프로세서 구조에 따른 DCT 알고리즘의 구현 성능 비교

준회원 이 재 성*, 정회원 박 영 철**, 종신회원 윤 대 희*

Performance Comparison of DCT Algorithm Implementations Based on Hardware Architecture

Jae-Seong Lee* *Associate Member*, Young-Cheol Pack** *Regular Member*,
Dae Hee Youn* *Lifelong Member*

요 약

본 논문에서는 MPEG 오디오 부호화 과정 중 서브밴드 필터뱅크를 구현하기 위해 사용되는 DCT(Discrete Cosine Transform) 과정에 대해 구현 시스템의 구조에 따른 DCT 알고리즘의 구현 결과와 성능 차이를 분석한다. 고속 DCT 알고리즘은 코사인 계수의 내적을 통해 구하는 직접 구현 방법보다 연산량이 현저하게 적은 것으로 알려져 있지만, 피연산자의 어드레스가 불규칙적이고 출력 데이터를 재정렬하는 과정이 필요하기 때문에 규칙성이 결여되며, 재정렬만을 위한 추가적인 연산이 필요한 경우도 있다. 따라서 DSP와 같이 반복적인 연산을 고속으로 수행하기 위해 최적화된 구조의 하드웨어에서는 알고리즘의 규칙성이 높은 직접 구현 방법에 비해 고속 알고리즘이 불리한 측면이 있으며, 더욱이 유효 자리수를 제한하는 경우, 직접 구현 방법에 비해 더 많은 프로세싱 단계를 거쳐야 하므로 누적 오차가 커진다. 본 논문에서는 알고리즘의 규칙성과 각 프로세서의 연산 방법간의 관계와 유효 자리수에 따른 누적 오차를 분석하고 프로세서의 구조에 따른 고속 알고리즘의 선택 기준을 제시하였다.

Key Words : DCT, MPEG Audio, DSP

ABSTRACT

This paper presents performance and implementation comparisons of standard and fast DCT algorithms that are commonly used for subband filter bank in MPEG audio coders. The comparison is made according to the architectural difference of the implementation hardware. Fast DCT algorithms are known to have much less computational complexity than the standard method that involves computing a vector dot product of cosine coefficient. But, due to structural irregularity, fast DCT algorithms require extra cycles to generate the addresses for operands and to realign interim data. When algorithms are implemented using DSP processors that provide special operations such as single-cycle MAC (multiply-accumulate), zero-overhead nested loop, the standard algorithm is more advantageous than the fast algorithms. Also, in case of the finite-precision processing, the error performance of the standard method is far superior to that of the fast algorithms. In this paper, truncation errors and algorithmic suitability are analyzed and implementation results are provided to support the analysis.

Ⅰ. 서 론

MPEG-1/2의 계층-II, III[1]에서 사용하는 서브밴드 필터 과정은 MP3 복호화 과정 중 약 50%의 연산량을 필요로 한다[2]. 서브밴드 필터 과정의 연산량을 감소시킬 수 있는 여러 가지의 고속 알고리

* 연세대학교 전자공학과 디지털신호처리 연구실 (dream7070@dsp.yonsei.ac.kr, dhyoun@yonsei.ac.kr),
** 연세대학교 컴퓨터정보통신공학부 (young00@dragon.yonsei.ac.kr)
논문번호 : KICS2006-02-088, 접수일자 : 2006년 2월 17일, 최종논문접수일자 : 2006년 5월 29일

즘이 제안된 바 있다[3-5].

기존 연구에서 제안된 고속 알고리즘들은 주로 곱셈 연산을 줄이는 대신 덧셈을 늘리거나 데이터를 재정렬하는 방식을 사용해 왔는데, 이는 기존의 마이크로 컨트롤러와 같은 하드웨어에서 곱셈 연산이 많은 처리시간이 필요로 하였기 때문이다[6]. 하지만 근래 DSP와 같은 프로세서에서는 MAC(multiply and accumulator)와 같이 곱셈과 덧셈을 동시에 수행할 수 있는 연산과, 하드웨어 반복 루프 기능[7]을 제공하므로 각 연산의 횟수뿐만 아니라 알고리즘이 갖는 규칙성 또한 매우 중요한 요소로 부각된다. 즉 고속 알고리즘에서 연산 횟수를 줄이기 위한 데이터 재정렬 과정은 알고리즘의 규칙성을 떨어뜨려 메모리 어드레스를 계산하기 위한 부가적인 연산량이 필요한 경우가 발생한다.

본 논문에서는 서브밴드 필터뱅크에서 사용되는 고속 알고리즘 중에서 가장 연산량을 많이 차지하는 Discrete Cosine Transform(DCT)[8] 알고리즘에 대해 알고리즘의 연산구조 및 구현 프로세서의 구조에 따른 성능 차이를 살펴보았다.

또한 DCT는 고정소수점 연산을 하는 경우, 절단(Truncation) 오차가 발생하는데 고속 알고리즘이 직접구현 방법 보다 연산 오차가 더 많이 발생하게 된다. 본 논문에서는 이를 오차 발생 모델을 통하여 자세히 분석하였다. DCT를 구현하기 위해서는 여러 가지 프로세서를 사용할 수 있다. 수식상의 연산량이 적은 고속 알고리즘이라도 구현 프로세서의 구조에 따라서는 실제 연산 수행시간이 많아질 수 있다. 이때 구현 시스템의 데이터 버스구조와 프로세서의 ALU 구조에 따라서 알고리즘의 연산 수행시간이 결정된다. 본 논문에서는 연산 수행시간에 영향을 주는 요인들에 대하여 분석한다.

본 논문의 순서는 2절에서 DCT 알고리즘과 Fixed-point 연산 오차에 대하여 설명하고, 3절에서는 실제로 DCT 알고리즘을 하드웨어로 구현할 때 나타나는 문제점에 관하여 설명한다. 4절에서는 C 프로그램과 DSP 구현을 통한 실험 결과에 대하여 논하고, 마지막 5절에서는 이들 내용을 바탕으로 결론을 맺는다.

Ⅱ. DCT 알고리즘과 Fixed-point 연산 오차 분석

N-point DCT를 코사인 계수와의 내적을 통해 직접 구현 방법을 사용하면 N^2번의 곱셈과 N(N-1)번의 덧셈이 필요하다. 이렇게 point 수의 제곱에 비례하는 연산량을 줄이기 위해 여러 고속 DCT 알고리즘이 제안되었다. 고속 알고리즘은 주로 단계별로 연산을 하므로 고정소수점 연산을 할 경우, 반올림 및 절단에 의한 고정소수점 연산 오차가 발생한다. 이 절에서는 DCT 연산의 오차를 분석하였다.

2.1 고속 DCT 알고리즘

고속 DCT 알고리즘은 두 가지 종류로 나눌 수 있는데 하나는 DCT 계수가 Discrete Fourier Transform (DFT)계수와 실수 부분이 유사한 점에서 착안된 Fast Fourier Transform (FFT)을 이용하는 방법[9, 10]과 시간 영역 혹은 주파수 영역에서의 간축(Decimation)을 사용하여 버터 플라이 구조로 구현하는 방법[11]이 있다. 이 절에서는 두 번째 접근 방법의 대표적인 고속 DCT 알고리즘인 Lee's DCT에 관하여 간략히 설명한다. 본 논문에서 사용하는 DCT와 IDCT의 변환식은 다음과 같다[8].

순방향

$$X(k)=\frac{2}{N}e(k)\sum_{n=0}^{N-1}\cos\left(\frac{(2n+1)k\pi}{2N}\right)x(n) \qquad k=0,1,\ldots,N-1 \tag{1}$$

역방향

$$x(n)=e(k)\sum_{n=0}^{N-1}\cos\left(\frac{(2n+1)k\pi}{2N}\right)X(n) \qquad n=0,1,\ldots,N-1 \tag{2}$$

여기서, $e(k)=\begin{cases}\frac{1}{\sqrt{2}}, & k=0\\ 1, & k\neq 0\end{cases}$

식 (1)과 (2)를 이용하여 DCT를 구현하는 경우, 곱셈수와 덧셈수는 각각 N^2과 N^2-N임을 쉽게 알 수 있다.

Lee's DCT[11]의 결과를 수식 (3), (4)에서 보인다.

$$X(k)=G(k) \tag{3}$$

$$\cdot X(2k+1)=H(k)+H(k+1) \qquad k=0,1,\ldots,\frac{N}{2}-1 \tag{4}$$

수식 (3), (4)에서 $G(k)$와 $H(k)$는 다음과 같이 정의 된다.

$$G(k)=\sum_{n=0}^{\frac{N}{2}-1}g(n)\,C_{2N/2}^{(2n+1)k} \tag{5}$$

$$H(k)=\sum_{n=0}^{\frac{N}{2}-1} h(n)\,C_{2N/2}^{(2n+1)k} \qquad (6)$$

또한 수식 (5), (6)에서 *g(n)*와 *h(n)*는 다음과 같다.

$$g(n)=x(n)+x(N-n-1) \qquad (7)$$

$$h(n)=\frac{x(n)-x(N-n-1)}{2C_{2N}^{(2n+1)}} \qquad (8)$$
$$n=0,1,\ldots,\frac{N}{2}-1$$

마지막으로 $C_{2N}^{(2n+1)k}$은 다음과 같이 정의 된다.

$$C_{2N}^{(2n+1)k}=\cos\left(\frac{(2n+1)k\pi}{2N}\right) \qquad (9)$$

수식 (7), (8)에서 보듯이 N-point DCT는 2개의 N/2-point DCT로 나눌 수 있고 같은 방법으로 4개의 N/4-point DCT로 분리할 수 있다. 이를 반복적으로 적용하면 최종엔 log_2N개의 2-point DCT만으로 연산이 가능해 진다. 따라서 수식상에 곱셈과 덧셈의 수는 $(N/2)log_2N$번의 실수 곱셈과 $(3N/2)log_2N-N+1$번의 실수 덧셈이다.

또한 Lee's DCT는 수식에서 알 수 있듯이 다소 복잡한 연산 과정을 필요로 한다. Lee's DCT를 위해서는 먼저 입력을 재정렬해야 한다. 재정렬된 입력은 2-point DCT형태까지 단계적으로 나뉘어 계산된다. 최종 출력 또한 bit-reverse 순서로 정렬 되어야 한다.

Lee's DCT 외에도 여러 알고리즘이 제안되었으나 모두 비슷한 연산량을 가지고 구조도 비슷한 구조를 가진다. 고속 알고리즘과 직접 구현 방법을 비교할 때 주로 연산 규칙성에 관하여 논할 것이므로 Lee's 알고리즘은 비교 대상으로 충분한 조건을 갖추었다고 할 수 있다.

2.2 유한 자릿수 표현에 따른 효과

고정 소수점 연산에서는 데이터를 유한 자릿수로 표현해야 하므로 고정소수점 오차가 발생한다. 또한 데이터를 한정된 메모리 혹은 레지스터에 저장을 함에 따라서 오차가 발생한다. 고속 알고리즘의 연산을 예를 들어보면, 4-point Lee's DCT의 구조를 보면 그림 1에서와 같이 3번의 단계에 걸쳐서 연산이 이루어 졌다. 그림에서 *Cm* 은 식(8)의 분모에 해당하는 값으로 다음과 같이 정의된다.

$$C_{2N}^{m}=Cm=\cos\left(\frac{m\pi}{2N}\right)$$

이 때 식 (8)에서의 코사인의 역수가 곱해지기 때문에 1보다 큰 수가 발생할 수 있으며, 따라서 오버 플로우를 방지하기 위해서는 곱한 결과의 소수점을 내려서 스케일링 해야 한다. 소수점을 내리는 것은 정수 부분의 자릿수를 늘리는 것으로 표현할 수 있는 수의 범위가 커진다. 하지만 상대적으로 소수 부분의 자릿수를 줄어들어 소수부분의 해상도가 낮아질 수 있다[16]. Lee's 4-point DCT에서 하나의 출력 샘플이 나오기까지 3번의 덧셈이 필요한데 각 단계별로 메모리 또는 레지스터로 옮겨야 하므로 각 단계에서 오차가 누적 되게 된다. 이는 DSP에서 사용하는 누산기가 다른 연산을 하기 위해서는 매번 누산기를 비워 두어야 하기 때문이다.

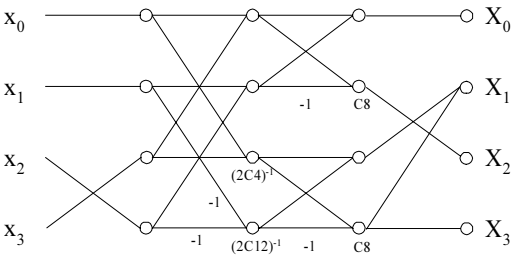


그림 1. 고속 Lee's 4-point DCT의 흐름도

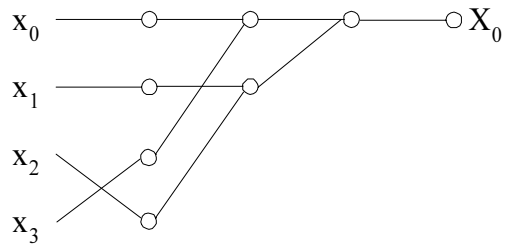


그림 2. Lee's DCT의 연산 과정의 예

곱셈에서는 2진수의 곱셈이므로 두 이진수의 자릿수의 합한 만큼의 자릿수로 결과가 표시된다. 그러므로 제한된 레지스터에 결과를 저장하려면 하위 자릿수를 버려야 하므로 오차가 발생할 수 있다. 단계적으로 연산되는 과정에서 이런 현상은 문제가 될 수 있다. 이렇게 각 노드에서 발생하는 오차의 범위는 (B+1)비트로 표현되는 유리수 일 때,

$$-\frac{1}{2}2^{-B}<\varepsilon_{\pi}<\frac{1}{2}2^{-B} \qquad (10)$$

와 같다. 오차는 이 범위 안에서 균일하게 분포되어

있으므로 오차의 분산은[15]

$$E\{\varepsilon_{\pi}^{2}\}=\frac{2^{-2B}}{12} \tag{11}$$

와 같다. N point DCT에서 하나의 출력을 내기 위해서는 일반적으로 (N-1)개의 버터 플라이 구조를 가진다. 따라서 출력 오차의 에너지는 다음과 같다.

$$E\{\varepsilon_{0}^{2}\}=(N-1)\frac{2^{-2B}}{12} \tag{12}$$

이때 N이 커지면 다음과 같이 근사화 할 수 있다

$$E\{\varepsilon_{0}^{2}\}\cong N\frac{2^{-2B}}{12} \tag{13}$$

DCT 출력의 절대값이 1 이하 일 때 다음의 조건을 만족해야 한다.

$$|X(k)|=\frac{2}{N}\sum_{n=0}^{N-1}\left|\cos\left(\frac{(2n+1)k\pi}{2N}\right)\right||x(n)|\le 1 \tag{14}$$
$$k=1,2,\ldots,N-1$$

$$\sum_{n=0}^{N-1}\left|\cos\left(\frac{(2n+1)k\pi}{2N}\right)\right|=\sqrt{\frac{N}{2}},\quad k\neq 0 \tag{15}$$

따라서 $|x(n)|\le\sqrt{1/2N}$이어야 한다. 이때 $x(n)$의 분산은

$$E\{|x(n)|^{2}\}=\frac{1}{6N} \tag{16}$$

이므로 전체$X(k)$의 분산은

$$\begin{aligned}E\{|X(k)|^{2}\}&=\frac{4}{N^{2}}\sum_{n=0}^{N-1}\left|\cos\left(\frac{(2n+1)k\pi}{2N}\right)\right|^{2}E\{|x(n)|^{2}\}\\&=\frac{4}{N^{2}}\frac{N}{2}\frac{1}{6N}\\&=\frac{1}{3N^{2}}\qquad,k=1,2\ldots,N-1\end{aligned} \tag{17}$$

이므로 $X(k)$의 신호 대 잡음의 에너지 비를 구해보면

$$\frac{E\{|X(k)|^{2}\}}{E\{|\varepsilon_{0}|^{2}\}}\cong\frac{1}{3N^{2}}\frac{12}{N2^{-2B}}=\frac{4}{N^{3}2^{-2B}} \tag{18}$$

이 된다.

고속 알고리즘에 반해 직접 구현 방법에서는 그림 3과 같이 각 입력 데이터의 곱셈 결과를 보호 비트를 포함하는 누산기에 누적시켜 더해 가면서 값을 구하므로 연산 중간에 스케일링이나 반올림에 의한 양자화 오차는 발생하지 않는다. 즉 수식 (14)과 같이 누산기 P에서는 각 입력 $x(n)$와 계수 $C_{2N}^{(2n+1)k}$의 곱한 결과를 누적시켜서 더하기만 하면 된다.

$$\sum_{n=0}^{N-1}x(n)C_{2N}^{(2n+1)k}=P \tag{19}$$

따라서 마지막의 결과를 레지스터 등에 저장할 때에만 스케일링과 반올림 오차가 발생한다. 결과적인 출력 오차의 에너지와 신호 대 잡음비는 다음과 같다.

$$\frac{E\{|X(k)|^{2}\}}{E\{|\varepsilon_{0}|^{2}\}}\cong\frac{1}{3N^{2}}\frac{12}{2^{-2B}}=\frac{4}{N^{2}2^{-2B}} \tag{20}$$

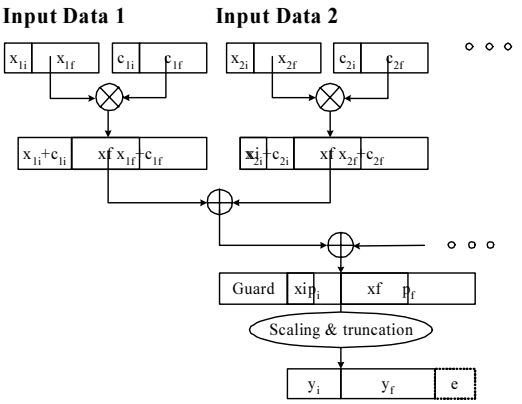


그림 3. 직접 구현 방법의 고정 소수점 연산 오차

고속 알고리즘에 비하여 직접 구현 방법에 의한 오차의 발생 빈도가 적기 때문에 오차를 비교하였을 때는 직접구현방법이 이득이 있을 것으로 예상할 수 있다. 32-point DCT 연산을 20비트의 DSP[12]에서 구현하고자 할 때 예상되는 고속 알고리즘의 SNR은 약 92dB 이고 직접 구현 방법의 경우 약 107 dB가 된다. 따라서 약 15dB 정도 SNR 차이를 기대할 수 있다.

Ⅲ. 하드웨어 구조에 따른 DCT 알고리즘 성능

하드웨어에서 DCT 알고리즘을 구현할 때는 절대

적인 수식상의 연산량 외에도 하드웨어의 특성을 고려한 성능을 분석해야 한다. 즉 구현하는 프로세서의 메모리 접근 방식이나 연산기의 구조, 메모리 사용량 등을 고려하여 분석함으로써 종합적인 알고리즘의 구현 성능을 평가할 수 있다. .

3.1 메모리 접근방식

DSP에서는 반복적인 곱셈과 누적 덧셈을 효율적으로 구현하기 위해 최적화된 메모리 접근 방식을 가진다. 하지만 Lee's 고속 DCT의 경우는 DSP의 메모리 접근 방식을 효율적으로 사용할 수 없는 구조로 되어 있다. 즉 Lee's DCT는 입출력을 재정렬해야 하고 2-point DCT까지 각 단계에 걸쳐 중간 결과 값을 메모리에 저장해야 한다.

Lee's DCT의 경우 $(N/2)\log_2 N$번의 실수 곱셈과 $(3N/2)log_2N-N+1$번의 번의 실수 덧셈이 필요하므로 각 연산시에 피연산자를 메모리에서 2개씩 패치(data patch)한다고 가정하면 N-point DCT 의 경우 $4Nlog_2N-2N+2$회의 메모리 접근이 필요하다. 또한 마지막 출력을 재정렬하는데 각 샘플당 2번의 메모리 접근한다고 가정하면 $2N$회의 메모리 접근이 필요하다. 그러므로 N-point DCT를 수행하기 위해서는 총 $4Nlog_2N+2$회의 메모리 접근이 필요할 것이다. 반면에 직접 구현 방법은 N-point의 경우 곱셈 N^2회와 덧셈 $-N$회가 필요한데 덧셈은 누산기에서 수행되므로 곱셈의 결과를 메모리에 저장하지 않고 누산기에서 직접 덧셈을 수행한다. 따라서 곱셈 수행 시 2번의 메모리 접근이 필요하다고 가정할 때 N-point DCT 의 경우 총 $2N^2$회로 Lee's DCT보다 입력 샘플(N)이 많아질수록 메모리 접근 회수는 더욱 차이가 나게 된다[12]. 메모리 접근 회수를 비교하면 Lee's DCT가 유리한 측면하다.

하지만 고속 Lee's DCT의 경우 메모리 접근 순서가 불규칙적이므로 해당 메모리의 주소를 계산해야 한다. 이는 데이터의 재배치를 통해서 수식상의 곱셈 등의 연산 수를 줄이기 위함이다. 반면, 직접 구현 방법을 사용하면 메모리의 접근 횟수는 많아도 선형적으로 접근하므로 따로 메모리 주소를 계산할 필요가 없다. DSP는 메모리 접근 하는데 1 클럭 사이클이 소요되지만 이와 달리 마이크로 컨트롤러와 같은 RISC 프로세서의 경우, 하드웨어에 따라 수 클럭 사이클이 소요되므로 메모리 접근 횟수가 많을수록 불리하다.

3.2 연산속도

다음과 같은 컨볼루션을 DSP로 구현 한다고 할 때,

$$\sum_{i=0}^{N-1} a_i \cdot b_i \qquad (21)$$

MAC 연산이 지원하지 않는 하드웨어일 경우 N번의 곱셈과 (N-1)번의 덧셈, 그리고 2N 번의 메모리 읽기, 1번의 메모리 쓰기가 필요하다. 각 명령어가 모두 1-클럭 사이클에 동작한다고 가정하였을 경우, 총 4N 번의 클럭 사이클이 필요하다. 즉 곱셈, 덧셈, 메모리 읽기 등을 모두 각각 수행해야 한다. 여기에 만약 메모리 읽기가 D-클럭 사이클이 걸린다고 한다면 2N + 2(N+D)만큼의 시간이 소요 되므로 연산량은 2D 사이클이 증가한다[12]. 반면에 MAC연산을 모두 지원하고 동시에 메모리에 불러오기까지 가능할 경우, 메모리 접근하는데 N+2번으로 가능하다. 하드웨어 구조에 따른 MAC연산량을 표 1에서 정리하였다. 예를 들어 32-point DCT의 경우를 보면 표 2와 같이 직접 구현 방법은 곱셈 수가 32의 제곱인 1024회가 필요하고, 덧셈 수는 992회가 필요하다. 또한 곱셈 및 덧셈을 위한 메모리 접근 회수는 2048회이고 DCT 연산 결과를 메모리에 저장하는데 32회가 소요된다[12]. 표 2를 바탕으로 표 1과 같이 MAC에서 메모리 접근 방법에 따라서 연산량을 예상해 보면 표 3와 같이 MAC의 구조에 따라서 연산량은 크게 차이가 난다.

Lee's DCT의 경우는 MAC을 사용할 수 없는 구조로 구성되어 있기 때문에 연산 시간은 MAC을 사용하였을 경우 보다 늘어나게 된다. 하지만 Lee's DCT의 경우 곱셈과 덧셈의 수가 적다. 32-point Lee's DCT의 경우, 표 4와 같이 덧셈 곱셈 80회, 덧셈 216회, 메모리에서 피연산자를 읽는데 288회, 최종 결과 및 중간 계산 결과 저장하는데 208회 등의 연산이 필요하다. Lee's 알고리즘은 모든 연산을 각각 수행 하여야 하므로 총 연산 수는 792회 이다. 이 연산 수는 표 3에서 직접 구현방법의 가장 적은 수행 사이클인 1026에 비하여 작은 수이나, 상대적인 연산상의 이점이 그리 크지 않음을 알 수 있다. 더욱이 Lee's 알고리즘은 데이터 재배치를 위한 추가적인 연산이 필요하기 때문에 직접구현 방법에 비하여 연산량의 이점은 더욱 줄어든다

표 1. 하드웨어 구조에 따른 MAC연산량

연산방법	P‖A‖M	P‖A, M	P, A, M	P, A, M+D
연산량	N+2	3N+2	4N	2N+2(N+D)

(P: 곱셈, A: 덧셈, M: 메모리 접근 ‖: 동시 수행 D: 메모리 접근 지연)

표 2. 직접 구현 방법에 의한 DCT의 연산량 (N=32)

	N_M	N_A	$N_{RAMread}$	$N_{RAMwrite}$	$N_{ROMread}$
값	1024	992	32	1024	1024

(N_M : 곱셈 회수, N_A:덧셈 회수, $N_{RAMread}$: RAM 읽기 회수, $N_{RAMwrite}$: RAM 쓰기 회수, $N_{ROMread}$: ROM 읽기 횟수)

표 3. 직접 구현 방법의 DCT의 수행 사이클 (N=32)

항목	P‖A‖M	P‖A, M	P, A, M	P, A, M+D
연산량	1026	3038	4048	4048+2D

표 4. Lee's 고속 DCT의 수행 사이클 (N=32)

	N_M	N_A	$N_{RAMread}$	$N_{RAMwrite}$	$N_{ROMread}$	합계
값	80	216	208	208	80	792

(* : 모든 연산이 1 클럭 사이클에 수행할 경우)

3.3 메모리 사용량

Lee's N-point DCT의 경우 2N개의 입출력 버퍼와 N개의 코사인 계수, 그리고 bit-reverse 입력을 위해 메모리 주소를 저장하고 있을 N개의 저장 공간이 필요하다. 따라서 최대 4N개의 메모리가 필요하게 된다. 직접구현방법 DCT의 경우 MAC을 통하여 순차적으로 메모리에 접근하기 때문에 2N개의 입출력 버퍼와 4N개의 코사인 계수가 필요하다. 그러므로 총 6N개의 메모리가 소요된다. 따라서 고속 알고리즘이 메모리 사용면에서는 유리하다.

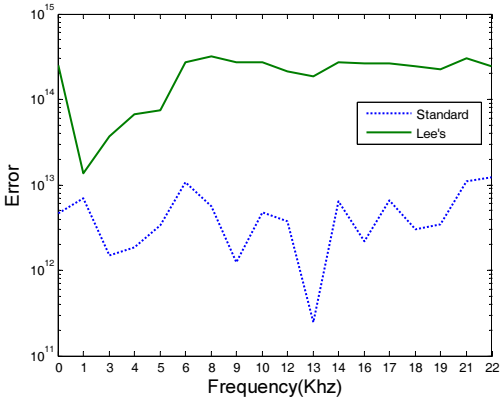
Ⅳ. 실험 결과

4.1 고정 소수점 연산에 따른 오차 측정

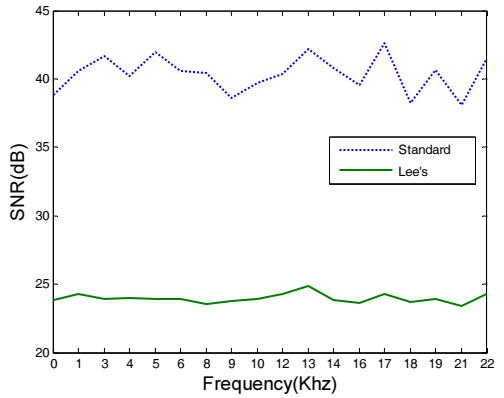
유한 자릿수 표현에 따른 고정 소수점 연산 결과를 비교하기 위해 Lee's Fast DCT와 기존의 MAC을 이용한 DCT 고정소수점 오차를 비교하였다. 이를 위해 Lee's Fast DCT와 직접 구현 방법을 통한 DCT에 대해 각각 20비트 고정 소수점 연산을 수행하였다. 그림 4에서 알 수 있듯이 입력으로 백색 잡음을 사용하여 기존의 MAC를 이용한 DCT가 Fast DCT보다 오차가 현저하게 적은 것을 알 수가 있다. 또한 SNR도 MAC를 이용한 DCT가 평균적으로 약 15dB 정도 우수한 것을 확인 하였다. 이 결과는 2.2절에서 유도한 식 (18)과 (20)에 의한 결과와 일치하는 것이다.

4.2 실시간 구현 결과

고속 DCT와 MAC을 이용한 DCT의 연산 속도를 비교하기 위하여 서로 다른 3가지 구조의 프로세서에 대하여 구현을 해보았다. 구현 결과로 제시



(a) 32-pt DCT의 오차



(b) 32-pt DCT의 SNR 비교

그림 4. 각 구현방법에 따른 DCT 출력 비교

되는 클럭 사이클 수는 각각의 DCT 알고리즘을 어셈블리 프로그래밍한 후, 연산량 및 메모리를 최적화하는 과정을 거쳐 얻어진 코드로부터 측정된 것이다.

4.2.1 개선된 하바드 구조 프로세서에 의한 구현

첫 번째로 고려한 DSP[12]는 데이터 메모리는 2개로 나뉘어 있고 각각 데이터 버스를 가지고 있어서 곱셈기에 동시에 2개의 피연산자를 불러올 수 있다.(P‖A‖M 구조) 데이터 메모리에 달려 있는 버스이므로 필터 계수와 같은 고정된 값뿐 아니라 가변적인 데이터도 동시에 불러 올 수 있다.

이 구조를 사용하여 32-point DCT를 MAC을 이용하여 수행할 경우, 1,323회의 어셈블리 명령어로 수행되었다. 반면에 Lee's Fast DCT를 사용한 경우는 1,479회의 어셈블리 명령어가 사용되었다. 수식적으로 Fast DCT는 곱셈 80회, 덧셈 209회이고, 기존 DCT는 곱셈 1,024회, 덧셈 1,024회이다.

DSP에서 구현한 결과와 수식상 연산량의 차이는 Fast DCT는 데이터 접근 순서가 규칙적이지 않기 때문에 데이터 pointer 조절을 위해 추가적인 연산이 필요하고, 불규칙한 연산 구조로 인해 MAC 알고리즘을 효율적으로 사용할 수 없는 구조로 인해 발생한다.

4.2.2 하바드 구조 프로세서에 의한 구현

하바드 구조[13]의 데이터 메모리에는 한 개의 데이터 버스만 가지고 있다. 물론 MAC 연산을 위해서 두 개의 피연산자를 동시에 불러오는 것이 필요하므로 프로그램 메모리에서도 피연산자를 불러올 수 있도록 되어있다. 하지만 다른 사칙 연산과는 데이터의 평행 이동이 동시에 수행될 수 없다. (P||A, M 구조) 데이터의 평행 이동이 불가능 하게 되면 Lee's 고속 DCT의 경우 모든 메모리 접근은 다른 사칙 연산등과 동시에 수행될 수 없고, 항상 독립적으로 수행 되어야 하기 때문에 매우 불리하게 작용한다. 이러한 이유로 하바드 구조의 DSP에서 32-point Lee's DCT를 구현한 결과, 개선된 하바드 구조보다 더 많은 2,192 클럭 사이클이 소요되었다. 한편 직접 구현 방법을 사용한 경우는 개선된 하바드 구조에서 구현한 경우와 유사힌 1,312 클럭 사이클이 소요되었다. 이는 하바드 구조에서처럼 메모리 접근이 불리한 프로세서에서는 고속 알고리즘이 직접 구현 방법에 비해 더 불리한 알고리즘이라는 사실을 보여준다.

4.2.3 폰 노이만 구조 프로세서에 의한 구현

마이크로 컨트롤러[14]는 폰 노이만 구조로 데이터 버스와 프로그램 버스가 나뉘어 있지 않다. 따라서 DSP와 달리 MAC 연산을 할 수 없는 구조이다. 또한 캐쉬 기반의 메모리 구조로써 캐쉬에 데이터가 없을 때는 외부 메모리로부터 데이터를 가져오기 때문에 지연이 발생한다. 따라서 마이크로 컨트롤러에서는 직접 구현 방법이 갖는 장점이 없게 된다. 마이크로 컨트롤러에서 구현한 결과는 ARM9 기반의 시뮬레이터에서 어셈블리 코딩을 하여, 단일 싸이클 메모리 접근이 가능하다는 가정하에 최적화된 소스 코드를 가지고 실험을 하였다. 구현 결과 32-point Lee's DCT는 3,693 클럭 사이클이 소요되었고, 직접 구현 방법의 경우에는 17,259 클럭 사이클이 소요되었다. 이 결과는 앞서 나온 마이크로 컨트롤러의 특징에 부합하는 결과로 직접 구현 방법을 통한 DCT가 Lee's DCT보다 연산 속도 면에서 4배 이상 느린 성능을 보였다.

표 5. Lee's DCT와 직접 구현 방법의 연산량 비교

	Lee's DCT			직접 구현방법		
곱셈	80			1024		
덧셈	209			1024		
Processor	A	B	C	A	B	C
수행 사이클	1479	2192	3693	1323	1312	17258

A type : 평행 연산(parallel instruction), 평행 메모리 접근(parallel move) 지원, 개선된 하바드 구조를 사용

B type : 제한된 평행 연산, 평행 메모리 접근 지원 하지 않음, 하바드 구조 사용

C type : 캐쉬 기반으로 메모리 접근 지연, 평행 연산 지원하지 않음, 폰 노이만 구조 사용

이상의 결과를 바탕으로 Lee's DCT와 직접 구현 방법간의 연산량 차이를 표 5에 정리하였다.

A type DSP[12]는 개선된 하바드 구조의 DSP이고 B type DSP[13]는 기존 하바드 구조를 사용하는 DSP이다. C type 프로세서[14]는 폰 노이만 구조를 갖는 마이크로 컨트롤러를 나타낸다. DSP에서 MAC 연산을 기반으로 하는 직접 구현 방법은 프로세서의 구조가 달라지더라도 수행 사이클 면에서 크게 차이가 나지 않았지만, 고속 알고리즘과 같이 메모리 접근이 빈번하고 불규칙적인 연산에서는 메모리 접근 방식에 따라서 많은 차이를 보인다.

반면 마이크로 컨트롤러는 캐쉬 기반의 메모리 구조로 메모리 접근하는데 지연이 발생 할 수 있고, MAC를 사용할 수 없는 구조이므로 직접 구현 방법의 알고리즘 규칙성이 장점이 되지 못하며, 오히려 많은 메모리 접근과 절대적으로 많은 연산량으로 인하여 마이크로 컨트롤러에서는 Lee's DCT보다 훨씬 더 많은 클럭 사이클이 필요한 것을 알 수 있었다.

V. 결 론

본 논문에서는 하드웨어 구조에 따라서 고속 DCT 알고리즘과 직접 구현 방법을 통한 DCT간의 성능 분석을 통하여 효율적인 고속 알고리즘의 선택 기준을 제시하였다. 고속 DCT 알고리즘은 수식상의 연산량은 직접 구현 방법보다 현저하게 적음에도 DSP에서는 MAC을 이용한 직접 구현 방법이 오히려 수행 사이클이 적었다. 또한 고속 DCT 알고리즘은 단계별로 중간 결과값을 메모리에 저장하

므로 오차가 누적되어 직접 구현 방법보다 오차가 많았다. 하바드 구조의 프로세서에서는 데이터의 평행 이동이 제한적이므로 Lee's DCT와 같이 메모리 접근이 불규칙할 경우 프로세서의 클럭 사이클 많아 진다. 이에 반하여 개선된 하바드 구조에서는 데이터의 평행이동이 가능하므로 하바드 구조보다는 연산량에서 이득이 있다. MAC을 이용한 직접 구현 방법은 하바드 구조와 개선된 하바드 구조에서 모두 비슷한 성능을 보였다. 또한 두 구조에서 모두 Lee's 고속 알고리즘보다 직접 구현 방법이 적은 클럭 사이클로 수행되었다. 다만 폰 노이만 구조를 갖는 마이크로 컨트롤러에서는 메모리 접근 지연이 발생하고 MAC 연산을 사용할 수 없으므로 직접 구현 방법이 더 많은 클럭 사이클이 소요되었다.

참 고 문 헌

〔1〕 ISO/IEC JTC1/SC29/WG11 MPEG, International Standard ISO/IEC IS 13818-3 "Information technology Generic Coding of Moving Pictures and Associated Audio:", Part 3: Audio, 1994.

〔2〕 Yingbiao Yao, and et al., "Embedded Software Optimization for MP3 Decoder Implemented on RISC Core," IEEE Trans. on Consumer Electronics, Vol. 50, No. 4, Nov. 2004.

〔3〕 K. Konstantinides, "Fast Subband Filtering in MPEG AUDIO Coding.", *IEEE Signal Processing Letters*, vol. I, no.2, pp. 26-28, Feb., 1994.

〔4〕 P. Duhamel, Y. Mahieux, and J. P. Petit, "A fast algorithm for the implementation of filter banks based on time domain aliasing cancellation," *in Proc. IEEE Int. Conf. Acoustics, Speech, and Signal Processing '91*,Toronto, ON, Canada, May 1991, pp. 2209-2212.

〔5〕 V. Britanak and K. R. Rao, "An efficient implementation of the forward and inverse MDCT in MPEG audio coding," *IEEE Signal Processing Lett*, vol. 8, pp. 4851, Feb. 2001.

〔6〕 Vijay K. Madisetti, "VLSI Digital Signal Processors- An Introduction to Rapid Prototyping and Design Synthesis", *IEEE Press*, 1995

〔7〕 Berkeley Design Technology, Inc., "Choosing a DSP Processor", http://www.BDTI.com

〔8〕 N. Ahmed, T. Natarajan, and K. R. Rao, "Discrete cosine transform," *IEEE Trans. Comput.*, vol. C-23, pp.90-94., Jan. 1974

〔9〕 B. D. Tseng and W. C. Miller, "On computing the discrete cosine transform," *IEEE Trans. Comput.*, Vol. C-27, pp.966-968, Oct., 1978

〔10〕 M. J. Narasimha and A. M. Peterson, "On the computation of the discrete cosine transform," *IEEE Trans. Commun.*, vol. COM-26, pp. 934-946, June 1978.

〔11〕 B. G. Lee, "A NewAlgorithm to Compute the Discrete Cosine Transform", *IEEE Trans, on Acoust., Speech, Signal Processing*, pp.1243-1245, Dec. 1984

〔12〕 김준석 "*DSP 코어 기반 오디오 프로세서의 설계*", 연세 대학교 박사 학위 논문, 2000,12

〔13〕 http://www.zaram.com/product1.htm

〔14〕 http://arm.com/products/CPUs/families/ARM9Family.html

〔15〕 Alan V. Oppenheim, "*Discrete time Signal Processing*", Prentice hall, 1999.

〔16〕 C. Loeffler, A. Ligtenberg, and G.S. Moschytz, "Practical Fast 1D DCT Algorithms with 11 Multiplications," Proc. Int'l Conf. Acoustics, Speech and Signal Processing, Vol. 2, IEEE, New York, 1989, pp. 988-991.

이 재 성 (Jae-Seong Lee) 준회원

2003년 2월 건국대학교 전자공학과 졸업

2005년 2월 연세대학교 전자공학과 석사

2005년 3월~현재 연세대학교 전자공학과 박사과정

<관심분야> 디지털신호처리, 오디오 신호처리

박 영 철 (Young-Cheol Pack) 정회원

한국통신학회 논문지 제29권 6C호 참고

윤 대 희 (Dae Hee Youn) 정회원

한국통신학회 논문지 제29권 6C호 참고