 후 32bit 결과를 결과 버스에 싣는다.

ALU1은 연산의 반복으로 일시 숫자가 커지거나 작아져 정밀도가 손실되는 것을 방지하기 위해 40bit를 처리할 수 있도록 설계되어 있다. 기본적인 산술 연산인 ADD, SUB, NEG, ABS 그리고 기본적인 논리 연산인 AND, OR, XOR, NOT, SHIFT를 지원한다. 또한 오디오 스트림 복호화 및 연산 시 정밀도 확보를 위해서 UNPACK명령어와 RND 명령어를 ALU1에 추가하였다.

UNPACK 명령어는 FIFO에 저장된 비트열에서 n개의 bit를 읽어 들여 특정 레지스터에 저장하는 기능을 1사이클에 수행한다. 이와 같은 동작은 모든 복호화 알고리듬 구현에 필요한 기능으로써, Shifter만으로 수행하는 경우 4개 이상의 명령어가 필요하다. UNPACK 모듈의 동작은 다음과 같다.

먼저 복호화될 비트열은 16k-bit FIFO에 입력되며, 입력된 비트열 중 첫 두개의 16 bit codeword가 UNPACK 모듈 내부 레지스터에 저장된다. UNPACK (n-bit) (1<=n<=16) 명령어가 수행되면 내부 레지스터에서 n bit를 결과버스에 싣는다. UNPACK 모듈은 내부 레지스터가 비어있는 경우에 FIFO로부터 16bit 씩 읽어서 채워 넣는다.

RND 명령어는 반올림(round)연산이다. 이 연산은 32-bit 레지스터의 값을 16-bit 레지스터에 저장하는 경우 발생하는 정확도 오차(precision error)를 통계적으로 최소화하기 위하여 도입하였으며 round-to-nearest- even0 알고리듬을 사용하였다.

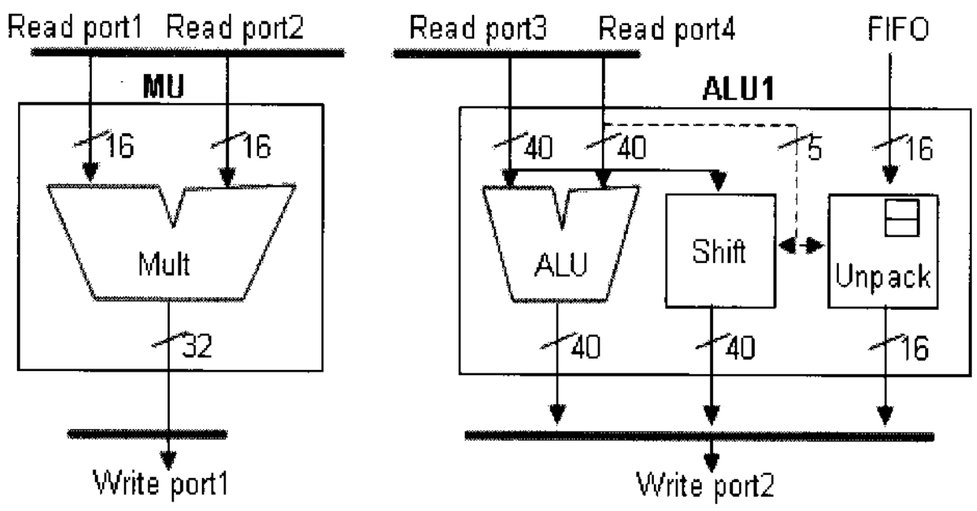

그림 4. 곱셈 유닛과 ALU1 의 구조
Fig. 4 Architecture of Multiplier Unit and ALU1

3-3. 읽기/쓰기 유닛(Load/Store Unit)과 ALU2

DSP 알고리듬은 대량의 데이터에 대한 연산을 수행하는 특징을 가지고 있어 레지스터에 대한 연산보다는 메모리에 대한 연산이 주를 이룬다. 통상적인 DSP 알고리듬은 입력데이터와 미리 지정된 계수들을 곱하고 그 결과를 누적하는 것이다. 이 때 입력 데이터와 계수들은 서로 다른 영역의 메모리에 저장되어 있으며, 프로세서는 이 다른 메모리로부터 동시에 값을 읽어 온다. 그러므로 입력데이터를 읽고 저장하는 주소 생성과 계수를 읽는 주소 생성을 각각 Linear, Modulo, bit-reverse의 조합과 Module 모드로, 동시에 수행할 수 있어야 한다.

제안된 프로세서에서 메모리 접근은 기본적으로 읽기/쓰기 유닛이 수행한다. 읽기/쓰기 유닛은 직접 어드레싱 뿐만 아니라, 주소 레지스터와 변경 레지스터를 조합한 간접 어드레싱을 지원한다. 읽기/쓰기 유닛은 주소 레지스터로 지정된 주소에 해당하는 메모리를 접근함과 동시에 주소 레지스터와 연계된 변경 레지스터의 값을 주소 레지스터의 값과 더하거나 뺌으로써 갱신될 주소를 만든다. 이로써 후변경 간접 어드레싱(Post-modify Indirect Addressing)을 수행한다

두 개의 메모리를 동시에 접근하기 위해서는 모두 두 개의 후변경 간접 어드레싱을 위한 회로가 필요하다. 다시 말해서, 읽기/쓰기 유닛이 데이터를 저장하고 있는 mem#1을 접근하고 있는 경우, 계수를 저장하고 있는 mem#0를 읽기 위한 주소 생성이 필요하다. 본 논문에서는 이 추가되는 회로를 주소 뿐만이 아니라 다른 연산도 함께 처리할 수 있는 구조를 고안하였고 이를 제안하였다

제안된 프로세서에서는 그림 5에서와 같이 mem#0을 읽기 위한 주소 생성을 위해서 ALU2를 이용한다. ALU2는 레지스터 파일이나 명령어로부터 데이터 ROM을 읽기 위한 주소를 받아 읽기/쓰기 유닛에 넘겨준다. 읽기/쓰기 유닛은 그 주소를 기반으로 mem#0를 읽어 그 결과를 다시 ALU2에 넘겨 주고 ALU2는 그 값을 받아 결과 버스에 싣는다. ALU2는 어드레스 생성 기능과 함께 ALU1과 동일한 산술 및 논리연산 등을 수행할 수 있도록 확장된 것으로써 ALU1이 처리할 수 있는 명령어를 동시에 처리하는 것도 가능하다. 그러므로 ALU2는 주어진 알고리듬의 요청에 따라 MAC 연산을 위한 어드레스 생성 기능 혹은 산술/논리 연산 기능을 선택하여 수행할 수 있는 기능을 가지고 DSP의 유연성을 높여 주는 역할을 한다. 두 개의 ALU는 Sub Word Processing이 가능하도록 해준다.

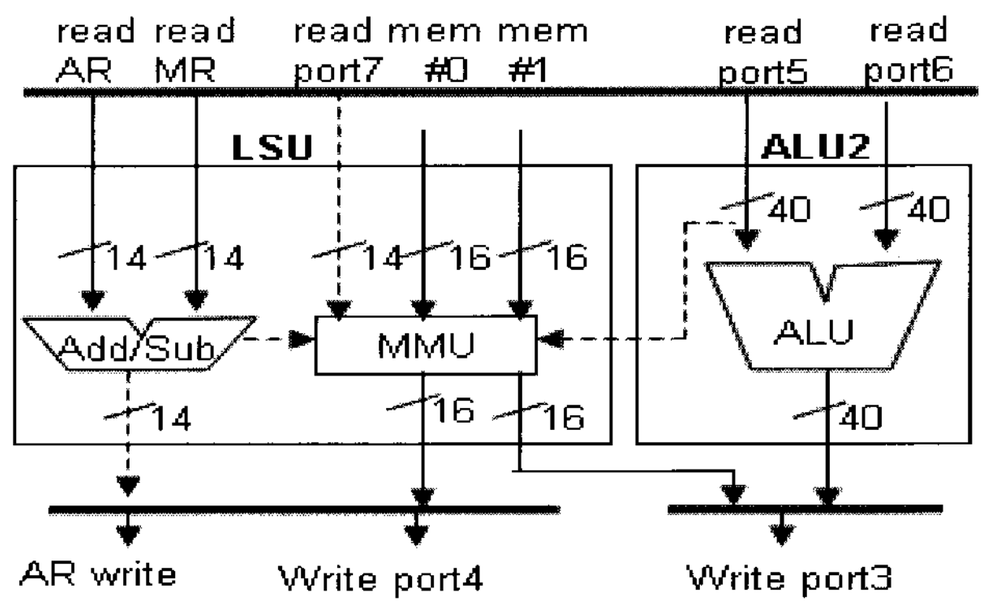

그림 5. 읽기/쓰기 유닛과 ALU2 의 구조
Fig. 5 Architecture of Load/Store Unit and ALU2

Ⅲ. 결과

제안된 프로세서는 VHDL을 이용하여 설계되어 표준 셀로 합성되었고, 전체 게이트 수는 43,993 게이트였다. 표 1은 제안된 프로세서의 명령어 집합을 나타 낸다.

또한 실제 오디오 알고리듬에서의 성능향상 정도를 확인하기 위하여 AC-3 디코딩 과정을 프로그래밍한 후 모의 실험을 수행하였다.

AC-3 디코딩 알고리듬에 대한 모의 실험결과를 표 2에 나타내었다. 표 2은 대표적인 DSP 작업인 inverse transform을 제외한 제어 작업 및 혼합 작업들을 구현하는데 필요한 cycle수를 나타낸 것으로써, 전체적으로 37.8%의 성능향상을 보았다.

표 1 명령어 집합
Table 1 Instruction Set

	Instruction	Operation
1	AND dst, src1, src2	Src1 AND scr2 → dst
2	OR dst, src1, src2	Src1 OR scr2 → dst
3	NOT dst, src	NOT src → dst
4	ADD dst, src1, src2	Src1 + scr2 → dst
5	SUB dst, src1, src2	Src1 - scr2 → dst
6	ABS dst, src	ABS(src) → dst
7	NEG dst, src	- src → dst
8	RND dst, src	Round(src) → dst
9	CMP src1, src2	Set status flag if src1 >= src2
10	MULT dst, src1, src2	Src1 x scr2 → dst
11	MOVE dst, src	Src → dst
12	SHIFT dst, src, value SHIFT dst, src, #value	Shift(src) by value → dst Shift(src) by #value → dst
13	UNPACK dst, value UNPACK dst, #value	Extract value bits from FIFO → dst
14	LDMEM0 dst, (addr) LDMEM0 dst, #addr	Mem0(addr) → dst
15	LDMEM1 dst, (addr) LDMEM1 dst, #addr	Mem1(addr) → dst
16	STORE src, (addr) STORE src, #addr	Src → Mem1(addr)
17	LOOP (addr) LOOP #addr	Addr 까지 Counter 값만큼 반복 수행
18	CALL (addr) CALL #addr	Addr 의 sub-routine 수행
19	RETURN	Sub-routine 에서 복귀
20	JUMP (addr) JUMP #addr	Addr 로 PC 변경
21	BRT cond, (addr) BRT cond, #addr	Cond 조건이 status 와 일치할 것으로 예측하여 PC 설정
22	BRN cond, (addr) BRN cond, #addr	Cond 조건이 status 와 일치할 것으로 예측하여 PC 설정
23	BRD cond, (addr) BRD cond, #addr	예측없이 다음 명령어를 PC 설정
24	MOVEV dst, #value	16bit value 값 → dst
25	NOP	NO Operation

LEGEND:
- dst: Destination register
- src: Source Register
- #value,#addr: 명령어를 통하여 값 직접 입력
- value : Register 를 통한 값 입력
- (addr): 주소 register 및 변경 register 를 통한 간접 어드레싱

표 2 AC-3 알고리듬의 주요 Task 수행시간 (1 frame, 5.1 channel, bit-rate=448 Kbps, fs=48KHz)
Table 2 Execution Time of AC-3 Algorithm Tasks

Decoding process	Typical DSP (# of cycles)	Proposed DSP (# of	Gain (%)
bit-unpacking	300	75	75.0
exponent decoding	25,134	12,566	50.0
bit allocation	169,038	84,513	50.0
Mantissa decoding	137,460	109,791	20.1
channel de-coupling	8,490	4,686	44.8
Total sum	340,422	211,631	37.8

Ⅳ. 결 론

본 논문에서는 통상적인 DSP 작업뿐만 아니라 제어 작업과 혼합 작업에서도 효과적인 동작을 수행하는 프로세서의 구조를 제안하고 구현하였다. 제안된 프로세서는 4개의 연산 유닛을 병렬로 배치한 후 수퍼스칼라 방식으로 동시 동작시켰다. 또한 각 연산기는 레지스터 파일을 통하여 파이프라인 구조로 동작하는 것이 가능하여 MAC(Multiplier & Accumulator)를 기반으로 하는 DSP연산 또한 사이클의 낭비 없이 수행이 가능하다. 이와 같은 연산기 구조는 기존의 연산 유닛인 MAC 및 데이터 주소 생성기를 확장한 것으로써 시스템 복잡도 증가가 미미하며, 연산 유닛의 활용 측면에서의 유연성이 향상된 것이다. 향후 명령어 충돌과 데이터 의존성을 미리 고려하는 컴파일러와 함께 사용될 경우 실험 결과보다 더 나은 성능을 얻을 수 있을 것이다.

참 고 문 헌

[1] ISO/IEC JTC1/SC29/WG11, *Generic Coding of Moving Pictures and Associated Audio*- CD 13818-3 (MPEG-Audio), Mar. 1994

[2] Advanced Television Systems Committee (ATSC), *Digital Audio Compression Standard (AC-3),* Doc. A/5, Nov. 1994

[3] Steve Vernon, "Design and implementation of AC-3 coders," *IEEE Transaction on Consumer Electronics*, Vol. 41, No. 3, pp. 754-759, Aug., 1995

[4] Christoph Baumhof, "A Novel 32 Bit RISC Architecture Unifying RISC and DSP," *ICASSP*, pp. 587-590, 1997

[5] He Qing and Hou Chao Huan, "RNIW: A novel general purpose DSP architecture," *ICASSP*, pp. 3302-3305, 1996

[6] H. Sato, E. Holmann, "A dual-issue RISC processor for multimedia signal processing," *ICASSP*, pp.591-594, 1997

[7] Colwell, R. P. et al., "A VLIW architecture for a trace

scheduleing compiler," *IEEE Transactions on Computers*, 37:(8), 1998.

[8] Joseph A. Fisher, "Very Long Instruction Word Architectures and the ELI-512," *Proceedings of the 10th Symposium on Computer Architecture,* pp. 140-150, June, 1983

[9] Mike Johnson, *Superscalar Microprocessor Design*, Prentice Hall, Inc., pp. 17-24, 1991

[10] Istael Koren, *Coputer Arithmetic Algorithms*, Prentice Hall International Editions, pp. 58-68, 1993

[11] Henrik V, Sorensen et al., "Real-Valued Fast Fourier Transform Algorithms," *IEEE Trans. on ASSP*, Vol. ASSP-35, No. 6, pp. 849-863, June 1997

저 자 소 개

Sung-Wook Park received the B.S., M.S., and Ph.D. degree in Electronic Engineering from Yonsei University in 1993, 1995, and 1998, respectively. He is now working for Samsung Electronics. His research interest includes VLSI signal processing, Multimedia Signal Processing and System.