論文99-36C-2-1

RISC 기반 DSP 프로세서 아키텍쳐의 성능 평가

(A Performance Evaluation of a RISC-Based Digital Signal Processor Architecture)

姜 志 浪 *, 李 鍾 馥 **, 成 元 鎔 *

(Jiyang Kang, Jongbok Lee, and Wonyong Sung)

요 약

디지털 신호처리용 응용 프로그램의 복잡도가 증가하면서, 효율적인 컴파일러를 지원하는 DSP 프로세서 구조의 필요성이 증대되고 있다. 많은 범용 레지스터와 직교적(orthogonal)인 명령어 집합을 가지는 RISC 프로세서 구조에 메모리 오퍼랜드, 전용 어드레스 계산 유닛, 단일 사이클 MAC 명령어, zero-overhead 하드웨어 루프 등 DSP 프로세서의 구조적 특징을 가하여 효율적인 컴파일러를 가지는 고성능의 RISC 기반 DSP를 구현할 수 있다. 본 논문에서는 이 네 가지 DSP 아키텍쳐 구성 요소를 지원하는 코드변환기를 개발하고, 이를 이용하여 각각의 DSP 아키텍쳐 구성 요소들을 보완하였을 때 성능에 미치는 영향을 정량적으로 평가하였다. 성능 평가 실험에는 C 언어로 작성된 7개의 DSP 벤치마크 프로그램과 QCELP 음성 부호화기를 이용하였으며, 평가 결과를 RISC 프로세서뿐만 아니라 Texas Instruments 사의 TMS320C3x, TMS320C54x, TMS320C5x DSP 프로세서와 비교하였다.

Abstract

As the complexity of DSP (Digital Signal Processing) applications increases, the need for new architectures supporting efficient high-level language compilers also grows. By combining several DSP processor specific features, such as single cycle MAC (Multiply-and-ACcumulate), direct memory access, automatic address generation, and hardware looping, with a RISC core having many general purpose registers and orthogonal instructions, a high-performance and compiler-friendly RISC-based DSP processors can be designed. In this study, we develop a code-converter that can exploit these DSP architectural features by post-processing compiler-generated assembly code, and evaluate the performance effects of each feature using seven DSP-kernel benchmarks and a QCELP vocoder program. Finally, we also compare the performances with several existing DSP processors, such as TMS320C3x, TMS320C54x, and TMS320C5x.

I. 서 론

최근 몇 년 사이에 디지털 TV, 화상 회의, ISDN (Integrated Services Digital Network), ATM (Asynchronous Transfer Mode)과 같은 멀티미디어 서비스가 보편화됨에 따라, 방대한 연산량을 요구하는 디지털 신호 처리 (DSP: Digital Signal Processing) 알고리즘을 실시간으로 처리할 수 있는 전용 시스템 (embedded system)의 설계 및 구현 문제가 중요하게 대두되고 있다. 전통적으로 이와 같은 응용 (application)에 많이 사용되어 온 프로그래머블 DSP 프로세서는 반복적인 수치 연산 처리에 적합하

* 正會員, 서울大學校 電氣工學部
(School of Electrical Engineering, Seoul National University)
** 正會員, LG 半導體 株式會社
(LG Semicon Co., Ltd.)
接受日字:1998年7月20日, 수정완료일:1999年1月4日

도록 높은 대역폭 (bandwidth)의 메모리 아키텍처, 전용 어드레스 계산 유닛 (dedicated address generation unit), 고속 MAC (Multiply-and-Accumulate) 연산 지원과 zero-overhead 하드웨어 루프 등 범용 프로세서 (GPP: General-Purpose Processor)와 뚜렷이 구별되는 특징을 가진 아키텍처를 가지고 있다 [1] [2] [3]. 한편, 최근 들어 높은 동작 주파수를 가지는 고성능 범용 RISC 프로세서들이 등장함에 따라, 고급언어 컴파일러의 지원이 용이한 RISC 아키텍처 기반의 마이크로 콘트롤러 (MCU: Micro Controller Unit)를 이용한 전용 시스템의 설계 또한 점차 증가하고 있는 추세이다 [4]. 그러나, 순수한 RISC 아키텍처 기반의 MCU들은 메모리 쓰기와 읽기, 그리고 수식 연산이 많은 DSP 응용에는 미흡한 면이 많다. 이에 따라, 최근에는 DSP 프로세서와 RISC MCU의 각각의 아키텍처로부터 장점을 적절히 취하여 빠른 연산 능력과 시스템의 저비용화를 동시에 목표로 하는 멀티미디어 시스템을 위한 전용 프로세서 설계의 새로운 경향이 등장하고 있다 [5].

이와 같이 MCU와 DSP 프로세서를 복합화하는 구조를 가지는 마이크로프로세서를 설계하는 경우, 비용 효율적 (cost-effective)인 설계를 위해서는 어드레스 계산 유닛 (AGU: Address Generation Unit), MAC 명령어의 지원과 같은 특정 아키텍처 요소의 추가가 실제 성능에 미치는 영향의 정량적인 평가가 반드시 필요하다. 이와 관련된 연구를 살펴보면, 명령어 집합 아키텍처 (ISA: Instruction Set Architecture) 수준에서 아키텍처와 실제 성능에 대한 관계 분석이 1970년대부터 최근까지 계속되고 있으나, 대부분 범용 프로세서의 경우에 한정되어 진행되었다 [6] [7] [8] [9]. 이중에서 특히 DEC의 VAX와 IBM RS/6000 아키텍처와 같이 DSP 프로세서의 기능적 특징인 복잡한 어드레싱 모드와 하드웨어 루프를 지원하는 범용 프로세서에 대한 성능 평가가 일부 시도되기도 하였으나, 벤치마킹 (benchmarking) 과정에서 실제 DSP 응용이 아닌 SPEC 등 범용 프로세서에 적합한 벤치마크를 사용하여 제한된 결과를 얻었을 뿐이다 [10] [11]. 한편, Aachen 대학에서는 DSP 알고리즘에 기반한 DSPstone 벤치마크를 제안하여 기존의 DSP 프로세서의 성능 평가를 시도한 바 있으나, 개별적인 아키텍처 구성 요소와 성능 향상의 관계에 대한 연구는 진행된 바 없다 [12] [13].

본 논문에서는 RISC 프로세서 구조를 기반으로 한 프로세서 모델에 대하여, DSP 프로세서 아키텍처의 특징을 각각 대표하는 네 가지 구체적인 아키텍처 구성 요소들을 보완하였을 때 성능에 미치는 영향을 정량적으로 평가하고자 하며, 이를 위해 이 네 가지 아키텍처 구성 요소들을 지원하는 코드변환기를 개발하였다. 성능 평가에는 실제 DSP 응용 프로그램을 컴파일하여 코드변환기로 후처리(post-processing)한 코드를 이용하였으며, 기존의 DSP 프로세서의 성능 비교도 수행하였다. 본 논문의 구성은 다음과 같다. 2 장에서는 범용 RISC 프로세서와 뚜렷이 구별되는 DSP 프로세서의 네 가지 대표적인 아키텍처 구성 요소와 이를 지원하는 코드변환기에 대해서 설명하였으며, 3 장에서는 성능 평가를 위한 DSP 커널 벤치마크의 종류와 성능 평가 환경을 설명하였다. 4 장에서는 이 아키텍처 구성요소들을 기본 RISC 프로세서 모델에 각각 추가하였을 때의 성능 향상을 7가지 DSP 커널 벤치마크를 통해 분석하였으며, 5 장에서는 QCELP vocoder 프로그램을 이용하여 전체 응용 프로그램의 성능 향상을 평가하였다. 6 장에서는 기존 DSP 프로세서와의 성능 비교를 보였으며, 마지막으로 7장에서 결론을 맺는다.

Ⅱ. RISC 기반 DSP 프로세서 아키텍처

범용 RISC 프로세서와 구별되는 DSP 프로세서의 주요 특징으로는 표 1에 요약한 바와 같이 높은 메모리 대역폭을 지원하기 위한 메모리 아키텍처, 배열 연산과 FFT 등 DSP 알고리즘을 효율적으로 지원하기 위한 다양한 어드레싱 모드 및 전용 어드레스 계산 유닛, 고속의 MAC 연산 지원과 zero-overhead 하드웨어 루프 등이 있다.

이들 네 가지의 아키텍처 특징들이 전체 성능 (performance)에 미치는 영향을 평가하기 위해서, 기본적인 RISC 프로세서에 각각의 특징을 개별적으로 추가한 아키텍처 모델들의 성능을 각각 측정하였다. 주어진 프로세서에서 특정한 프로그램을 수행시켰을 때 소요되는 시간은 수행 사이클 수와 머신 사이클 타임 (machine cycle time)의 곱으로 구할 수 있다. 본 논문에서는 하나의 기본 프로세서 모델에 기반을 두고 약간의 구성 요소들을 변형해가면서 실험하는 방식을 취했으므로, 동작 주파수 또는 머신 사이클 타임과 같

은 다른 요소의 변화는 최소화할 수 있다고 가정하였으며, 이에 따라 RISC 기반 DSP 프로세서의 성능을 수행 사이클 수를 측정하여 평가하였다.

표 1. DSP 프로세서의 주요 특징
Table 1. Main features of DSP processors.

특징	아키텍처 구성 요소	예
높은 메모리 bandwidth	하버드 아키텍처 메모리 오퍼랜드	많은 데이터를 처리해야하는 신호처리 알고리즘에 적합
다양한 어드레싱 모드	전용 어드레스 계산 유닛	Indirect addressing with post-increment/decrement, Bit-reversed addressing, Circular addressing 효율적인 배열 처리
빠른 Multiply-Accumulate	MAC 명령어 병렬 연산 지원	대부분 신호처리 알고리즘에 많이 사용 (예: 필터링, FFT 등)
효율적인 루프 지원	Zero-overhead 하드웨어 루프	반복적인 알고리즘의 효율적 처리

설정한 기본 하드웨어 모델은 그림 1에 보인 바와 같이 ALU와 하드웨어 곱셈기를 하나씩 가지는 레지스터 중심 (register-based) 구조의 범용 32 비트 RISC 콘트롤러이며, 기존의 SPARClet [14] 및 SPARClite [15] 프로세서와도 유사한 아키텍처를 가지고 있다. 이 아키텍처는 SPARC V8 [16]을 기반으로 하여 두 개의 어드레스 계산 유닛과 블럭 반복 (block repeat)을 위한 레지스터 등의 하드웨어 자원을 추가로 가지며, 레지스터 윈도우는 채용하지 않았다. 데이터 버스는 칩 내부 (on-chip)의 SRAM으로부터 1 사이클에 2 개의 32 비트 데이터를 읽어 올 수 있으며, 명령어 버스는 32 비트의 데이터 폭을 가지는 것으로 가정하였다. 이 프로세서 모델이 사용하는 명령어들은 기본적으로 SPARC 어셈블리어를 그대로 사용했으며, 이밖에 기타 하드웨어의 변동에 따라 새로운 명령어와 어드레싱 모드를 추가할 때에는 Texas Instruments 사의 TMS320C5x DSP의 명령어를 차용하여 사용하였다 [17].

벤치마크 프로그램들은 SPARC V8 아키텍처의 코드로 컴파일된 후에 코드 변환 소프트웨어(code converter)에 의해 RISC 기반 DSP의 코드로 변환된다. 코드 변환 소프트웨어는 주어진 SPARC 어셈블리 코드로부터 얻어낸 제어 흐름 그래프 (control flow graph)에 데이터 플로우 분석 (data flow analysis)을 수행한 뒤, RISC 기반 DSP 모델의 아키텍처 특성에 적합하도록 소스 코드를 변환해낸다.

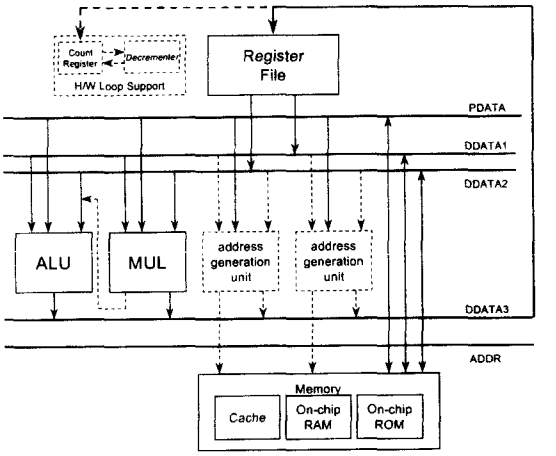


그림 1. RISC 기반 DSP 아키텍처 모델
Fig. 1. Proposed RISC-based DSP architecture model.

본 실험에서 사용된 각 아키텍처 구성 요소의 특징과, 이를 지원하기 위한 코드변환기의 동작은 아래와 같다.

1. 메모리 오퍼랜드

많은 데이터 처리를 필요로 하는 디지털 신호처리 알고리즘의 효과적인 수행을 위해서는 많은 데이터를 메모리로부터 주고받을 수 있는 능력이 필요하다. DSP 프로세서는 이를 위해서 특수한 메모리 구조를 가지고 있으며, 여러 개의 내부 버스를 가지는 하버드 아키텍처 (Harvard architecture), 다중 메모리 뱅크와 한 명령어 사이클에 접근 가능한 칩 내부의 메모리는 그 대표적인 예이다. 이와 같은 특징들로 인해 DSP 프로세서는 load-store 중심 구조의 일반적인 범용 RISC 프로세서와는 달리 메모리로부터 오퍼랜드 (operand)를 직접 가져올 수 있는 구조를 가지고 있다.

본 연구에서는 DSP 프로세서의 높은 대역폭의 메모리 구조를 명령어 아키텍처 수준에서 모델링하기 위해 일반적인 ALU 명령어와 승산 명령어에 메모리 오퍼랜드를 사용할 수 있도록 하였다. 이에 따라 연산에 필요한 데이터를 메모리에서 레지스터로 가져오는 명령어와 그 데이터를 이용하여 연산하는 명령어를 별도로 사용하지 않아도 되므로, 그림 2에 보는 바와 같이 2 ~ 3 개의 명령어를 한 개로 나타내는 이점을 가진

다.

코드 변환기는 다음과 같은 방법으로 메모리 오퍼랜드를 지원한다. 소스 프로그램을 처음부터 읽어나가다가 ALU 명령어가 나오면 그 오퍼랜드 값을 정의(define)하는 명령어를 찾아서 그것이 메모리로부터의 load 라면 메모리 오퍼랜드를 사용하는 ALU 명령어로의 변환을 시도한다. 그 결과, 더 이상 사용되지 않게 된 load 명령어는 제거된다. 이 과정은 명령어 비트 폭의 제한과 같은 다른 제약조건들도 함께 고려하여 수행된다.

한편, 메모리 오퍼랜드를 사용하는 것은 이와 같이 프로그램의 수행 사이클 수를 줄일 수 있으나 메모리의 느린 처리 속도로 인해 프로세서의 동작 주파수를 낮출 가능성이 있기 때문에 오히려 전체 성능에 나쁜 영향을 미칠 수도 있다. 따라서, 하드웨어 구현 시에는 고속의 칩 내부 메모리와 소프트웨어 제어 가능한 캐쉬 (software-controllable cache)의 지원, 그리고 효율적인 메모리 파이프라인 (pipeline) 등 고성능 메모리 시스템의 설계가 요구된다.

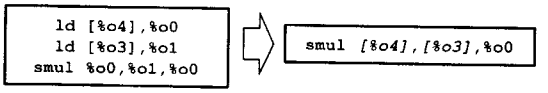


그림 2. 메모리 오퍼랜드 사용 예
Fig. 2. An example of memory-operands.

2. 어드레스 계산 유닛

수치 연산을 최대 속도로 수행하면서 동시에 데이터 이동에 필요한 어드레스 계산을 병렬적으로 수행하기 위해 일반적으로 DSP 프로세서는 데이터 패스에 독립적인 어드레스 계산 유닛을 가지고 있다. 어드레스 계산 유닛은 신호처리 알고리즘의 효율적인 수행에 적합하도록 원형 데이터 버퍼의 구현에 적합한 circular 어드레싱과 FFT를 위한 bit-reversed 어드레싱 등 다양한 어드레싱 모드를 지원한다. 이중 가장 대표적인 것은 간접 어드레싱에 이은 포스트 증가 (indirect addressing with post-increment) 모드이며, 배열과 같이 메모리에 연속적으로 저장되어 있는 데이터에 대한 반복 연산에 적합하다. 포스트 증가 간접 어드레싱 모드를 지원하는 범용 프로세서인 VAX에서 실험한 결과에 따르면, SPEC 벤치마크를 사용한 경우 사용 빈도가 3 % 미만에 그치는 것으로 나타났다 [10]. 한편, 역시 포스트 증가 간접 어드레싱 모드를 지원하는 IBM RS/6000의 실험 결과에서는 비교적 DSP 알고리즘과 유사한 특성을 가지는 Livermore Loop에서 18 %의 성능 향상을 보였고 나머지 SPEC과 Linpack 벤치마크에서는 4 ~ 8 %의 성능 향상을 보였다 [11].

제안하는 RISC 기반 DSP 모델에서는 포스트 증가 간접 어드레싱 모드와 포스트 감소 간접 어드레싱 모드 (indirect addressing with post-decrement)를 지원하는 어드레스 계산 유닛을 2개까지 동시에 사용할 수 있는 것으로 가정했다. 그림 3에 보인 바와 같이, 포스트 증가 간접 어드레싱 모드를 사용하면 별도의 어드레스 계산 명령어가 필요 없으므로 2 ~ 3개의 명령어를 한 개의 명령어로 수행할 수 있다.

이 어드레싱 모드들을 지원하기 위해서, 코드 변환기는 아래와 같은 과정을 수행한다. 먼저, 프로그램에서 간접 어드레싱을 하는 명령어를 하나씩 찾는다. 그 명령어의 후속 명령어들 중에서, 주소를 읽어오는 레지스터 값을 처음으로 갱신하는 명령어를 찾아 "register += immediate" 형태를 갖추고 있는지 검사한다. 이와 같은 형태이고 그 immediate 상수의 값이 기본 데이터의 크기와 일치하면 발견한 간접 어드레싱 모드를 포스트 증가/감소 간접 어드레싱 모드로 변환하고, 필요 없는 레지스터 갱신 명령어는 제거한다.

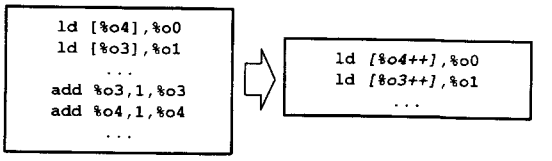


그림 3. 포스트 증가 간접 어드레싱 모드 사용 예
Fig. 3. An example of indirect addressing with post-increment.

3. MAC 명령어 지원

범용 RISC 프로세서와 구별되는 DSP 프로세서의 특징 중에 가장 대표적인 것은 하드웨어 승산기를 사용하여 한 사이클만에 수행되는 MAC (Multiply-and-Accumulate) 명령어이다. MAC 연산은 디지털 필터링, 상관값 (correlation), FFT의 계산 등 많은 신호처리 알고리즘의 계산에 매우 유용함이 이미 잘 알려져 있다. 특히 대부분의 알고리즘에서 MAC 명령어는 루프 커널과 같이 전체 수행 시간에서 차지하는 비중이 큰 부분에서 실행되므로 그 중요도가 매우 크

다. 제안하는 RISC RISC DSP기반 DSP 모델에서도 그림 4에서 보인 것처럼 본래 SPARC 아키텍쳐에서 지원되지 않는 MAC 명령어를 추가하여 실험하였다.

MAC 명령어를 지원하기 위해, 코드 변환기는 곱셈의 결과 값을 다른 레지스터에 누산(accumulation)하는 명령어 패턴을 찾아서 하나의 MAC 명령어로 치환하는 과정을 수행한다.

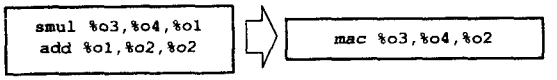


그림 4. MAC 명령어 사용 예
Fig. 4. An example of MAC instruction usage.

4. 하드웨어 루프 지원

신호처리 알고리즘은 흔히 커널 (kernel)이라고도 불리는 작은 크기의 코드 블럭을 반복적으로 수행하는 경우가 많으므로, DSP 프로세서들은 효과적인 루프 처리를 위해 하드웨어 루프 또는 zero-overhead 루프라는 기능을 지원한다. 하드웨어 루프란 *for*-루프, *while*-루프와 같은 루프 구문을 루프 카운터 (loop counter) 변수 값의 갱신과 종료 검사 등의 오버헤드 없이 수행할 수 있도록 지원하는 기능이며, 약간의 부가적인 하드웨어를 필요로 한다. 하드웨어 루프 기능은 다시 한 개의 명령어만을 반복하는 단일 명령어 하드웨어 루프 (= single repeat)와 여러 명령어로 이루어진 코드 블럭을 반복하는 복수 명령어 하드웨어 루프 (= block repeat)의 두 가지로 나뉜다.

제안하는 RISC 기반 DSP 모델에서는 이 두 가지 하드웨어 루프를 모두 지원하며 중첩(nestable)될 수 있다고 가정하였다. 중첩될 수 있는 루프의 개수는 제한하지 않았으나, 실제 실험에서 2 단계 이상 중첩되는 경우는 없었다. 그림 5에 Texas Instruments 사의 TMS320C5x의 니모닉 (mnemonic)을 그대로 사용하여 하드웨어 루프 반복 명령어 (*rptb*, *rpt*)로 간단한 FIR 필터의 내부 루프를 구현한 예를 보였다. 이 예에서 보듯이, 복수 명령어 하드웨어 루프를 사용한 경우 루프의 반복횟수를 M으로 놓았을 때 3*M+1 개의 루프 오버헤드가 2 개로 줄어든 것을 알 수 있다. 즉, 한 iteration에서 수행되는 명령어의 수를 N이라고 하면, N*M+1 개의 명령어를 (N-3)*M+2 개의 명령어로 수행하게 된다. 이때, 반복되는 명령어의 수가 작을수록, 그리고 루프의 반복횟수가 커질수록 하드웨어 루프로 인한 이득이 커진다. 만일 한 iteration에서 수행할 명령어의 수를 루프 오버헤드를 제외하고 1 개로 줄일 수 있다면, 단일 명령어 하드웨어 루프를 어드레스 계산 유닛 및 MAC 명령어와 함께 사용하여 최적의 성능을 낼 수 있다.

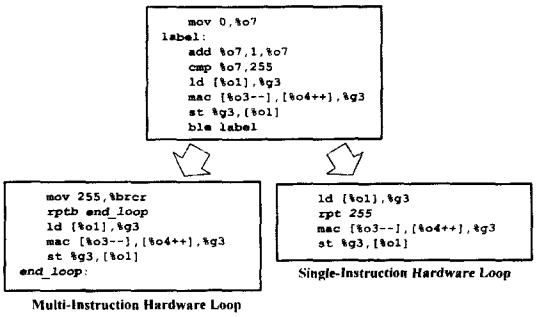


그림 5. 하드웨어 루프 사용 예
Fig. 5. Examples of single- and multi-instruction hardware loops.

코드 변환기는 C 소스에서 for-루프에 해당하는 구조를 찾아서 해당하는 하드웨어 루프를 사용하는 코드로 치환한다. DSP 알고리즘에서 사용되는 루프 구조는 거의 이와 같이 상수번 반복 수행하는 간단한 루프이므로 쉽게 발견할 수 있다.

III. 성능 평가 환경

II 장에서 요약한 네 가지 아키텍쳐 구성 요소가 전체 성능에 미치는 영향을 정량적으로 평가하기 위해, 제안한 RISC 기반 DSP 프로세서에서 벤치마크 프로그램이 수행될 때의 사이클 수를 시뮬레이션을 통하여 측정하였다. RISC 기반 DSP 프로세서는 멀티미디어 전용 시스템을 목표로 하고 있으므로, 일반적으로 많이 사용하는 SPEC 벤치마크 대신 실제 DSP 응용에 가까운 벤치마크 프로그램을 선택하여 실험하였다. 벤치마크 프로그램은 크게 DSP 커널 벤치마크와 응용 프로그램 벤치마크의 두 가지로 구분된다. DSP 커널 벤치마크로는 표 2에 보인 것과 같이 FIR 필터, IIR 필터, FFT, IDCT 등 가장 많이 사용되는 대표적인 7 가지 DSP 알고리즘을 사용하였다. 보다 실제 응용에 가까운 평가를 위해 응용 프로그램 벤치마크로 Qualcomm 사에 의해 개발된 analysis- by-synthesis 방식의 음성 부호화 알고리즘인 TIA IS-96A QCELP 가변전송율 음성부호화기를 사용하였다 [18]. 본 실험에서는 1 프레임 (20ms, 160 샘

플) 길이의 전형적인 음성 신호를 부호화 하여 성능을 측정하였다.

표 2. DSP 커널 벤치마크 프로그램
Table 2. Description of DSP-kernels benchmark programs.

프로그램 이름	알고리즘 기술
mult_10_10	10x10 행렬 곱셈
fir_256_64	256 탭 FIR 필터, 64 샘플
iir_4_64	4 번 cascade된 biquad IIR 필터, 64 샘플
lmsfir_32_64	32 탭 adaptive LMS FIR 필터, 64 샘플
latnrm_32_64	32 차 lattice 필터, 64 샘플
fft_1024	complex FFT, 1024 샘플, radix-2 decimation-in-time, in-place
IDCT	8-point IDCT, Chen's algorithm

성능 평가 실험은 다음과 같은 과정을 거쳐 수행되었다. 먼저, C 언어로 코딩한 벤치마크 프로그램을 컴파일하여 SPARC V8 버전의 어셈블리 코드를 얻었다. C 컴파일러로는 GNU *gcc* 컴파일러 버전 2.7.2를 사용하였으며, 레벨 3 최적화 옵션을 주어 컴파일 하였다. 이 SPARC 어셈블리 코드를 본 실험을 위해 개발한 코드 변환 소프트웨어 (code converter)를 사용하여 RISC 기반 DSP 아키텍처의 어셈블리 코드로 변환하였다. 이 변환된 어셈블리 소스에 RISC 기반 DSP 모델 시뮬레이터를 사용하여 수행 사이클 수를 추정한다. RISC 기반 DSP 모델 시뮬레이터는 빠른 측정을 위해서 Shadow 프로그램으로부터 뽑아낸 명령어 트레이스 (trace)를 함께 사용한다 [19]. 설명한 벤치마킹 환경의 개요를 그림 6에 보였다. 프로그램의 데이터는 모두 칩 내부의 메모리에 있다고 가정하였다.

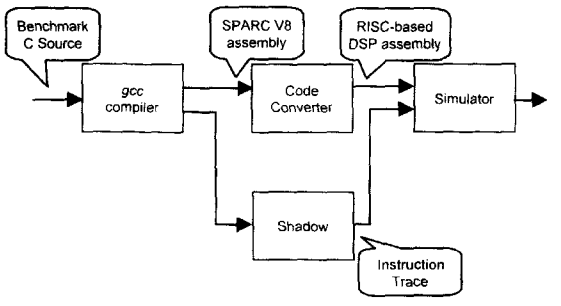


그림 6. 벤치마킹 환경 개요
Fig. 6. Benchmarking environments.

Ⅳ. DSP 커널 벤치마크의 성능 평가

먼저 DSP 커널 벤치마크 대해서, DSP 아키텍처 구성 요소를 첨가하지 않은 기본 RISC 프로세서의 경우와 메모리 오퍼랜드, 어드레스 계산 유닛, 1 사이클 MAC 명령어, 하드웨어 루프를 각각 지원하는 경우, 그리고 이 모든 요소를 모두 갖춘 RISC 기반 DSP 프로세서의 수행 사이클 수를 비교하여 그림 7과 표 3에 각각 보였다. 그림 7의 그래프에서 막대 그래프는 7 개의 DSP 커널 벤치마크 프로그램의 수행 사이클 수의 기하 평균값을 나타내며, 꺾은 선 그래프는 기본 RISC 프로세서에 대한 성능비를 나타낸다.

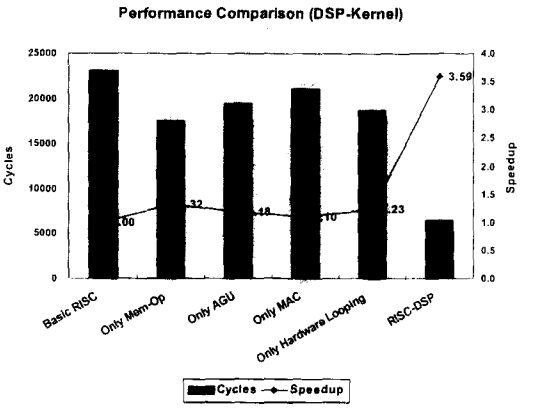


그림 7. RISC 기반 DSP 프로세서의 성능 비교(DSP 커널 벤치마크)
Fig. 7. Performance comparison of RISC-based DSP (DSP-kernel benchmarks).

표 3. RISC 기반 DSP의 벤치마크 실험 결과
Table 3. Benchmarking results of RISC-based DSP processor.

벤치마크		기본 RISC 모델 (사이클 수)	성능 향상비				
			메모리 오퍼랜드	어드레스 계산 유닛	1 사이클 MAC	하드웨어 루프	모든 DSP 아키텍처 구성요소 지원
DSP 커널	fft_1024	158830	29 %	15 %	7 %	10 %	104 %
	fir_256_64	180800	37 %	22 %	10 %	37 %	958 %
	iir_4_64	9856	31 %	32 %	8 %	8 %	148 %
	lmsfir_32_64	44032	31 %	23 %	10 %	38 %	739 %
	matrix_10_10	11980	33 %	9 %	9 %	32 %	265 %
	latnrm_32_64	58432	18 %	23 %	9 %	28 %	171 %
	IDCT	399	49 %	8 %	15 %	14 %	74 %
	기하 평균	-	32 %	18 %	10 %	23 %	259 %
QCELP		1361235	14 %	17 %	8 %	23 %	126 %

기본 RISC 프로세서 모델은 ALU와 하드웨어 승산기를 가지고 있으나, MAC 명령어와 어드레스 계산 유닛, 하드웨어 루프, 메모리 오퍼랜드를 지원하지 않는다. DSP 커널 벤치마크의 결과에서, 개별적으로 적용한 DSP 아키텍쳐 구성 요소들 중에서는 메모리 오퍼랜드를 지원하는 경우가 23073 사이클에서 17442 사이클로 약 32 %의 가장 두드러진 성능 향상을 보였다. 이것은 DSP 커널 벤치마크에서 메모리 오퍼랜드를 적용할 수 있는 경우가 다른 아키텍쳐 구성요소에 비해서 상대적으로 더 많았기 때문이다.

메모리 오퍼랜드만 적용한 경우, lattice 필터와 IDCT를 제외한 나머지 벤치마크들은 비슷한 성능향상비(speedup)를 보이는 것을 표 3에서 볼 수 있다. 특히 lattice 필터 프로그램에서 성능비가 제일 작고 IDCT에서 제일 크게 나타나는 것은, 메모리 오퍼랜드를 사용한 경우의 성능비가 가장 안쪽 루프 커널에서 ALU 연산이 차지하는 비율과 밀접한 관련이 있기 때문이다. 실험에서 사용한 lattice 필터 프로그램의 경우에 가장 안쪽 루프에 데이터 복사(copy)와 같이 ALU를 사용하지 않는 코드가 상당한 비율을 차지한 반면, IDCT 프로그램에서는 ALU 연산이 대부분을 차지했다. 한편, DSP 커널 벤치마크에 포스트 증가 간접 어드레싱 모드를 적용한 경우에는 각 벤치마크간의 성능비의 분포가 다양하게 나타나는 것을 표 3에서 볼 수 있다. IIR 필터의 경우 '*p++'와 같이 포인터 연산을 사용하여 연속적으로 메모리에 접근하는 C 코드를 사용하여 많은 성능 향상이 있었으나, Chen의 IDCT 알고리즘의 경우에는 연속적인 메모리 접근 자체가 거의 나타나지 않아 포스트 증가 간접 어드레싱 모드로 인한 이득이 거의 없었다. 하드웨어 루프 기능만을 지원했을 경우에는 가장 안쪽 루프의 크기에 따라 성능비가 크게 두 그룹으로 나뉘는 것을 볼 수 있다. 즉, FIR 필터와 같이 단일 명령어 하드웨어 루프를 사용할 수 있을 정도로 루프 크기가 작아서 두드러진 성능 향상을 보이는 벤치마크들과 biquad IIR 필터와 같이 루프 크기가 크고 저조한 성능 향상을 보이는 벤치마크들이 확연히 구분되었다.

그림 8에는 네 가지 DSP 아키텍쳐 구성 요소를 모두 적용했을 경우 RISC 기반 DSP 프로세서의 성능비를 각 벤치마크별로 나타내었다. 각각의 벤치마크 알고리즘에 대하여, RISC 기반 DSP 모델의 성능비는 매우 편차가 심한 것을 볼 수 있다. 특히, FIR 필터와 LMS adaptive FIR 필터와 같이 단일 명령어 하드웨어 루프 기능을 충분히 사용할 수 있는 경우에는 매우 높은 성능비 (FIR: 10.58, LMS-FIR: 8.39)를 나타냈으나, Chen의 IDCT 알고리즘과 같이 불연속적인 메모리 접근과 반복 횟수가 작은 루프의 예에서는 낮은 성능비 (1.74) 값을 보였다.

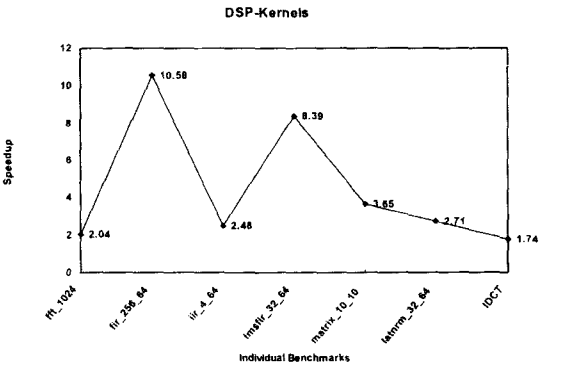


그림 8. 각 벤치마크별 성능향상비 (DSP 커널 벤치마크)

Fig. 8. Speedup of each benchmark (DSP-kernel benchmarks).

네 가지 모든 DSP 아키텍쳐 구성 요소를 지원한 경우의 성능비에 대해서는 다음과 같이 예측할 수 있다. 기본 RISC 프로세서 모델에서의 사이클 수를 T, 메모리 오퍼랜드를 지원했을 때 감소되는 사이클 수를 A라고 하면, 메모리 오퍼랜드를 지원하는 경우의 성능비 x는 식 (1)과 같다.

$$x = \frac{T}{T-A} \tag{1}$$

마찬가지로, 어드레스 계산 유닛, 1 사이클 MAC 명령어, 하드웨어 루프를 지원하는 경우 감소되는 사이클 수를 각각 B, C, D라고 하고, 이 때의 성능비를 각각 y, z, w라고 하면, 네 가지 DSP 구성요소를 모두 지원하는 경우의 성능비는 아래의 식 (2)와 같이 계산된다.

$$Performance_{Combined\ Features} = \frac{T}{T-(A+B+C+D)} = \frac{1}{-3+\frac{1}{x}+\frac{1}{y}+\frac{1}{z}+\frac{1}{w}} \tag{2}$$

위 식에 각 DSP 아키텍쳐 구성 요소를 하나씩 적용했을 경우의 성능비를 대입하여 계산하면 3.10

(210%의 성능 향상비)을 얻을 수 있다. 즉, 개별적인 기능에 의한 성능 향상비는 비교적 작게 보이지만, 이들의 복합 효과는 매우 커질 수 있다. 이 때, 실제 코드 변환기는 네 가지 DSP 아키텍처 구성요소를 지원하도록 프로그램 코드를 변환할 뿐만 아니라, 간단한 핍홀 최적화 (peephole optimization) 과정을 수행하여 성능을 더욱 향상시킨다.

```
        *output = 0;
        for (j = 0; j < NTAPS; j++)
        *output += (*data--) * (*coef++);
```

(a) FIR 필터 커널의 C 코드

```
    st    %g0,[%o1]      ; *output = 0
    mov   0,%o7          ; j=0; initialize loop index
LM00001:
    ld    [%o3],%g2      ; load data
    add   %o7,1,%o7      ; j++; loop index++
    ld    [%o4],%i0      ; load coef
    cmp   %o7,255        ; j > 255? check if loop ends
    ld    [%o1],%g3      ; load *output
    smul  %g2,%i0,%g2    ; mul
    add   %g3,%g2,%g3    ; acc
    st    %g3,[%o1]      ; store *output
    add   %o4,4,%o4      ; coef++
    add   %o3,-4,%o3     ; data--
    ble   LM00001        ; branch
```

(b) 코드변환기 입력 어셈블리 코드

```
    st    %g0,[%o1]      ; *output = 0
    mov   0,%o7          ; j=0; initialize loop index
      mov 255, %brcr     ; initialize loop counter
      rptb el00          ; multi-instruction hardware loop
LM00001:
    ld    [%o1],%g3      ; load *output
      mac [%o3--],[%o4++],%g3  ; %g3 += (*data--) * (*coef++)
      st  %g3,[%o1]      ; store *output
el00:
```

(c) 코드 변환 과정중의 어셈블리 코드 (핍홀 최적화 직전)

```
    st    %g0,[%o1]      ; *output = 0
    mov   0,%o7          ; j=0; initialize loop index
    ld    [%o1],%g3      ; load *output
    rpt   255            ; single-instruction hardware loop
    mac   [%o3--],[%o4++],%g3  ; %g3 += (*data--) * (*coef++)
    st    %g3,[%o1]      ; store *output
```

(d) 핍홀 최적화를 거친 최종 코드변환기 출력 코드

그림 9. FIR 필터의 코드 변환 예
Fig. 9. Code conversion example: FIR filter.

즉, 그림 9의 FIR 필터 예제에서 보이는 바와 같이 루프내의 불필요한 load-store 쌍을 루프 밖으로 꺼내는 과정을 수행한다. 그 결과, 본 실험에서 사용한 FIR 필터, LMS adaptive FIR 필터와 같이 커널 코드의 크기가 작은 벤치마크는 커널 코드를 포스트 증가 간접 어드레싱 모드를 사용하는 단 1개의 MAC 명령어로 압축하여 하드웨어 루프로 반복 수행할 수 있었으며, 그 결과 매우 높은 성능 향상비를 얻을 수 있었다. DSP 커널 벤치마크의 실험 결과를 종합해 보면, 7가지 벤치마크의 성능 향상비의 기하평균값은 3.59를 나타내었다. DSP 커널 벤치마크의 나머지 네 가지 알고리즘 (FFT, IIR 필터, 행렬 곱셈, lattice 필터)은 2.04에서 3.65에 이르는 다양한 성능비 값을 나타냈다.

V. 응용 프로그램 수준의 성능 평가

이번에는 QCELP 벤치마크에 대해서, 각 경우의 RISC 기반 DSP 프로세서의 수행 사이클 수를 비교하여 그림 10과 표 3에 보였다. 그림 10에서도 역시 막대 그래프는 수행 사이클 수를 나타내며, 꺾은 선 그래프는 기본 RISC 프로세서를 기준으로 한 성능비를 나타낸다. 이 그림에서, 어드레스 계산 유닛, 1 사이클 MAC 명령어, 하드웨어 루프를 각각 적용한 경우에는 DSP 커널 벤치마크 (그림 7)에서와 거의 같은 각각 17 %, 8 %, 23 %의 성능 향상비를 보였다. 반면에, 메모리 오퍼랜드를 지원하는 경우의 성능 향상은 14 %에 그쳤다. 한편, 모든 DSP 아키텍처 구성요소를 지원한 경우에는 126 %의 성능 향상을 얻었는데, 이 값은 역시 앞의 식 (2)를 이용하여 계산한 113 %보다는 약간 크지만, DSP 커널 벤치마크 실험 결과의 259 %에 크게 못 미치는 것을 알 수 있다.

QCELP 벤치마크에서 메모리 오퍼랜드를 지원하는 경우의 성능 향상이 다른 경우에 비해 저조하게 나타난 것은 메모리 오퍼랜드를 사용하는 ALU 명령어의 비율이 DSP 커널 벤치마크의 경우보다 훨씬 작기 때문이다. 즉, DSP 커널 벤치마크의 경우에 네 가지 DSP 아키텍처 구성 요소를 모두 갖춘 모델에서 메모리 오퍼랜드를 사용하는 ALU 명령어가 전체 명령어의 54 %를 차지했었으나, QCELP 벤치마크는 31 %에 그쳤을 뿐이다. 그 주된 이유는 규모가 작고 간단한 DSP 커널 벤치마크와는 달리 QCELP 벤치마크에서는 많은 전역 심볼 (global symbol)을 참조하는 코

드가 필요했기 때문이다. 테이블 참조 또는 배정도 (double precision) 상수의 참조와 같이 전역 심볼을 읽거나 쓰려면, 32 비트의 데이터 어드레스가 필요하다. 그런데, RISC 기반 DSP 프로세서의 명령어 길이 역시 32 비트이므로 한 번에 데이터 어드레스를 로드 (load)하지 못하고 대신 어드레스의 상위 22 비트와 하위 10 비트로 나누어 두 번에 읽어야 한다. 이 때, 심볼 어드레스의 상위 22 비트를 읽어오는 과정에서 *gcc* 컴파일러는 특정한 레지스터만을 계속해서 사용하려는 경향을 보였다. 따라서 메모리 오퍼랜드를 적용할 수 있는 코드들이 그 레지스터를 서로 사용하려다가 충돌이 일어났고, 그 결과 DSP 커널 벤치마크의 경우보다 낮은 성능 향상비를 보였다.

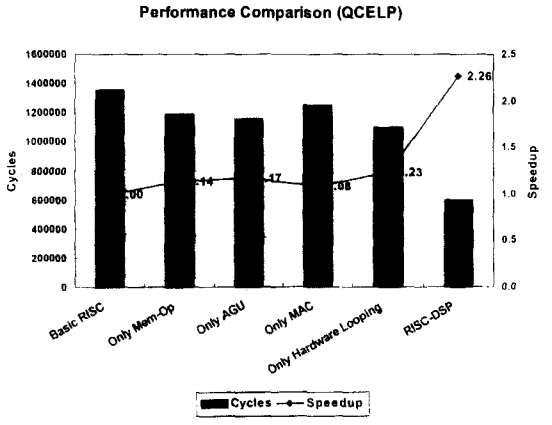


그림 10. RISC 기반 DSP 프로세서의 성능 비교 (QCELP 벤치마크)

Fig. 10. Performance comparison of RISC-based DSP (QCELP benchmark).

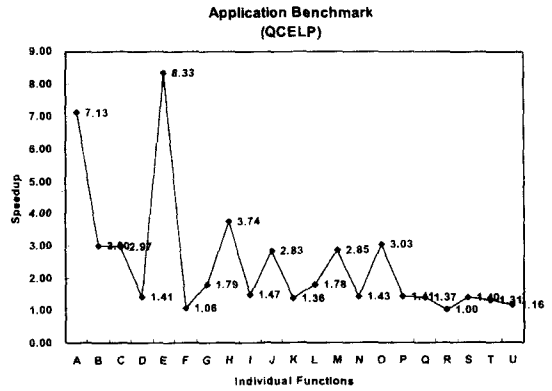


그림 11. 각 함수별 성능향상비 (QCELP 벤치마크)

Fig. 11. Speedup of each function (QCELP benchmark).

한편, 그림 11에는 QCELP 벤치마크에 대해서 RISC 기반 DSP 모델의 성능비 값을 각 함수별로 평균 수행 사이클 수에 따라 A부터 U까지 나타내었다. 역시 RISC 기반 DSP 프로세서는 각 함수의 특징에 따라 성능비 값의 변화가 심한 것을 관찰할 수 있다. QCELP 벤치마크에서 성능비가 높게 나오는 함수들은 상관 계수 (autocorrelation) 계산 (그림 11의 'E'), 컨볼루션 (convolution) 연산 ('A', 'O'), 필터링 ('B', 'C', 'M'), 피치 (pitch) 검색 ('H') 및 코드북 (codebook) 검색 ('J')을 수행하는 함수들이었으며, 대부분 많은 데이터에 대해 반복적인 루프 연산을 수행하는 전형적인 DSP 알고리즘의 특징을 보였다. 반면, 낮은 성능비 값을 나타내는 함수들은 양자화 (quantization) (그림 11의 'U'), 프레임 비트 패킹 (frame bit packing) ('T')에 관련되는 함수들로, 비교에 이은 *if* 문, *switch* 문이 그 수행 시간의 대부분을 차지했다. 이와 같이 전자에 해당하는 함수들 (이하 'DSP 함수군(群)')은 DSP 구조에 적합한 연산을 주로 수행하며, 후자에 해당하는 함수들 (이하 'MCU 함수군')은 MCU 구조에 적합한 연산을 수행한다고 개념적으로 분류할 수 있다. 본 연구에서 가정한 RISC 기반 DSP 모델은 DSP 함수군에서는 2 이상의 성능비 값을 기록하며 DSP 커널 벤치마크와 비슷한 정도의 성능 향상을 보였으나, MCU 함수군에서는 거의 성능 향상을 보이지 못했다. DSP 커널 벤치마크와 QCELP 벤치마크에서 RISC 기반 DSP 모델의 성능비가 두 배 이상 차이 나는 것은, Amdahl의 법칙에 따라 설명할 수 있다. 이에 따르면 프로그램 전체의 성능비는 수행 시간에서 성능비가 큰 부분과 성능 향상이 없는 부분이 각각 차지하는 비율에 따라 제약되며, 식으로는 아래 (3)식과 같이 나타낼 수 있다 [10].

$$Speedup_{overall} = \frac{Execution\ Time_{old}}{Execution\ Time_{new}} = \frac{1}{(1 - Fraction_{enhanced}) + \frac{Fraction_{enhanced}}{Speedup_{enhanced}}} \quad (3)$$

QCELP 벤치마크에서는 (3)식의 $Fraction_{enhanced}$, 즉 성능비가 큰 부분이 차지하는 비율을 77.3 %로 추정할 수 있었다. QCELP 벤치마크에서 비록 DSP 함수군에서 높은 성능 향상이 있더라도 MCU 함수군의 수행 시간에는 거의 변함이 없으므로 프로그램 전체의 성능비는 DSP 커널 벤치마크의 경우의 절반 정도에 그치게 되는 것으로 위 식에 따라 설명할 수 있다. 그

러나 실제 각각의 함수는 그 내부에서 반복적인 루프 연산과 *if* 문, *switch* 문을 모두 처리하는 경우가 대부분이므로, QCELP 벤치마크의 모든 함수를 이 두 가지 함수군으로 명확히 분류하기는 어렵다.

VI. 기존 DSP 프로세서와의 비교

본 연구에서 제안된 RISC 기반 DSP 프로세서의 성능과 하드웨어 비용을 기존의 DSP 프로세서와 비교하여 보았다. 먼저 현재 가장 널리 사용되고 있는 DSP 프로세서들인 Texas Instruments 사의 TMS320C5x, TMS320C54x[20], TMS320C3x[21]를 비교 대상으로 하여 앞 절에서 사용한 벤치마크로 비교 실험하였다. 또한, DSPstone의 FIR 벤치마크를 이용하여 그 밖의 DSP 프로세서들과도 성능을 비교해 보았다.

1. 아키텍처 구성요소의 비교

RISC 기반 DSP 프로세서 아키텍처상의 구성요소를 TMS320C5x, TMS320C54x, TMS320C3x DSP 프로세서와 비교하여 표 4에 보였다. RISC 기반 DSP 프로세서는 32 비트의 명령어폭을 가지고 한 개씩의 승산기와 ALU, 두 개의 AGU를 가지는 등, TMS320C30 프로세서와 비슷한 규모의 하드웨어를 가지고 있는 것을 알 수 있다. 그러나 범용 레지스터와 어드레스 레지스터를 각각 8 개씩 가지는 TMS320C30과는 달리, RISC 기반 DSP는 별도의 어드레스 레지스터 없이 32 개의 범용 레지스터를 가지며 부동 소수점 연산을 지원하지 않는 차이점이 있다.

표 4. 하드웨어 자원 비교
Table 4. Comparison of hardware resources.

	승산기 (갯수)	ALU (갯수)	AGU (갯수)	메모리 오퍼랜드	범용 레지스터 (갯수)	명령어 비트폭 (bits/instr.)	내부 메모리 대역폭 (program+data) (bits/cycle)	기타
TI 'C54x	1	2	2	0	2 ACC + 8 AR	16/32/48	16+32	ACC-based
TI 'C5x	1	1	1	0	1 ACC + 8 AR	16/32	16+16	ACC-based
TI 'C3x	1	1	2	0	8 + 8 AR	32	32+64	Register-based
RISC 기반 DSP	1	1	2	0	32	32	32+64	Register-based

2. 컴파일러를 이용한 성능 비교

성능 비교 실험에는 앞에서와 같이 7 가지 DSP 커널 벤치마크와 QCELP 벤치마크를 사용하였다. 이때, 실험에 사용한 TI DSP 프로세서들이 가지고 있는 내부 메모리는 각각 수 킬로 워드씩으로 제한되어 QCELP 프로그램 전체를 넣을 수 없었기 때문에, QCELP에서 가장 많은 계산량을 차지하는 코드북 검색에 해당하는 서브루틴만을 대상으로 실험하여 페널티가 많은 외부 메모리 접근을 피하도록 하였다. 실행 코드를 얻기 위해서 RISC 기반 DSP 프로세서는 *gcc* 컴파일러와 코드 변환기를 이용하였으며, 기타 TI DSP 프로세서들은 TI C 컴파일러에 최고의 최적화 옵션을 주어 사용하였다. 부동소수점 DSP 프로세서인 C3x에 대해서는 부동소수점으로 수정된 코드를 사용하였다.

표 5. 기존의 DSP 프로세서와의 성능 비교
Table 5. Performance comparison with other DSP processors.

벤치마크		사이클 수 (RISC 기반 DSP의 사이클 수로 normalize한 사이클 수)			
		RISC 기반 DSP	'C3x	'C5x	'C54x
DSP 커널	fft_1024	77933	121084 (1.55)	391540 (5.02)	240313 (3.08)
	fir_256_64	17088	66313 (3.88)	83394 (4.88)	82816 (4.85)
	iir_4_64	3968	7691 (1.94)	16512 (4.16)	6080 (1.53)
	lmsfir_32_64	5248	15883 (3.03)	23744 (4.52)	22018 (4.20)
	matrix_10_10	3280	11262 (3.43)	8930 (2.72)	8630 (2.63)
	latnrm_32_64	21568	30219 (1.40)	78528 (3.64)	57090 (2.65)
	IDCT	229	207 (0.90)	538 (2.35)	317 (1.38)
	총합 (기하평균)	129814	252609 (2.05)	603186 (3.77)	417264 (2.65)
QCELP		262504	337816 (1.29)	506720 (1.93)	499680 (1.90)

실험 결과, 표 5에서 보듯이 RISC 기반 DSP 프로세서는 DSP 커널 벤치마크의 경우 C3x DSP의 2.1배, C54x의 2.7배, C5x의 3.8배에 해당하는 좋은 성능을 보였다. 한편, QCELP 벤치마크의 실험 결과에서는 C3x DSP의 1.3배, C54x와 C5x의 1.9배의 성능을 보였다. RISC 기반 DSP 프로세서와 비교했을 때, C3x의 코드는 특히 병렬 MAC 연산(예: MPYF3

|| ADDF3)을 잘 활용하지 못하고 있었기 때문에 덜 좋은 성능을 보였다. 한편, C5x의 경우에는 다른 하드웨어 자원으로의 연결이 제한적인 승산기를 컴파일러가 효과적으로 사용하지 못하는 점과 비효율적인 어드레싱 모드로 인한 오버헤드 등의 원인으로 가장 나쁜 성능을 보였다. C54x와 C5x의 경우에는 범용 레지스터가 부족한 점 역시 효율적이지 못한 코드를 만들어 내는 원인으로 작용하였다.

3. DSPstone 벤치마크를 사용한 성능 평가

본 RISC 기반 DSP의 성능을 다른 DSP 프로세서의 성능과 비교하기 위해, DSPstone의 FIR 벤치마크를 이용하여 실험하여 그 결과를 아래 표 6에 보였다.

표 6. DSPstone 벤치마킹 결과

Table 6. DSPstone benchmarking result.

Processor	ADI 2183 32MHz	Moto 56001 81MHz	NEC 77111 75MHz	TI C51 100MHz	ADI 2105L 60MHz	TI C40 60MHz	TI C40 60MHz	RISC-DSP 75MHz
Compiler	ADI 5.1	Moto 1.11	NEC 1.0	TI 6.5	ADI 2.0	TI 4.5	Tartan 2.01	Code Converter
t_r [μs]	0.38	0.62	0.27	0.50	0.35	0.7	0.7	
t_g [μs]	3.37/3.67	4.0/4.0	1.56/0.28	2.48/2.48	1.12/0.9	2.83	2.73/0.73	0.69
c_g (c_r)	175/191(20)	324(50)	117/21(20)	124(25)	67/54(21)	85(21)	82/22(21)	52(52)
p_g (p_r)	21/23(6)	21(8)	18/7(6)	24(8)	11/10(7)	11(9)	13/9(9)	10(10)
d_g (d_r)	33/33(33)	33(33)	33/33(33)	33(33)	33/33(33)	33(33)	33/33(33)	33(33)
m_g (m_r)	54/56(39)	54(41)	51/40(39)	57(41)	44/43(40)	44(42)	46/42(42)	43(43)

t - 실행시간; c - 사이클 수; p - 프로그램 메모리; d - 데이터 메모리; m - p와 d의 합
아래 첨자 g는 컴파일되었음을 의미하며, 아래첨자 r은 manual 어셈블리 코드를 의미함

여기서 RISC 기반 DSP 프로세서는 모든 아키텍처 구성요소를 지원했을 때의 성능을 나타내며, 동작 주파수는 98년 0.35μm 삼성 CMOS 스탠다드 셀 라이브러리[22]를 사용한 구현을 가정하여 75 MHz로 계산하였다. 비교를 위해 사용된 다른 프로세서의 성능은 [13]에 발표된 실험 결과를 현재까지 발표된 같은 패밀리의 가장 빠른 클럭 주파수로 각각 보정하여 나타낸 것이다. 표 6에서, t와 c는 각각 실행 시간과 사이클 수를 나타내며, 아래 첨자 g와 r은 컴파일러로 생성한 코드(generated code)인지 프로그래머가 직접 작성한 어셈블리 코드(reference code)인지의 여부를 각각 나타낸다. 또한, d와 p는 각각 데이터 메모리와 프로그램 메모리의 사용량을 가리키며, m은 메모리 사용량의 총량을 나타낸다. 한편, 실험 결과 중에서 a/b의 형태로 나타난 것은 각각 ANSI C를 사용한 실험 결과(a)와 컴파일러 회사에서 제공하는 확장된 C 언어를 사용한 결과(b)임을 각각 의미한다. 실험 결과, RISC 기반 DSP 프로세서에서 코드 변환기를 통해 얻은 코드의 실험 결과는 다른 프로세서에서 컴파일러를 통해 사용한 결과와 비교하여 매우 좋은 성능을 보이는 것을 알 수 있다.

Ⅶ. 결 론

본 논문에서는 메모리 오퍼랜드, MAC 명령어, 포스트 증가 간접 어드레싱, 하드웨어 루프와 같은 대표적인 DSP 프로세서 아키텍처의 특징을 RISC 프로세서에 결합시킨 구조의 프로세서 모델에서, 각 요소들이 성능에 미치는 영향을 평가하고 기존의 DSP 프로세서와 성능을 비교하였다. 특히, 효율적인 고급 언어 프로그래밍을 지원하는 장점을 가지는 본 아키텍처의 객관적이고 빠른 성능 평가를 위해, 대표적인 DSP 프로세서 아키텍처 구성 요소들을 효과적으로 활용하도록 최적화하는 코드변환기를 개발하여 사용하였다. 실험에 사용한 벤치마크 프로그램으로는 SPEC 벤치마크가 아닌 다양한 DSP 커널 알고리즘과 QCELP 음성부호화기를 채택하였다. 실험 결과, 네 가지 DSP 프로세서 아키텍처 구성 요소를 모두 갖춘 RISC 기반 DSP 프로세서는 DSP 커널 벤치마크와 QCELP 벤치마크에서 각각 259 %와 126 %의 성능 향상비를 나타내었다. 또한, RISC 기반 DSP 프로세서의 성능은 벤치마크의 특성에 따라 성능비 값이 최저 1에서 최고 10까지 매우 변화가 심하게 나타나는 것을 볼 수 있었다. 이와 같이 컴파일러 친근한 아키텍처의 채택과 코드변환기를 통한 빠른 성능 평가 실험 방법을 사용하여, 큰 하드웨어 추가 없이 컴파일된 코드로 높은 성능을 얻을 수 있는 RISC 기반 DSP 프로세서 구조를 제안하고 그 성능을 평가할 수 있었다.

감사의 글

※ 본 연구는 서울대학교 반도체공동연구소의 교육부 반도체분야 학술연구조성비 (과제번호: ISRC 97-E-2101) 및 한국과학재단의 특정기초 연구과제 96-0102-04-01-3에 의해 수행되었음.

참 고 문 헌

[1] E. A. Lee, "Programmable DSP Architectures: Part I," *IEEE ASSP Magazine*, pp. 4-19, Oct. 1988.

[2] E. A. Lee, "Programmable DSP Architectures: Part II," *IEEE ASSP Magazine*, pp. 4-14, Jan. 1989.

[3] *Buyer's Guide to DSP Processors*, Berkeley Design Technology, Inc., 1994.

[4] S. Segars, K. Clarke, and L. Goudge, "Embedded Control Problems, Thumb, and the ARM7TDMI," *IEEE MICRO*, vol. 15, no. 5, pp. 22-30, Oct. 1995.

[5] M. Dolle and M. Schlett, "A Cost-Effective RISC/DSP Micro-processor for Embedded Systems," *IEEE MICRO*, vol. 15, no. 5, pp. 32-40, Oct. 1995.

[6] C. C. Foster, R. H. Gonter, and E. M. Riseman, "Measures of opcode utilization," *IEEE Transactions on Computers*, vol. 13, no. 5, pp. 582-584, May, 1971.

[7] A. Lunde, "Empirical evaluation of some features of instruction set processor architecture," *Communications of the ACM*, vol. 20, no. 3, pp. 143-152, Mar. 1977.

[8] R. F. Cmelik, S. I. Kong, D. R. Ditzel, and E. J. Kelly, "An Analysis of MIPS and SPARC Instruction Set Utilization on the SPEC Benchmarks," *Proc. ASPLOS-IV*, pp. 290-302, 1991.

[9] D. Bhandarkar and D. W. Clark, "Performance from Architecture: Comparing a RISC and CISC with Similar Hardware Organization," *Proc. ASPLOS-IV*, pp. 310-319, 1991.

[10] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1990.

[11] C. B. Hall and K. O'Brien, "Performance Characteristics of Architectural Features of the IBM RISC System/6000," *Proc. ASPLOS-IV*, pp. 303-309, 1991.

[12] V. Živojnović, J. Martínez, C. Schläger, and H. Meyr, "DSPstone: A DSP-oriented Benchmarking Methodology," *Proc. ICSPAT '94*, Dallas, Oct. 1994.

[13] V. Živojnović, H. Schraut, M. Willems and R. Schoenen, "DSPs, GPPs, and Multimedia Applications - An Evaluation Using DSPstone," *Proc. ICSPAT '95*, Boston, Oct. 1995.

[14] *SPARClet Architecture Overview*, http://www.temic.de/semi/nt/sparclet/archit.html.

[15] *SPARClite User's Manual*, http://www.fujitsumicro.com/products/embctrl/pdf/sparcman.html.

[16] *The SPARC Architecture Manual, Version 8*, Prentice-Hall, Inc., 1992.

[17] *TMS320C5x User's Guide*, Houston, Texas Instruments Inc., 1993.

[18] *Digital Cellular System CDMA Analog Dual-Mode Mobile Station-Base Station Compatibility Standard*, Qualcomm, Inc., Mar. 1992.

[19] *Introduction to SHADOW*, Sun Microsystems. Inc., Jul. 1989.

[20] *TMS320C54x Reference Set Volume 1*, Houston, Texas Instruments Inc., 1996.

[21] *TMS320C3x User's Guide*, Houston, Texas Instruments Inc., 1994.

[22] *STD90/MDL90 0.35μm 3.3V CMOS Standard Cell Library for Pure Logic/MDL Products Data Book*, Samsung Electronics Co., Ltd, Jan. 1998.

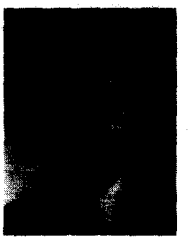
저 자 소 개

姜 志 浪(正會員)

1971년 12월 30일생. 1994년 2월 서울대학교 제어계측공학과 졸업(공학사). 1996년 2월 서울대학교 제어계측공학과 졸업(공학석사). 1996년 3월 ~ 서울대학교 전기공학부 박사과정 재학중. 주관심분야는 고성능 DSP 아키텍쳐, 컴파일러 및 VLSI 신호처리 등임

成 元 鎔(正會員)

1955년 4월 14일생. 1978년 2월 서울대학교 전자공학과 졸업(공학사). 1980년 2월 한국과학원 전기 및 전자공학과 졸업(공학석사). 1980년 2월 ~ 1983년 7월 금성사 중앙연구소. 1987년 7월 미국 University of California, Santa Barbara 전기 및 컴퓨터공학과 졸업(공학박사). 1989년 2월 ~ 현재 서울대학교 전기공학부 및 반도체 공동연구소 부교수. 1997년 ~ 현재 서울대학교 반도체공동연구소 집적시스템 설계센터(SEED : System EnginEering and Design center) 센터장. 주관심분야는 병렬처리 컴퓨터와 VLSI를 이용한 고속신호처리등임



李 鍾 馥(正會員)

1964년 8월 20일생. 1988년 2월 서울대학교 컴퓨터공학과 졸업(공학사). 1990년 2월 서울대학교 컴퓨터공학과 졸업(공학석사). 1998년 2월 서울대학교 전기공학부 졸업(공학박사). 1991년 ~ 1995년 서울대학교 반도체공동연구소 CAD 조교. 1998년 ~ 현재 LG반도체 메모리사업본부 설계 5팀 선임 연구원. 주관심분야는 마이크로 프로세서 구조, 프로세서 성능 모델 및 프로세서-메모리 복합칩임