論文 98-23-6-1

4-way 수퍼 스칼라 디지털 시그널 프로세서 코어 설계

正會員 김 준 석*, 유 선 국**, 박 성 욱*, 정 남 훈*, 고 우 석*, 이 근 섭*, 윤 대 희*

On Designing 4-way Superscalar Digital Signal Processor Core

Joon-Seok Kim*, Sun-Kook Yoo**, Sung-Wook Park*, Nam-Hoon Jung*,
Woo-Suk Ko*, Keun-Sup Lee*, Dae-Hee Youn* *Regular Members*

요 약

최근의 오디오 압축 알고리듬은 다양한 코딩 기법을 조합하여 사용하고 있다. 이들은 DSP 작업(DSP task), 제어 작업(controller task), 그리고 혼합 작업(mixed task)으로 나눌 수 있다. 기존의 DSP 프로세서들은 이들중 DSP 작업만을 효율적으로 처리하도록 설계되어 있어 제어작업이나 혼합 작업에 대해서는 자원을 효율적으로 활용하지 못하는 단점이 있다. 본 논문에서는 기존의 DSP 프로세서가 가지는 DSP 작업에 대해여 고성능을 그대로 유지하면서 제어작업과 혼합작업에도 좋은 성능을 가지는 새로운 구조를 제안하고 구현하였다. 제안된 프로세서 YSP-3는 4개의 실행 유닛 (곱셈기, 2개의 ALU, 메모리 접근 유닛)을 병렬로 배치한 후 4-way 수퍼스칼라 명령어 구조를 사용하여 각 유닛을 독립적으로 사용할 수 있도록 하였다. 제안된 구조는 일반적인 DSP 알고리듬과 AC-3 디코딩 알고리듬을 실행하여 성능을 평가하였다. 마지막으로 VHDL을 통해 0.6 μm-3ML 표준셀 기술로 합성한 후 Compass상에서 모의실험을 통해 33MHz의 시스템 클럭에 대해 최대 지연시간 상황에서 실시간 동작을 확인하였다.

ABSTRACT

The recent audio CODEC(Coding/Decoding) algorithms are complex of several coding techniques, and can be divided into DSP tasks, controller tasks and mixed tasks. The traditional DSP processor has been designed for fast processing of DSP tasks only, but not for controller and mixed tasks. This paper presents a new architecture that achieves high throughput on both controller and mixed tasks of such algorithms while maintaining high performance for DSP tasks. The proposed processor, YSP-3, operates four algorithms while maintaining high performance for DSP tasks. The proposed processor, YSP-3, operates functional units (Multiplier, two ALUs, Load/Store Unit) in parallel via 4-issue super-scalar instruction structure. The performance evaluation of YSP-3 has been done through the implementation of the several DSP algorithms and the part of the AC-3 decoding algorithms.

* 연세대학교 전자공학과
** 연세대학교 의용공학과
論文番號:98107-0312
接受日字:1998年 3月 12日

I. 서 론

현재 미국과 유럽 표준안으로 채택된 AC-3[1]와 MPEG-2[2]와 같은 오디오 코덱 알고리듬은 높은 압축율을 얻기 위해 여러 가지 코딩 기법을 조합하여 사용하고 있다. 이들 알고리듬은 처리과정에서 요구하는 연산의 속성으로 다음과 같이 분할할 수 있다. 먼저 비트열(Bit Stream) 분석, 동작 모드 설정, 외부 시스템과의 통신 등과 같은 제어작업을 들 수 있다. 두번째로 필터링, 행렬 곱셈과 다양한 변환(Transform)등의 DSP 작업이 있다. 마지막으로 심리음향 모델을 이용한 마스킹 곡선 계산, 크기 인자(Scalefactor) 추출, 테이블 참조 등의 혼합 작업을 들 수 있다.

이중 DSP 작업은 데이터 어레이에 대해 곱셈 및 누산(MAC)을 반복하는 작업으로 특히 반복 작업이 규칙적이며 횟수도 미리 결정되어있다는 특징을 가지고 있다. 예를 들어 128 포인트 FIR 필터의 경우는 한 개의 출력을 얻기 위해서 128개 입력 데이터와 128개 필터 계수에 대해 128번의 MAC 연산을 수행한다.

한편 제어작업은 입력신호를 해석하고 시스템 제어를 행하며 외부 시스템 자원과의 통신을 수행하는 작업으로, 연산량은 적으나 조건 판별과 단순 연산을 반복적으로 수행한다. 이때 반복회수나 처리 과정이 실제 수행시 입력에 따라 결정되는 속성을 가지고 있다. 마지막으로 혼합 작업의 경우는 DSP 작업과 제어작업의 중간적인 속성을 지닌다. 즉, 연산의 복잡도와 조건 판별이 적절하게 혼합되어 있다.

오디오 압축 알고리듬을 처리 부하(processing load) 관점에서 살펴보면, 제어작업과 혼합작업이 절반 가량을 차지하고 DSP 작업이 나머지 절반을 차지한다[3]. 이러한 알고리듬을 실시간으로 구현하기 위한 시스템을 설계할때 기존의 DSP 프로세서를 사용하는 것은 적합하지 않다. 기존의 DSP 프로세서는 DSP 작업은 효과적으로 처리하지만, 제어작업과 혼합 작업은 성능이 매우 떨어지기 때문이다. 기존의 DSP 프로세서는 디지털 신호처리 알고리듬의 핵심이 되는 곱셈 및 누산을 고속으로 처리하기 위해 곱셈기, ALU, Shifter 등을 직렬로 연결하여 사용하고 있다. 이 구조를 통해 한 명령어 사이클에 한번의 MAC가 가능해져 고속의 처리가 가능하다. 그러나 이들은 MAC 이외의 연산에 대해서는 그들이 가지는 연산 장치들을 효율적으로 운용하지 못한다. 즉 MAC 이외의 명령어는 DSP 프로세서가 가지는 여러가지 연산기중 1개만을 사용할 수 있기 때문에 효율적인 처리가 불가능하다.

프로세서의 처리속도를 향상시키기 위하여 시스템 클럭을 높이는 방법을 사용할 수 있다. 예를 들면, 20MHz에 동작하는 프로세서를 40MHz에 동작하도록 하는 방법을 말한다. 그러나 이를 위해서는 더욱 높은 수준의 공정 기술을 필요로 하는 어려움이 있다. 즉, 1.0um 공정에서 생산된 20MHz 프로세서를 더욱 높은 주파수에서 동작시키기 위해서는 1.0um 미만의 공정기술로 만들어야 한다. 따라서 프로세서의 구조를 높은 처리능력을 가지도록 새롭게 구성하여 처리속도를 높이는 것이 바람직하다.

기존의 DSP 프로세서의 문제점을 구조적으로 개선하기 위해서 많은 연구들이 수행되어져 왔다[4][5][6]. 그러나 이러한 연구들은 제어 작업과 혼합작업에서 고성능을 가지도록 개선하였지만, 연산유닛(functional unit)의 활용도와 유연성(flexibility)이 낮은 문제점을 여전히 가지고 있다. 본 논문에서는 이 문제점을 해결하기 위해 먼저 기존 DSP 프로세서가 가지는 연산유닛을 병렬로 배치 한 후, 연산유닛들병렬로 배치한 연산 유닛을 곱셈기, 2개의 ALU, 그이 각각 한 개의 명령어를 동시에 처리하도록 하였다. 리고 메모리 제어 유닛으로 구성된 4개로 설정한 후, 각각이 1개의 명령어를 처리하도록 하였다. 여기에 파이프라인 명령어 처리를 수행하여 1 명령어 사이클(instruction cycle)에 4개의 명령어를 동시에 수행하도록 하였다.

이러한 구조를 이용하면 제어 작업과 혼합 작업에서는 4-이슈(issue) RISC 마이크로 프로세서와 같이 동작하여 기존의 DSP 프로세서보다 2배 이상 빠른 처리속도를 얻을 수 있다. 또한 DSP 작업에 대해서는 한 연산기의 결과를 다른 연산기의 입력으로 공급하여 MAC 연산을 처리하므로 기존의 DSP와 같은 성능을 얻을 수 있다. 이렇게 설계된 4-way 수퍼스칼라 DSP 프로세서를 YSP-3라 이름지었다.

이러한 구조의 프로세서에 VLIW(Very Long Instruction Word) 명령어 구조를 사용하는 것도 좋은 방법이다. 그러나 VLIW는 명령어 길이가 상당히 길어지며, 동시에 수행 가능한 명령어가 4개가 되지 못하면 나머지 명령어 영역은 사용하지 못하므로 명령어 코드의 효율성이 매우 낮아지는 문제점이 있다[7][8].

다른 접근 방법으로는 수퍼스칼라 명령어 구조(Superscalar Instruction Structure)를 들 수 있다. 이때 수퍼스칼라 명령어 구조는 동시에 수행할 명령어를 골라내는 이슈(Issue)과정이 요구되는 단점이 있다. 그러나 컴파일 할때 명령어의 순서를 미리 정리하고 순차적 이슈-순차적 종료(In-Order-Issue and In-Order-Completion) 방법으로 이슈를 하도록 하면 이슈과정이 간단해지고 비교적 적은 로직으로 구현이 가능하다[9].

II. 구 조

명령어 처리 과정은 크게 3단계로 분할할 수 있다. 명령어를 메모리로부터 읽어오는 과정(명령어 페치 과정), 명령어를 해석하는 과정(명령어 디코딩 과정), 그리고 해석된 명령어에 따라 수행하는 과정(실행 과정)이 그것이다. 이러한 명령어 처리 단계는 순차적으로 진행되어야 하며 각 단계는 최소한 1개의 시스템 사이클을 요구한다. 따라서 1개의 명령어를 처리하는데 최소한 3사이클이 소요된다.

이때 각각의 과정을 시간적으로 중첩(Overlap)시켜 파이프라인 처리하면 처리 속도를 향상 시킬 수 있다. 시간적으로 중첩되는 단위를 파이프라인 단계(stage)라 부르는데, 각각의 단계는 모두 같은 수행시간을 가지도록 설계된다. 만일 특정단계에서만 많은 시간을 요구하면 다른 단계에서는 그만큼 노는 시간이 많아지므로 비효율적이다. 즉 실행 단계에서 50ns를 요구하는데 디코딩 단계에서는 10ns만을 요구한다면 디코딩 단계는 40ns를 놀게되는 것이다.

이러한 관점에서 보면, 실제적으로는 명령어 페치와 명령어 디코딩과정은 비교적 빠른 시간에 수행이 가능하지만, 기존의 DSP와 같이 여러 연산 유닛이 직렬로 연결되는 경우에는 실행단계가 매우 길어진다. 그러나 서론에서 설명한 바와 같이 모든 연산 유닛을 병렬로 배치하면 전체적으로 통과해야할 로직의 길이가 짧아져 수행 시간이 줄어드는 장점을 얻게된다. 그리고 명령어 디코딩 과정의 경우 수퍼스칼라 명령어 구조를 사용하게 되면 동시에 수행할 명령어를 골라내는 명령어 이슈(Issue)과정이 요구된다. 이 과정을 수행하는 시간을 실행 유닛이 요구하는 시간과 같도록 맞추면 효율적인 시간, 자원활용이 가능해진다.

본 논문에서는 파이프라인을 다음과 같이 3단계로 분할하였다. 이때 각 단계는 모두 1 시스템 사이클 내에 수행되도록 한다.

(1) 명령어 페치(Pre-Fetch) 단계:4개의 명령어를 메모리로부터 읽어온다.

(2) 명령어 이슈/디코딩(Issue-Decoding) 단계:읽어온 4개의 명령어중 동시에 수행 가능한 명령어를 골라내고 이들을 해석한다.

(3) 실행 (Execute) 단계:이슈된 명령어를 실행한다.

이들 3개의 파이프라인 단계를 효율적으로 운영하기 위해 각각의 단계를 수행하는 명령어 페치 유닛(PFU:Pre-Fetch Unit), 이슈/디코딩 유닛(IDU:Issue-Decoding Unit), 그리고 실행 유닛(EU:Execution Unit)을 두었다. 그리고 효율적인 파이프라인 운영을 위해 하버드 구조를 사용하였다. 그림 1에 YPS-3의 전체 구조와 파이프라인 동작을 나타내었고, 표 1에 명령어 세트를 나타내었다.

1. 실행 유닛(Execution Unit)

앞서 설명한 바와 같이, 실행 유닛은 곱셈 유닛(MU:Multiplier Unit), ALU1, ALU2, 그리고 읽기/쓰기 유닛(LSU:Load/Store Unit)의 4개의 연산유닛을 두었다. 그림 2에 실행 유닛의 구조를 나타내었다.

1.1 곱셈 유닛과 ALU1

곱셈 유닛은 2개의 16-bit 입력을 받아 곱한 후 32-bit의 결과를 결과 버스(result bus)에 싣는다.

ALU1은 40-bit의 덧셈/뺄셈, 부호 변환, 절대값 변환 등의 산술연산과 AND, OR, XOR, NOT등의 논리연산을 행한다. 이러한 기본적인 연산 외에 ALU1은 몇 개의 특수한 연산을 수행한다. 먼저 40×16-bit 배럴 쉬프터를 내장하여 산술 쉬프트를 수행한다. 두번째로는 내장된 배럴 쉬프터를 활용하여 16비트의 입력으로부터 원하는 개수(최소 1비트에서 최대 16비트)만큼의 비트를 잘라내는 Bit-unpack 연산을 행한다. 이 연산이 필요한 이유는 오디오 코덱뿐만 아니라 대부분의 압축 알고리듬은 압축된 데이터를 비트열 형태로 만들기 때문에, 디코딩시에는 이 비트열을 해석하기 위해 적당한 개수의 비트를 잘라내어 읽어와야 하기 때문이다. 만일 이 과정을 특별한 모듈을 사용하지 않고 쉬프터만을 이용한다면 한번 읽는데

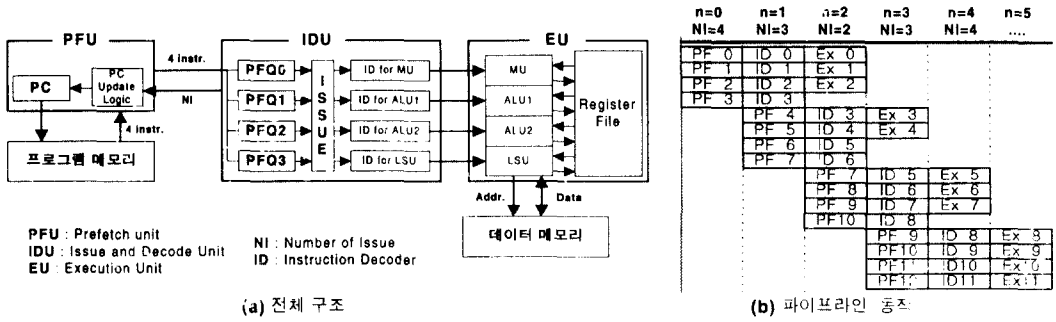


그림 1. 전체구조와 파이프라인 동작

표 1. 명령어 세트

명령어	동작	명령어	동작
AND dst, src1, src2	src1 and src2 → des	UNPACK dst(상수)	비트열 FIFO → dst
OR dst src1 src2	src1 or src2 → des	LDROM dst(주소)	ROM(주소) → dst
NOT dst src1 src2	not src → des	LDRAM dst(주소)	RAM(주소) → dst
ADD dst src1 src2	src1 + src2 → des	STORE src(주소)	src → RAM(주소)
SUB dst src1 src2	src1 - src2 → des	LOOP (주소)	(주소)까지 반복 실행
ABS dst src	abs (src) → des	CALL (주소)	(주소)의 서브루틴 콜
NEG dst src	- (src) → des	RTN	서브 루틴에서 [illegible]
RND dst src	round(src) → des	JUMP (주소)	(주소)로 점프
CMP src1 src2	set Status Flag if src1 ≥ src2	BRT (조건) (주소)	(조건)이 맞으면(주소)로 점프(2.2.2. 참조)
		BRN (조건) (주소)	
MULT dst src1 src2	src1 × src2 → dst	BRD (조건) (주소)	
MOVE dst src	src → dst	MOVEV dst(16bit 값)	(16bit 값) → dst
SHIFT dst src(상수)	shift(src) → dst	NOP	No Operation

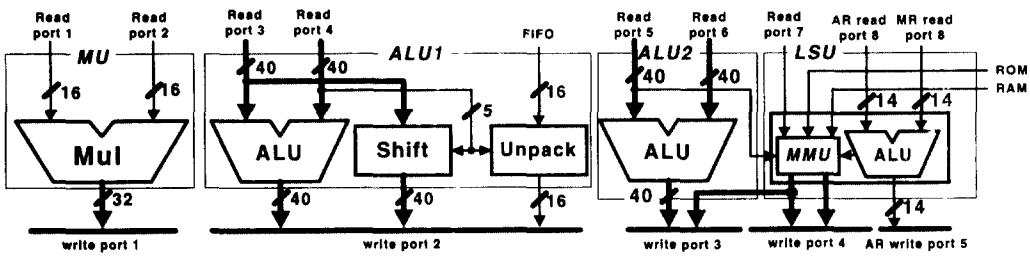


그림 2. 실행유닛의 내부구조

약 4개의 명령어가 요구된다. 따라서 많은 양의 비트를 읽어야 하는 코덱 응용에서는 1사이클에 원하는 비트수 만큼 읽어들일 수 있는 별도의 모듈을 이용하는 것이 바람직하다. 이 과정을 위해 내장된 쉬프터를 제어하며 입력 비트열을 관리하기 위한 Unpack 모듈을 두었다.

Unpack 모듈은 다음과 같이 동작한다. 먼저 입력 비트열을 내부에 16k-bit 입력 FIFO를 두어 저장해 놓는다. 그후 Unpack명령을 실행하면 제일 처음에 입력된 16비트를 Unpack 모듈로 읽어온다. 읽어온 16비트중 n ($1 \le n \le 16$)비트만을 읽어내고자 한다면 상위 n비트를 잘라 결과 버스에 실어주고, (16-n)개의 남은 비트를 다시 unpack 모듈의 내부 레지스터에 담아 놓는다. 이후에 새로이 m 비트를 읽어오라는 Unpack명령이 실행되면 이 레지스터에 들어 있는 값을 이용해 m비트를 결과버스에 실어준다. 만일 m이 (16-n)보다 크면 새로 16비트를 입력 FIFO로부터 읽어와서 모자른 비트를 채워 m비트를 결과버스에 싣고 나머지 (32-n-m) 비트를 다시 내부 레지스터에 담아놓는다.

마지막으로 ALU1의 특수한 기능은 반올림(Round) 연산이다. 이 연산은 RND 명령어에 의해서 수행된다. 32비트 레지스터 값을 16비트 레지스터에 저장하는 경우와 같이 목적지(destination) 레지스터의 비트폭이 근원(source) 레지스터의 비트폭 보다 작은 경우 목적지 레지스터 비트폭에 맞도록 반올림을 수행한다. 이때 반올림은 round-to-nearest-even 방법[10]을 사용하여, 고정 소수점 연산기를 가지는 DSP 프로세서의 최대 단점인 정확도 오차(precision error)를 통계적으로 최소화 시킬 수 있다.

1.2 읽기/쓰기 유닛(Load/Store Unit)

읽기/쓰기 유닛(LSU)는 모든 데이터 메모리 접근을 담당한다. 데이터 메모리는 내부 메모리와 외부 메모리로 구성되어 있다. 이중 내부 데이터 메모리는 16-bit RAM과 ROM, 32-bit의 RAM과 ROM의 4개로 구성되어 있고, 외부 데이터 메모리는 16-bit RAM으로 구성되어 있다. DSP 프로세서는 대량의 데이터 어레이에 대해 연산을 수행하는 특징을 가지고 있어, 레지스터에 대한 연산보다는 메모리에 대한 연산이 주를 이룬다. 따라서 가급적 오버헤드(Overhead)없이 메모리 값을 읽고 쓸 수 있어야만 고속의 처리가 가능하다. 이러한 이유로 대부분의 DSP 프로세서는 별도의 데이터 메모리용 주소 발생기를 내장하고 있으며, 다양한 DSP알고리듬을 수용할 수 있도록 선형(Linear), 모듈로(Modulo) 어드레싱, 비트리버스(Bit-Reverse) 모드등을 지원한다. 본 논문에서 설계된 읽기/쓰기 유닛도 이러한 사항을 만족하도록 내부에 주소 발생용 연산기를 내장하여 후 갱신(postmodify) 선형, 모듈로, 비트리버스 어드레싱 모드를 지원한다.

1.3 ALU2

ALU2는 ALU1과 동일한 40-bit ALU를 가지고 있어 산술연산과 논리연산을 수행한다. 여기에 내부 데이터 롬을 읽는 특수한 기능을 부여하였다. 많은 DSP 알고리듬이 입력 데이터와 미리 지정된 계수들과 곱셈및 누산(MAC)을 수행한다. 이때 입력 데이터는 RAM에 저장하고, 계수들은 미리 결정된 값이므로 ROM에 저장 시켜놓는다. 예를 들어 FIR이나 IIR 필터링의 경우 샘플링된 입력 데이터는 RAM에 저장되고, 필터 계수는 ROM에 저장된다. 따라서 MAC연산을 고속으로 행하기 위해서는 RAM과 ROM을 동시에 읽을 수 있어야 한다. 그러나 읽기/쓰기 유닛은 모든 데이터 메모리의 어드레스와 데이터 버스를 관리하고 있지만 1개의 결과 버스만을 가지고 있으므로 동시에 2개를 읽어 레지스터 파일에 전달할 수 없다. 따라서 다른 유닛의 결과 버스를 사용해야 하는데, 이를 위해 ALU2를 사용한다. 모든 데이터 메모리 접근은 읽기/쓰기 유닛이 담당한다.

모든 메모리 접근은 읽기/쓰기 유닛이 담당하지만, 읽기/쓰기 유닛이 데이터 RAM에 접근하기 위해 사용중이고 데이터 ROM을 또 읽어야 하는 경우에는 ALU2가 동작한다. 이때 ALU2는 레지스터 화일이나 명령어로 부터 데이터 ROM을 읽기위한 주소를 받아 읽기/쓰기 유닛에 넘겨준다. 읽기/쓰기 유닛은 그것을 받아 데이터 ROM을 읽어 그 결과를 다시 ALU2에게 넘겨주고, ALU2는 그 값을 받아 결과 버스에 싣는다. 그리고 ROM을 읽기 위한 명령어가 간접 어드레싱(Indirect Addressing)을 사용하여 주소 갱신을 요구하면 ALU2의 내부 ALU를 사용하여 수행한다.

모든 연산 유닛은 앞서 설명한 기능 이외에 명령어로부터 주어지는 상수값을 바로 결과버스에 실어 레지스터 파일에 저장할 수 있는 바이패스(By-Pass)기능을 공통적으로 가진다.

1.4 레지스터 파일(Register File)

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

배정도 레지스터는 2개의 단정도 레지스터로 사용될 수 있도록 설계하여, 1개의 32-bit값을 저장하거나 2개의 16-bit값을 저장할 수 있다. 이러한 구조를 사용하여 서브 워드 프로세싱(Sub-Word Processing)이 가능하다. 즉 1개의 데이터를 몇 개의 필드로 분할하여 각각이 독립적인 데이터를 의미하도록 하면 병렬로 배치된 여러개의 연산기가 이들을 동시에 처리할 수 있다. 또한 이들은 1개의 복소수 값을 가지도록 할 수 있으므로 FFT와 같은 복소수를 사용하는 응용에 유용하게 활용될 수 있다.

어드레스 레지스터와 어드레스변경 레지스터를 조합하여 간접어드레싱으로 메모리 접근을 하면 지정된 어드레스 레지스터에 저장되어 있는 값이 해당 메모리의 어드레스 버스에 실린다. 읽기/쓰기 유닛은 그 어드레스에 해당하는 메모리에 접근을 시작함과 동시에 함께 지정된 변경 레지스터의 값을 어드레스 레지스터값과 더하거나 뺀다. 메모리 접근을 완료하면 결과 버스에 메모리 값을 싣고(쓰기 경우에는 아무값도 싣지 않는다.) 갱신된 새로운 어드레스 값을 어드레스 레지스터에 되돌려 준다. 이를 통해 후 변경 간접 어드레싱(Post-modify Indirect Addressing)을 수행한다.

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

만일 연산 유닛의 입력이 입력으로 지정된 레지스터보다 많은 비트를 요구할 경우에는 부호 확장이 이루어진다. 예를 들면 ALU1의 입력으로 16-bit의 범용 레지스터가 선택되면 40-bit로 부호확장이 된다.

2. 페치 유닛과 이슈/디코딩 유닛

2.1 이슈/디코딩 유닛

수퍼스칼라 프로세서의 가장 큰 단점은 명령어 이슈과정이 요구된다는 점이다. 명령어 이슈는 프로그램 메모리의 일부 영역에 대해 윈도우를 취한후 그 명령어 윈도우(Instruction Window)내에 동시에 실행할 명령어를 찾아내는 작업이다. 이 작업은 명령어 상호간에 자원의 충돌(Resource Conflict), 데이터의 의존성(Data Dependency)이 발생하는가를 조사하는 것이다. 그러나 이 과정을 수행하기 위해서는 많은 실리콘 면적과 처리 시간을 요구하는 어려움이 있다.

명령어 이슈 방법에는 순차적 이슈(In-Order Issue)와 비순차적 이슈(Out-of-Order)의 2가지가 있다. 비순차적 이슈란 현재 명령어가 앞의 명령어와 충돌이나 의존성이 발생하더라도 그 다음 명령어들에 대해서도 계속 조사하여 가능한 한 많은 명령어를 골라내 실행을 한다. 따라서 명령어 실행 순서가 메모리에 저장된 순서와 달라지는 특징을 가지고 있으며 1사이클에 수행할 수 있는 명령어 수인 이슈율(Issue Rate)이 높아지는 장점을 가지고 있다. 그러나 그만큼 복잡하고 많은 로직을 요구한다. 반면 순차적 이슈는 어떤 명령어와 다음 명령어 사이에 충돌이나 의존성이 하나라도 존재하면 이슈를 중단한다. 따라서 실행 순서는 메모리에 저장된 순서와 동일하다. 이 방법은 이슈율은 낮지만 구현이 간단한 장점이 있다.

그러나 순차적 이슈라 하더라도 명령어를 프로그램 메모리에 싣기 전에 컴파일 과정에서 이러한 충돌이나 의존성이 최소화 되도록 명령어를 미리 재구성(re-arrange)해 놓으면 비순차적 이슈와 동일한 이슈율을 가지도록 할 수 있다. 즉 비순차적 이슈 과정이 수행하는 명령어 재정렬의 작업을 컴파일러가 대신하도록 하여 순차적 이슈만으로도 같은 성능을 낼 수 있는 것이다. 또한 모든 명령어는 동일한 처리 시간을 가지도록 설계되었으므로 순차적 이슈만 이루어진다면 순차적 종료(In-Order Completion)가 만족된다. 본 논문에서는 컴파일러와 순차적 이슈 방법을 통한 순차적 종료 방법을 사용함으로서 비교적 간단하게 고성능을 얻도록 하였다.

이슈/디코딩 유닛은 PFQ0-PFQ3에 대해 2개의 명령어가 동일한 자원을 요구하는 경우인 자원의 충돌(예를 들면 2개의 명령어가 곱셈기를 요구하는 경우)과 현재 명령어가 앞선 명령어의 처리 결과를 사용해야만 하는 경우인 데이터 의존성을 판단하여 동시에 수행 가능한 명령어를 골라낸다. 이슈/디코딩 유닛은 충돌이나 의존성이 하나라도 발생하면 바로 이슈를

중단하고 그때까지 이슈된 명령어를 각 연산 유닛용 명령어 해석기에 보낸다. 그림 1에 이와 같은 명령어의 흐름을 나타내었다.

명령어는 25-bit으로 구성되며 5개의 필드로 구분된다. MSB부터 (1) 명령어 코드 필드(Op-code field: 5-bit), (2) 목적지 레지스터 필드(Destination Register Field: 5-bit), (3) 메모리 어드레싱 모드 필드(2-bit), (4) 2번째 근원 레지스터 필드(2'nd Source Register Field: 5-bit), (5) 1번째 근원 레지스터 또는 주소레지스터와 변경레지스터쌍 필드(1'st Source Register Field/AR and MR pair field;8-bit)로 나뉜다.

YSP-3에서 자원이란 연산기와 메모리를 가리키는데, 명령어 코드 필드에 그 명령어가 어떤 명령어이며 어떤 연산기와 메모리를 사용할 것인지를 나타내고 있다. 따라서 자원의 충돌은 5-bit의 명령어 코드 필드로 판별한다. 데이터의 의존성은 두 명령어가 사용하는 레지스터에 대해 판별한다. 현재 명령어가 이전 명령어 결과가 저장되는 레지스터를 근원 레지스터로 사용하는 경우를 말한다. 이는 5-bit의 목적지 레지스터와 다른 명령어의 하위 13-bit인 1번째, 2번째 근원 레지스터 필드를 가지고 판별한다.

2.2 페치유닛(Pre-fetch Unit)

페치유닛은 내부에 프로그램 카운터(PC)를 가지고 있어 현재 읽어와야 할 명령어의 주소를 저장해 놓는다. 그리고 PC가 가리키는 번지부터 4개의 명령어를 읽어 이슈/디코딩 유닛으로 전달해 준다. 이슈/디코딩 유닛은 이들 명령어를 내부의 명령어 레지스터인 PFQ0-PFQ3에 저장한다. 순차적인 진행시의 n번째 사이클에 PC 값은 다음 식에 의해 얻어진다.

$$PC(0) = 0,$$
$$PC(n+1) = PC(n) + NI(n) \qquad \text{식 (1)}$$

여기서 NI(n)은 이슈/디코딩 유닛에서 주어지는 값으로 n 번째 사이클에서의 이슈된 명령어 개수이다. 프로그램의 흐름을 변경하는 명령어인 CALL, RTN (Return), JUMP, BR(Branch;conditional jump), 그리고 LOOP와 같은 명령어는 단독으로 이슈된다. 그리고 PC값을 명령어가 지정하는 값으로 대체하며, NI(n)은 4가 된다. 이후에 다시 순차적인 진행이 이루어지면 식 (1)에 따라 PC값이 변경된다.

프로그램 카운터를 갑작스럽게 변화시키는 명령어, 예를 들면 CALL, RTN, LOOP, BR, JUMP등은 단독으로 이슈되며, 페치 유닛은 루프 스택, 루프 카운터, PC 스택등 자신이 가지고 있는 모든 자원들을 활용하여 프로그램 흐름을 제어한다. 그리고 하드웨어적으로 LOOP 기능을 제공하여 별도의 오버헤드 없이 명령어 구간 반복을 수행할 수 있다.

파이프라인 프로세서는 조건 분기인 BR(Branch)명령어에 대해서 가장 큰 문제점을 가지고 있다. 파이프라인 프로세서는 명령어를 미리 읽어와야 하는데 조건 분기 명령어의 경우는 그 다음의 명령어가 어떤 것이 와야하는지 모르기 때문에 파이프라인 흐름이 잠시 중단될 수밖에 없다. 이를 해결하기 위해 예측 분기(branch prediction)라는 방법을 사용한다. 즉 분기가 실패할지 성공할지를 프로세서(페치 유닛)가 나름대로 예측하여 진행시켜 파이프라인의 흐름을 중단시키지 않도록 유지한다. 만일 예측이 실패하면 예측을 통해 페치하고 디코딩한 것들을 버리고 다시 제대로 된 명령어를 페치해 온다.

YSP-3는 성능향상을 위해 다음 3가지 모드의 분기 예측을 지원한다. 분기가 성공할 것으로 예측하는 BRT (Branch-taken), 분기가 실패할 것으로 예측하는 BRN (Branch-not-taken), 그리고 분기의 성공 여부에 관계없이 다음 명령어를 수행하면서 분기 조건의 결과를 기다리는 지연 분기인 BRD (Delayed- Branch)를 지원한다. BRT나 BRN은 예측이 실패하면 1사이클의 지연(Latency)이 발생하지만 BRD는 지연이 없다.

III. 동 작

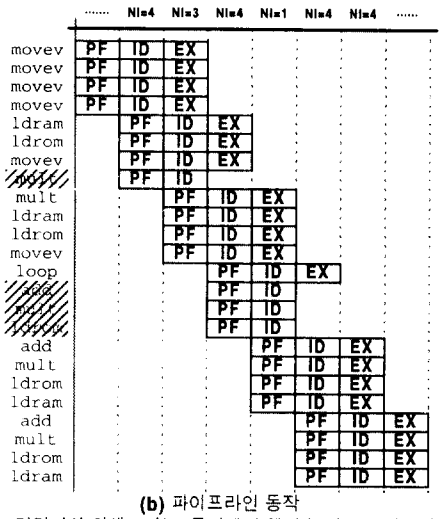
그림 3에 필터 계수 h(i)와 입력 샘플 x(n)과 콘볼루션을 행하여 출력 샘플 y(n)을 계산하는 64-tap FIR Filter 예제의 프로그램과 파이프라인 동작을 나타내었다. 각각의 연산 유닛은 다음과 같이 동작한다.

(1) 주소 레지스터 AR0는 데이터 RAM으로부터 입력 데이터 x(n)을 읽어 R0에 저장하기 위한 포인터로 사용하고(Line 1), AR1은 필터 계수 h(i)를 읽어 R1에 저장하기 위한 포인터로 사용한다(Line 2). 그리고 AR2는 출력 샘플 y(n)를 기록하기 위한 포인터로 사용한다(Line 3)

(2) AR0와 AR1은 변경 레지스터인 MR0의 값을 이용하여 후증가(Post-Increment) 된다(Line 5, 6, 9,

```
 1 movev  ar0, ^sample  ; ar0= x(n) ptr
 2 movev  ar1, ^coeff   ; ar1= h(i) ptr
 3 movev  ar2, ^output  ; ar2= y(k) ptr
 4 movev  mr0, #1       ; mr0= post-increment
                        ; value

 5 ldram  r0, (ar0, mr0) ; r0= x(0)
 6 ldrom  r1, (ar1, mr0) ; r1= h(0)
 7 movev  acc1, #0       ; clear acc1

 8 mult   acc0, r0, r1   ; acc0=x(0)*h(0)
 9 ldram  r0, (ar0, mr0) ; r0= x(1)
10 ldrom  r1, (ar1, mr0) ; r1= h(1)
11 movev  cntr, #62      ; repeat 62 times

12 loop   :end            ; following 4 instr.

13  add  acc1, acc1, acc0 ; acc1 = acc1+acc0(n)
14  mult acc0, r0, r1     ; acc0 = x(n)*h(i)
15  ldrom  r1, (ar1, mr0) ; r1 = h(i++)
16 end:
17  ldram  r0, (ar0, mr0) ; r0 = x(n++)

18 add  acc1, acc1, acc0 ; acc1=acc1+acc0(62)
19 mult acc0, r0, r1     ; acc0=x(63)*h(63)

20 add  acc1, acc1, acc0 ; acc1=acc1+acc0(63)

21 shift  acc1, acc1, #8 ; bit-width adjustment
                         ; 40 bit -> 32 bit

22 rnd    r0, acc1       ; round(32bit->16bit)

23 store  r0, (ar2, mr0) ; y(k++)=r0
```

(a) 64-tap FIR Filter 프로그램



(b) 파이프라인 동작

(명령어의 회색표시는 동시에 수행될수 없음을 나타냄)

그림 3. 64-tap FIR Filter의 프로그램과 파이프라인 동작

10, 15, 17, 23). LDRAM은 읽기/쓰기 유닛에서 수행되고(Line 5, 9, 17), LDROM은 ALU2에서 수행된다(Line 6, 10, 16).

(3) 곱셈 유닛은 R0와 R1을 곱한 후 40-bit ACC0에 저장한다(Line 8, 14, 19).

(4) ALU1은 ACC0와 ACC1을 더해 ACC1에 저장한다. 즉 (3)에서 수행한 곱셈과 함께 MAC (Multiply and Accumulate)연산을 수행한다(Line 13, 18, 20).

이와 같은 파이프라인 처리 방법으로 64번의 MAC 연산을 67사이클 만에 수행한다

VI. 성능 평가

설계된 YSP-3는 VHDL로 기술하여 0.6um-3ML CMOS 기술의 표준 셀(standard cell)로 합성하였다. YSP-3의 전체 게이트 수는 43,993.3 게이트이고 33 MHz에서 동작함을 확인하였다. 따라서 이슈율이 최대치인 4가 되면 132MIPS의 성능을 보이며, 최소치인 1이 되더라도 33MIPS를 확보할 수 있다.

성능 평가를 위해 먼저 YSP-3의 사이클 단위 시뮬레이터를 C 언어를 이용하여 작성하였다. 이 시뮬레이터를 이용하여 몇 개의 일반적인 DSP 알고리듬과 AC-3 디코딩 과정을 프로그래밍한 후 모의실험을 수행하였다.

먼저 AC-3 디코딩 알고리듬에 대한 모의 실험 결과를 표 2에 나타내었다. 표에 나타낸 결과치는 5.1 채널 449kbps AC-3 비트스트림의 1프레임에 대해 수행한 결과이다. 이중 비트 할당과정은 AC-3 알고리듬 중 가장 복잡한 혼합 작업(Mixed Task)으로 구성된 것으로 한 프레임에 대해 평균 2.561 ms에 수행된다. 역시 혼합 작업의 대표적인 예인 커플링 좌표 곱셈을 포함한 가수부 디코딩 과정은 3.327ms에 수행된다.

이와 같이 제어 작업과 혼합작업에서 고성능을 보이는 것은 앞서 설명한 바와 같이 높은 연산 유닛 활용도와 유연성의 장점을 보여준다.

4가지의 DSP 알고리듬에 대한 성능은 표 3에 나타내었다. 표에 알 수 있듯이 DSP 작업에서도 좋은 성능을 나타낸다. Radix-2 1024-point 복소 고속푸리에 변환(complex valued FFT)과 실수 고속 푸리에 변환(real valued FFT)은 각각 1.094ms와 657.2us에 수행된다[11]. 이때 각 FFT 과정 수행시 각각의 버터플라이 연산시 마다 반올림 연산을 수행하였다.

표 2. 449kbps AC-3 알고리듬의 부분별 수행시간

AC-3 알고리듬	수행 시간
Exponent Decoding	380.8 us
Bit Allocation Decoding	2561 ms
Mantissa Decoding	3.327 ms
Channel Decoupling	142.9 us

표 3. 일반적인 DSP 알고리듬 처리 성능

디지털 신호 처리 알고리듬	수행 시간
100-tap FIR filter	3.303 us
Two 8×8 matrix product	15.70 us
Radix-2 1024 point complex FFT	1.094 us
Radix-2 1024 point real FFT	657.2 us

V. 결 론

본 논문에서는 DSP 작업뿐만 아니라 제어작업과 혼합 작업에 대해서도 좋은 성능을 가지는 4-way 수퍼스칼라 프로세서의 구조를 제안하고 구현하였다. 제안된 프로세서는 4개의 연산유닛을 병렬로 배치한 후 수퍼스칼라 명령어 구조를 도입하였다. YSP-3는 각각의 연산 유닛을 독립적으로 사용하므로 단순한 연산으로 구성되어 있는 제어작업과 혼합작업을 고속으로 처리할 수 있다. 또한 4개의 연산유닛을 파이프라인 동작 시켜 MAC 연산을 효율적으로 수행하므로 DSP 작업에도 좋은 성능을 보인다.

YSP-3는 연산기 활용에 있어서 높은 유연성을 제공하므로, 명령어를 충돌이나 의존성이 없도록 배치하여 프로그램 작성을 하면 어떠한 응용 분야에도 좋은 성능을 발휘할 수 있다. 비단 오디오 코덱 응용뿐만 아니라 멀티미디어와 통신 응용분야에서도 ASIC 설계에 사용하기 적합하므로 저가격으로 좋은 성능의 시스템을 설계하는데 용이하다. 마지막으로 고품질 데이터를 얻도록 하기 위해 내부 연산기의 비트수를 확장하는 작업과 프로그램 작성을 용이하게 할 수 있도록 C 컴파일러 개발 등을 추후 연구 목표로 하고 있다.

참 고 문 헌

1. ISO/IEC JTC1/SC29/WG11 No. 703 "Generic Coding of Moving Pictures and Associated Audio-CD 13818-3(Part 3. MPEG-Au- dio)" Mar., 1994.
2. Advanced Television Systems Committee (ATSC) Standard Doc. A/52, "Digital Audio Compression Standard(AC-3)", Nov., 1994.
3. Steve Vernon "Design and implementation of AC-3 coders", IEEE Transactions 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 scheduling 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, IEEE, June, 1983.
9. Mike Johnson, "Superscalar Microprocessor Design", Prentice Hall, Inc., pp. 17-24, 1991.
10. Istael Koren, Computer 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, pp. 849-863, VOL. ASSP-35, No. 6, June 1987.

김 준 석(Joon-Seok Kim) 정회원

1972년 10월 23일생
1995년:연세대학교 전자공학과(공학사)
1997년:연세대학교 전자공학과(공학석사)
1997년~현재:연세대학교 전자공학과 박사과정
※주관심분야:ASIP, 오디오 신호처리 등
e-mail:monia@radar.yousei.ac.kr

유 선 국(Sun-Kook Yoo) 정회원

1959년 1월 8일생
1981년:연세대학교 전기공학과(공학사)
1985년:연세대학교 의용공학과(공학석사)
1989년:연세대학교 의용공학과(공학박사)
1989년~1995년:순천향대학교 조교수
1995년~현재:연세대학교 의용공학과 조교수
※주관심분야:생체신호처리, 고속정보 시스템, VLSI 신호처리 등

박 성 욱(Sung-Wook Park) 정회원

1971년 3월 27일생
1993년:연세대학교 전자공학과(공학사)
1995년:연세대학교 전자공학과(공학석사)
1995년~현재:연세대학교 전자공학과 박사과정
※주관심분야:오디오 신호처리, VLSI 신호처리, 설계 자동화

정 남 훈(Nam-Hoon Jeong) 정회원

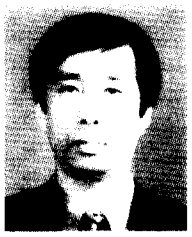
1973년 12월 7일생
1995년:연세대학교 전자공학과(공학사)
1997년:연세대학교 전자공학과(공학석사)
1997년~현재:연세대학교 전자공학과 박사과정
※주관심분야:디지털 신호처리, 오디오 신호처리, VLSI 신호처리 등

고 우 석(Woo-Suk Ko) 정회원

1973년 1월 2일생
1996년:연세대학교 전자공학과(공학사)
1998년:연세대학교 전자공학과(공학석사)
1998년~현재:연세대학교 전자공학과 박사과정
※주관심분야:오디오 신호처리, VLSI 신호처리 등

이 근 섭(Keun-Sup Lee) 정회원

1975년 2월 1일생
1997년:연세대학교 전자공학과(공학사)
1997년~현재:연세대학교 전자공학과 석사과정
※주관심분야:오디오 신호처리, VLSI 신호처리 등



윤 대 희(Dae-Hee Youn) 정회원

1951년 5월 25일생
1977년:연세대학교 전기공학과(공학사)
1979년:미국 Kansas 주립대학교 전기공학과(공학석사)
1982년:미국 Kansas 주립대학교 전기공학과(공학박사)
1982년~1985년:미국 Kansas 주립대학교 조교수
1985년~현재:연세대학교 전자공학과 정교수
※주관심분야:적응 신호처리, 음성/오디오 신호처리, 레이다/소나 신호처리, VLSI 신호처리