論 文
40～4～1

고정 소수점 연산에 의한 고속 DCT 알고리듬의 오차해석

A Fixed-Point Error Analysis of fast DCT Algorithms

尹 逸 東*·李 商 郁**
(Il-Dong Yun · Sang-Uk Lee)

Abstract- The discrete cosine transform (DCT) is widely used in many signal processing areas, including image and speech data compression. In this paper, we investigate a fixed-point error analysis for fast DCT algorithms, namely, Lee[6], Hou[7] and Vetterli[8]. A statistical model for fixed-point error is analyzed to predict the output noise due to the fixed-point implementation. This paper deals with two's complement fixed-point data representation with truncation and rounding. For a comparison purpose, we also investigate the direct form DCT algorithm. We also propose a suitable scaling model for the fixed-point implementation to avoid an overflow occurring in the addition operation. Computer simulation results reveal that there is a close agreement between the theoretical and the experimental results. The result shows that Vetterli's algorithm is better than the other algorithms in terms of SNR.

1. 서 론

DCT (discrete cosine transform)는 상관 계수가 높은 신호의 감축에 있어 최적변환인 KLT (Karhunen Loeve transform)의 성능에 가장 근접하는 변환이기 때문에 1974년 Ahmed등에 의하여 처음 제안된 이후로 디지탈 신호처리 분야, 특히 음성신호나 영상신호의 데이타 감축에서 가장 많이 사용되는 변환중의 하나이다[1]. 따라서 고속 연산을 위한 여러 종류의 FDCT (fast discrete cosine transform) 알고리듬과 하드웨어 구현을 위한 다양한 구조들이 제안되어 왔다[3～8]. 그러나 어떤 디지탈 신호 처리 알고리듬을 실제적인 하드웨어로 구현하는 경우에는 입력 신호나 계수들을 유한 자리의 이진수로 표현해야 하므로 이로 인한 출력의 오차가 발생하며 이를 FWL (finite word length) 영향이라 한다. 그러므로 어떤 고속 알고리듬을 이용하는데 있어 그 속도뿐만 아니라 FWL의 영향 또한 그 알고리듬의 사용에 있어서 반드시 고려해야 할 문제중의 하나이다. 특히 고속 알고리듬들은 DFT (Discrete Fourier Transform)의 경우에서 보는바와 같이 원래의 알고리듬보다 FWL의 영향에 의한 출력 오차가 큰 경향이 있으며 고속 알고리듬중에서도 그 특성에 따라

*正 會 員 : 서울大 大學院 制御計測工學科 博士課程
**正 會 員 : 서울大 工大 制御計測工學科 敎授 · 工博
接 受 日 字 : 1990年 11月 28日
1 次 修 正 : 1991年 3月 17日

FWL의 영향을 많이 받는 것과 적게 받는 것이 있다[2]. 그런데 DCT가 현재 영상 부호화 등에서 많이 사용되고 있지만 각 고속 알고리듬에 대한 FWL 영향에 대한 연구는 수행되어 있지 않다. 그리고 현재 반도체 기술로는 알고리듬을 하드웨어로 구현함에 있어서 여러가지 이유에서 대부분 고정소수점연산을 아직까지 이용하므로 본 논문에서는 여러가지 DCT 알고리듬에 대한 FWL영향중에서 고정 소수점 연산에 의한 오차를 분석하였고 이를 위하여 고정 소수점 연산에서 발생하는 오차의 특성과 DCT 알고리듬을 고정 소수점으로 계산할때 오버 풀로우 방지를 위한 스케일링(scaling) 방법을 고찰하였다.

본 논문에서 오차를 분석한 고속 알고리듬은 Hou[7], Lee[6]와 Vetterli[8]로 모두 순환식으로 정의되며 N-point의 경우 곱셈수가 모두 $1/2N\log_2 N$로 현재까지 제안된 고속 알고리듬중에 가장 적은 수의 곱셈수를 갖는 것중의 하나이다. 아울러 이와 비교하기 위하여 직접 구현 방법에 대한 분석을 아울러 하였다.

2. DCT의 정의와 오차모델

2.1 DCT의 정의

DCT오차 분석에 앞서서 우선 DCT의 정의에 대하여 살펴본다. 본 논문에서 분석한 DCT는 가장 많이 사용되는 DCT-2[2]로서 주어진 데이타 수열(sequence) $\{x(k): k=0, 1, \cdots, N-1\}$에 대하여 DCT수열 $\{y^*(i): i=0, 1, \cdots, N-1\}$이 다음과 같이 주어진다.

$$Y^*(i)=\left[\frac{2}{N}\right]\cdot u(i)\sum_{k=0}^{N-1}x(k)\cos\left[\frac{(2k+1)i\pi}{2N}\right], \tag{1}$$

여기서

$$u(i)=\begin{cases}\frac{1}{\sqrt{2}}, & i=0,\\ 1, & i\neq 0.\end{cases}$$

앞으로의 논문의 전개에서는 앞의 계수는 생략한 $Y^*(i)$의 비정규화된(donormalized) 형태를 $y(i)$라 하고 다음과 같이 정의하였다.

$$Y^*(i)=2u(i)y(i) \tag{2}$$

또한 이후로의 수식은 편의상 행렬은 대문자로, 벡터는 소문자로 나타내며 괄호안의 처음값은 행렬과 벡터의 차수이고 아래 첨자는 행렬 또는 벡터 원소의 위치를 표시한다. 그래서 식 (2)에서의 $y(i)$를 행렬과 벡터 형태로 나타내면

$$y(N)=C(N)x(N) \tag{3}$$

여기서 $C(N)_{i,j}=\frac{1}{N}\cos\left\{\frac{(2j+1)i\pi}{2N}\right\}$이다.

2.2 오차모델과 오차의 통계적 특성

일반적인 디지탈 신호 처리에서 FWL 영향으로 인하여 어떤 오차가 생기는가를 알아보자. 디지탈 신호 처리에서 FWL영향은 크게 3가지로 나누어 볼 수 있다. 첫째는 연속적인 신호를 디지탈로 변환하여 유한한 수로 표현할 때 생기는 양자화 오차(quantization error)이다. 둘째는 상수나 계수들이 유한자리의 정도(precision)로 표현될때 생기는 오차이다. 마지막으로 산술연산의 결과로 증가하는 자리수를 일정하게 유지함으로써 발생하는 오차가 있다. 예를 들면 b비트 베이타를 b비트 상수로 곱할 경우 (2b-1)비트의 결과가 생긴다. 이 결과를 다음 연산을 위해 b비트로 유지할 때 발생하는 오차이다. 이러한 유한 자리수를 유지하는 방법에는 절단(truncation) 또는 반올림(round off)이 있다. 절단 오차와 반올림 오차는 데이타의 표현 방식에 따라 특성이 다르게 되는데 본 논문에서는 가장 보편적으로 이용되는 2의 보수(2's complement) 형태의 표현에 대하여 분석하였고 그림 1에 각각의 특성을 나타내었다. 일반적으로 이러한 오차는 입력 값을 균일 양자기(uniform quantizer)로 양자화 할 때 발생하는 양자화 오차로 간주하는데 시스템 분석의 입장에서는 이러한 오차가 설계된 시스템에 비선형한 영향을 주기 때문에 일반적으로 그 영향을 잡음 또는 오차 신호

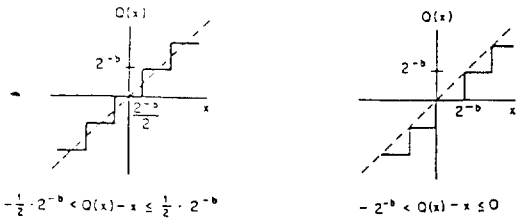

그림 1 반올림과 절단에 의한 오차특성
(a) 반올림 오차 (b) 절단 오차

Fig. 1 A characteristics of rounding error and truncation error
(a) rounding error (b) truncation error

로 나타낸다. 이러한 오차 신호에 대한 수학적 분석을 위해서는 이를 정칙 함수(deterministic function)로 나타내기가 어렵기 때문에 오차에 대한 보다 쉬운 모델링이 요구된다. 일반적으로 오차신호를 백색 잡음으로 간주하는데 최근의 연구결과 입력 신호의 동적 영역(dynamic range)가 어느 이상으로 커지면 다음의 가정들이 만족된다고 알려져있다[9].

양자기 $[\cdot]_Q$로 발생한 오차를 $e(n)=Q[ax(n)]-ax(n)$로 정의할 때

i) 오차는 정체된(Stationary)불규칙 프로세스이다.

ii) 오차의 확률분포는 균일하다.

iii) 오차와 입력 신호는 비상관화(uncorrelated)되어 있다.

iv) 오차는 상호 독립이다. 즉, 백색 잡음 프로세스이다.

이러한 가정하에 반올림 오차와 절단 오차에 대한 통계적 특성을 알아보자. 반올림오차 e_R의 분포는 그림 1(a)에 나타낸 것과 같이 $-1/2Q<e_R\leq 1/2Q$의 범위에서 균일하게 분포된다. 또한 데이타폭을 부호에 1비트와 크기에 b비트로 하여서 양자화 구간폭 Q를 2^{-b}로 하였다. 이때 반올림으로 인하여 k비트를 버린다고 할 때, 즉 $(b+k)$비트를 b비트로 반올림 하였을 경우에 오차의 평균 $\mu_{R(k)}$과 분산 $\sigma^2_{R(k)}$은

$$\mu_{R(k)}=\frac{1}{2L}Q, \tag{4}$$

$$\sigma^2_{R(k)}=\frac{L^2-1}{12L^2}Q^2,\quad L=2^k \tag{5}$$

로 쉽게 유도할 수 있다. 또한 절단 오차 e_T의 분포는 그림 1(b)에 나타낸 것과 같이 $-Q<e_T\leq 0$의 범위에서 균일하게 분포된다. 절단 오차는 반올림 오차와는 달리 항상 음이기 때문에 분석하는 알고리듬의 평균 오차에 큰 영향을 주게 된다. 절단으로 k비트를 버린다고 할 때 오차의 평균 $\mu_{T(k)}$과 분산 $\sigma^2_{T(k)}$은

$$\mu_{T(k)}=\frac{1-L}{2L}Q, \tag{6}$$

$$\sigma^2_{T(k)}=\frac{L^2-1}{12L^2}Q_2,\quad L=2^k\cdot \tag{7}$$

3. DCT 알고리듬의 오차 분석

3.1 직접 구현 방법의 오차 분석

현재까지 상용화되어 있는 DCT chip은 대부분

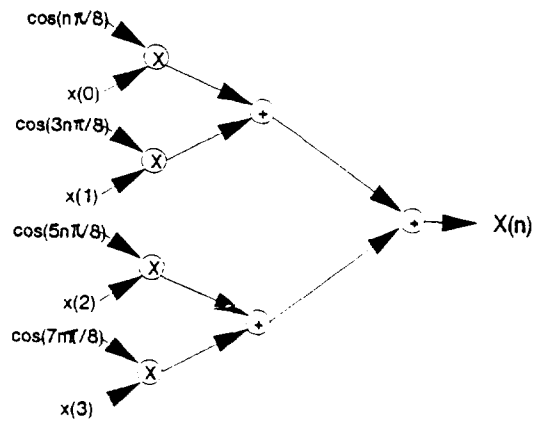

그림 2 직접 구현 방법의 구조
Fig. 2 A archtecture for direct-form implementation

직접 구현 방법으로 되어 있는데 이는 행열과 벡터의 곱을 계산하는 구조이다. 즉, 하나의 DCT출력을 얻기 위하여 $N(N=2^m)$개의 DCT계수와 N개의 입력 데이타 간에 내적(inner product)으로 표현되는데 그림 2에 $N=4$인 경우의 구조를 나타내었다. 이 구조는 데이타와 계수간의 곱셈 결과 N개를 m단에 걸쳐서 더하는 구조이다.

고정 소수점 연산에서는 모든 값이 -1과 1사이에 존재한다고 가정하므로 덧셈과 뺄셈에서 오버플로우를 방지하기 위한 스케일링 방법으로 모든 덧셈기 앞 단에서 오른쪽으로 한 비트 이동시킨다. 그러므로 이러한 스케일링에 의한 절단 오차는 각 덧셈기 마다 발생하며 모든 출력에 같은 크기의 영향을 준다. 그림 2에서 곱셈기 바로 다음의 덧셈기를 첫번째로 하면 i번째 단에는 $2^{(m-i+1)}$개의 오차원(error source)이 있고 이때에 발생한 오차는 $(m-i)$번의 스케일링을 하게 되어서 절단에 의한 오차의 평균 $\boldsymbol{tm}(N)$은

$$\boldsymbol{tm}(N)_k=\sum_{i=1}^{m}2^{(m+1-i)}\cdot\left(\frac{1}{2}\right)^{(m-i)}\cdot\mu_{T(1)},$$
$$k=0,\cdots,N-1$$

여기서 $\mu_{T(1)}$는 1비트 절단할 때 생기는 오차의 평균이다. 이 식을 정리하면

$$\boldsymbol{tm}(N)_k=2m\mu_{T(1)} \tag{8}$$

오차의 분산 $\boldsymbol{tv}(N)$의 경우는 스케일링시 전단 까지의 분산이 (1/4)씩 줄어들므로 오차의 평균의 경우와 같은 방법으로 전개하면 다음과 같다.

$$\boldsymbol{tv}(N)_k \sum_{i=1}^{m} 2^{(m+1-i)}(1/4)^{(m-i)} \cdot \sigma^2_{T(1)},$$

여기서 $\sigma^2_{T(1)}$는 1비트 절단할 때 생기는 오차의 분산이다. 이 식을 정리하면

$$\boldsymbol{tv}(N)=4(1-1/N)\sigma^2_{T(1)} \tag{9}$$

식 (8)과 (9)를 검토하면 데이타수 N이 증가함에 따라 오차의 평균은 로그함수적으로 증가하고 오차의 분산도 아주 느리게 증가함을 알 수 있다.

곱셈에 의한 반올림 오차는 직류 성분의 출력을 제외한 모든 출력에서 동일하게 나타나며 첫 단에서 발생하여 m단의 덧셈기를 통과하며 $1/\boldsymbol{N}$로 스케일링되어서 더해지므로 반올림에 의한 오차의 분산 $\boldsymbol{rv}(\boldsymbol{N})$은 다음과 같다.

$$\begin{aligned}\boldsymbol{rv}(\boldsymbol{N})_k &= \sum_{i=1}^{N}(1/4)^m \cdot \sigma^2_{R(b)},\\ &= \sigma^2_{R(b)}/N, \quad k\neq 0\end{aligned} \tag{10}$$

같은 방법으로 오차의 평균을 구하면 다음과 같다.

$$\begin{aligned}\boldsymbol{rm}(N)_k &= \sum_{i=1}^{N}(1/2)^m \cdot \mu_{R(b)},\\ &= \mu_{R(b)}, \quad k\neq 0\end{aligned} \tag{11}$$

그래서 식 (8-11)에 의하여 직접 구현방법의 오차를 분석하였다. 그 결과로 반올림에 의한 오차는 N의 증가에 따라 분산은 감소되고 평균은 일정함을 알 수 있다. 이러한 근거로 본 논문에서 제안하는 DCT의 직접 구현 방법이 고속 알고리듬들에 대한 적절한 비교 성능 평가의 대상이 된다고 할 수 있다.

3.2 Lee 알고리듬의 오차 분석

$\boldsymbol{N}$-point DCT가 $N/2$-point DCT 순환식으로 정의된 Lee알고리듬을 그림 3과 같다. 이 때에 출력 $\boldsymbol{y}(N)$은 비트 reverse 순서로 되어 있다. 이를 행렬 형태로 나타내면 다음과 같다.

$$\boldsymbol{y}(N)=\boldsymbol{T}(N)\boldsymbol{x}(N)/N, \tag{12a}$$

$$\begin{aligned}\boldsymbol{T}(N)=\Big[&\boldsymbol{T}(N/2)\\ &\boldsymbol{G}(N/2)\boldsymbol{T}(N/2)\boldsymbol{V}(N/2)\\ &\boldsymbol{T}(N/2)\boldsymbol{U}(N/2)\\ &-\boldsymbol{G}(N/2)\boldsymbol{T}(N/2)\boldsymbol{V}(N/2)\boldsymbol{U}(N/2)\Big]\end{aligned} \tag{12b}$$

여기서 $U(N)$는 역대각 단위 행렬이고, $\boldsymbol{G}(\boldsymbol{N})$ 및 $\boldsymbol{V}(\boldsymbol{N})$은 각각 다음과 같이 정의된다.

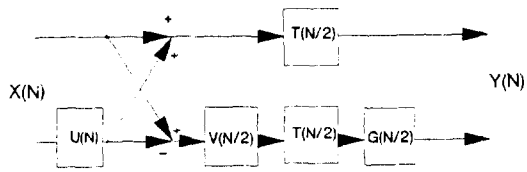

그림 3 Lee 알고리듬의 블럭선도
Fig. 3 A Block diagram of Lee algorithm

$$G(N)=\begin{bmatrix}1&1&\cdot&\cdot&0&0\\0&1&1&\cdot&\cdot&0\\0&\cdot&\cdot&\cdot&1&1\\0&0&\cdot&\cdot&0&1\end{bmatrix},$$

$$V(N)=\mathrm{diag}\{1/(2\cos\varphi_m)\},\ m=0,\cdots,N-1,$$

$$\varphi_m=(2m+1)\pi/(4N)$$

Lee 알고리듬의 특징은 식 (12)와 같은 순환식으로 간단하게 정의되며 N-point DCT의 경우에 $1/2N\log_2 N$번의 곱셈과 $(3/2N\log_2 N-N+1)$번의 덧셈이 필요하다. 그러나 행렬 $V(N)$에서 1보다 큰 계수를 곱해야 하기 때문에 계수를 곱하기 전에 데이타를 미리 scale하여 주어야 한다. $\boldsymbol{N}$-point Lee DCT의 경우에 항상 $\max\{V(N/2)_{i,j}\}<N/2$를 만족하고 덧셈기 앞에서 1/2로 scale하여 주어야 하므로 첫 단에서 1/N로 scale한다. 식 (3)의 DCT정의로부터 두번째 단 이후에는 더 이상의 scale이 필요하지 않음을 알 수 있다. 그러므로 두번째단 이후의 행렬 $V(\cdot)$의 곱셈에서는 계수가 1보다 큰 경우 계수를 1보다 작게 하여주고 그 대신 곱하는 데이타를 그만큼 크게 보상하여 준다. 이 때 보상하는 상수를 2의 배수로 하여 추가적인 하드웨어 없이 입력데이타를 왼쪽으로 이동시키는 구조를 갖게 한다.

다음에는 오차 분석에 앞서 평균과 분산이 각각 μ_x, σ_x^2로 일정하고 서로 상관관계가 없는 N개의 확률 변수들에 대한 DCT결과에 N을 곱한 — 스케일링을 하지 않은 — DCT출력 $y(i)$에 대한 평균값과 분산은 각각 다음과 같다.

$$E\{y(i)\}=N\delta(i)\mu_x, \tag{13}$$

$$\mathrm{Var}\{y(i)\}=(N/2)(1+\delta(i))\sigma_x^2 \tag{14}$$

식 (14)에서 직류 성분의 분산이 더 큰 이유는 본 논문에서 대상으로 하는 DCT가 정규화 되지 않은 식 (3)의 정의를 대상으로 하기 때문이다. 여기서 $\delta(i)$는 Dirac delta함수이다.

이 식을 이용하여 절단에 의한 오차를 분석하

면, 절단 오차는 첫번째 단에서만 발생하므로 이 때 발생한 오차를 N-point DCT변환된 것이 출력의 오차가 되므로 오차평균 벡터 $\boldsymbol{tm}(N)$과 오차분산 벡터 $\boldsymbol{tv}(N)$를 식 (13)과 (14)로 부터 구하면 다음과 같다.

$$\boldsymbol{tm}(N)=N\delta(i)\mu_{T(m)}, \tag{15}$$

$$\boldsymbol{tv}(N)=1/(2\mathrm{N})(1+\delta(i))\sigma^2_{T(m)} \tag{16}$$

여기서 $\mu_{T(m)}$과 $\sigma^2_{T(m)}$은 각각 m비트 이동시킬 때, 즉 $1/N$로 스케일링 할 때 발생하는 오차의 평균과 분산이다. 또한, 반올림 오차는 각 단마다 발생하는데 예를 들어 설명하면 첫단에서 발생하는 오차를 $N/2$-point DCT 한 결과에 $N/2$-point DCT 내부에서 발생한 오차와 더한 후에 행렬 $G(N/2)$와의 곱으로 나타낼 수 있다. 같은 방법으로 다음 단의 오차를 순환적으로 계산한다. 이러한 관계를 순환식으로 평균 $\boldsymbol{rm}(N)$에 대하여 정리하면 다음과 같다.

$$\boldsymbol{rm}(N)=(\boldsymbol{I}(N/2)\oplus\boldsymbol{G}(N/2))\boldsymbol{rmp}(N), \tag{17a}$$

여기서 $I(N)$은 크기 N인 단위 행렬이고, $\oplus$는 직접 합(direct sum)[10]이다.

$$\boldsymbol{rmp}(N)_i=\begin{cases}\boldsymbol{rm}(N/2)_i, & 0\le i<N/2,\\ \boldsymbol{rm}(N/2)_{i-N/2}+(N/2)\delta(i-N/2), & \\ \mu_{R(b)}, & N/2\le i<N,\end{cases} \tag{17b}$$

$$\boldsymbol{rv}(1)=[0]$$

여기서 $rmp(N)$은 행렬 $G(\cdot)$을 곱하기 전까지의 합으로 $N/2$-point DCT내부에서 발생한 오차 $rm(N/2)$과 행렬 $V(\cdot)$의 곱에서 발생한 오차를 $\frac{N}{2}$-point DCT한 결과인 $N(1+\delta(\cdot))/2$의 합으로 주어진다. 같은 방법으로 오차의 분산 $\boldsymbol{rv}(N)$은 식 (17b)의 둘째항의 $(N/2)\delta(i-N/2)\mu_{R(b)}$을 새로 더해지는 오차의 분산 $(N/4)(1+\delta(i-N/2))\sigma^2_{R(b)}$으로 대치하여 정리하면 다음과 같다.

$$\boldsymbol{rv}(N)=(I(N/2)\oplus\boldsymbol{G}(N/2))\boldsymbol{rvp}(N), \tag{18a}$$

$$\boldsymbol{rvp}(N)_i=\begin{cases}\boldsymbol{rv}(N/2)_i, & 0\le i<N/2,\\ \boldsymbol{rv}(N/2)_{i-N/2}+(N/4)(1+\delta & \\ (i-N/2))\sigma^2_{R(b)}, & N/2\le i<N,\end{cases} \tag{18b}$$

$$\boldsymbol{rv}(1)=[0].$$

3.3 Hou 알고리듬의 오차 분석

N-point DCT가 $N/2$-point DCT로 순환식으로 정의된 Hou DIF(decimation in frequency) 알고리듬을 그림 4에 도시하였다. Hou DIT(decimation in time) 알고리듬은 DIF알고리듬에 비하여 오차의 성능이 떨어지기 때문에 분석에서 제외시켰다. 이 때에 입력은 $\boldsymbol{x}(N)$은 짝수항이 먼저 오고 홀수항이 그 역순으로 오는 순서이고 출력 $\boldsymbol{y}(N)$은 비트 reverse 순서로 되어 있다. 이를 행렬 형태로 나타내면 다음과 같다.

$$\boldsymbol{y}(N)=\boldsymbol{T}(N)\boldsymbol{x}(N), \tag{19a}$$

$$\boldsymbol{T}(N)=\begin{bmatrix}1/2\,\boldsymbol{T}(N/2) & \\ 1/2\,\boldsymbol{K}(N/2)\boldsymbol{T}(N/2)\boldsymbol{Q}(N/2) & \\ 1/2\,\boldsymbol{T}(N/2) & \\ -1/2\,\boldsymbol{K}(N/2)\,T(N/2)\boldsymbol{Q}(N/2)\end{bmatrix}, \tag{19b}$$

여기서 $\boldsymbol{K}(N)=\boldsymbol{R}(N)\boldsymbol{L}(N)\boldsymbol{R}(N)$이고 이 때 $\boldsymbol{R}(N)$은 비트 reversal행렬이고 $\boldsymbol{L}(N)$은 삼각 행렬이고 $\boldsymbol{Q}(N)$은 대각 행렬로 다음과 같이 정의된다.

$$\boldsymbol{L}(N)=\begin{bmatrix}1 & 0 & \cdot & \cdot & 0 & 0\\ -1 & 2 & 0 & \cdot & \cdot & 0\\ 1 & -2 & 2 & \cdot & \cdot & 0\\ \cdot & \cdot & \cdot & \cdot & \cdot & 0\\ -1 & 2 & -2 & \cdot & \cdot & 2\end{bmatrix},$$

$$\boldsymbol{Q}(N)=\mathrm{Diag}\{\cos\theta_m\},\quad m=0,\cdots,N-1,$$

$$\theta_m=(4m+1)\pi/(2N)$$

이러한 Hou DIF알고리듬의 특징은 곱셈 수에 있어서 앞절의 Lee알고리듬과 동일하지만 이동과 덧셈으로 이루어진 행렬 $\boldsymbol{L}(\cdot)$의 영향으로 행렬 $\boldsymbol{K}(\cdot)$전단까지의 오차가 서로 섞이게 되어 결과적으로 N이 증가함에 따라 오차의 증가가 크게 된다. 그러나 Lee알고리듬과는 달리 행렬 $\boldsymbol{Q}(\cdot)$에 1 보다 큰 계수가 없기 때문에 스케일링 방법에서 유리하게 된다. Hou 알고리듬의 스케일링 방법은 직접 구현 방법과 마찬가지로 $\log_2 N$개의 단으로 이루어지고 각 단 마다 1/2로 scale하면 되므로 모든 덧셈기와 뺄셈기 앞단에서 오른쪽으로 한 비트 이동시킨다. 이러한 스케일링 방법은 Lee 알고리

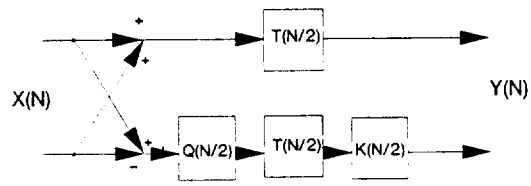

그림 4 Hou 알고리듬의 블럭선도
Fig. 4 A Block diagram of Hou algorithm

듬에서와 같이 처음에 $1/N$을 곱하는 방법보다 우수하다는 것이 FFT등의 다른 변환의 오차분석으로부터 잘 알려져 있다[2].

오차 분석에 앞서서 평균과 분산이 각각 μ_x, σ_x^2로 일정하고 서로 상관관계가 없는 N개의 확률변수들에 대한 DCT결과 $y(i)$에 대한 평균과 분산은 각각 다음과 같다.

$$E\{y(i)\}=\delta(i)\mu_x, \tag{20}$$

$$\mathrm{Var}\{y(i)\}=1/2(1+\delta(i))\sigma_x^2/N \tag{21}$$

우선, 평균 오차는 Lee알고리듬에서와 같은 방법을 이용하여 절단에 의한 오차 평균벡터 $\boldsymbol{tm}(N)$를 구하는 순환식을 정리하면

$$\boldsymbol{tm}(N)=\begin{bmatrix} I(N/2) & 0(N/2) \\ 0(N/2) & \boldsymbol{K}(N/2)\end{bmatrix}\begin{bmatrix}\boldsymbol{tm}(N/2)\\ \boldsymbol{tm}(N/2)\end{bmatrix} +[2\ 0\cdots 0]^T\mu_{T(1)}, \tag{22}$$

$$\boldsymbol{tm}(1)=[0]$$

같은 방법으로 반올림에 의한 오차 평균 벡터 $\boldsymbol{rm}(N)$를 구하는 순환식은 다음과 같이 정리된다.

$$\boldsymbol{rm}(N)=\begin{bmatrix}\boldsymbol{I}(1/2N) & 0(1/2N)\\ 0(1/2N) & \boldsymbol{K}(1/2N)\end{bmatrix}\cdot rmp(N), \tag{23a}$$

$$\boldsymbol{rmp}(N)_i=\begin{cases}\boldsymbol{rm}(N)_i, & 0\le i<N/2,\\ \boldsymbol{rm}(N)_{i-N/2}+1/2N(1+\delta(i-1/2N))\mu_{R(b)}, & N/2\le i<N,\end{cases} \tag{23b}$$

$$\boldsymbol{tm}(1)=[0].$$

다음에서 오차의 분산을 분석하였다. Lee알고리듬과는 달리 알고리듬 내부의 $L(L<N)$ point DCT에서 발생한 오차와 L-point DCT 전단에서 발생한 오차를 그 단에서 더하지 않고 그 각각을 독립적으로 계산하여 Lee 알고리듬의 분석보다는 복잡하지만 일반적인 방법을 이용하였다. 이러한 방법은 같은 i번째 단에서 발생한 독립적인 오차원 $rvp(N,\ m-i)$ 또는 $tvp(N,\ m-i)$이 출력 단까지 이르면서 곱해지는 행렬 $A(i)$의 각 원소를 제곱한 행렬 $B(i)$를 곱함으로써 i번째 단에 있는 오차원에 대한 출력에의 영향을 구하여 모든 단에서의 오차를 더하는 방법이다. 이것을 식으로 정리하면 다음과 같다.

$$\boldsymbol{tv}(N)=\boldsymbol{tvp}(N,\ m)+\sum_{i=1}^{(m-2)}B(i)\cdot\boldsymbol{tvp}(\boldsymbol{N},\ m-i-1), \tag{24a}$$

$$\boldsymbol{tvp}(N,\ k)_i=\boldsymbol{tvp}(\boldsymbol{N},\ 1/2k)(i_{\mathrm{mod}N/2}), \tag{24b}$$

$$\boldsymbol{tvp}(\boldsymbol{N},\ m)_i=2(1+\delta(i))\sigma^2{}_{T(1)}/\boldsymbol{N},\ \ \boldsymbol{N}=2^m \tag{24c}$$

위 식의 벡터 $tvp(N,\ m-i)$는 i번째 단에서 발생한 절단 오차의 분산이다.

$$\boldsymbol{rv}(\boldsymbol{N})=\sum_{i=1}^{(m-1)}B(i)\cdot\boldsymbol{rvp}(\boldsymbol{N},\ m-i), \tag{25a}$$

$$\boldsymbol{rvp}(\boldsymbol{N},\ \boldsymbol{k})_i=\boldsymbol{rvp}(\boldsymbol{N},\ 1/2\boldsymbol{k})_{(i\mathrm{mod}N/2)}, \tag{25b}$$

$$\boldsymbol{rvp}(\boldsymbol{N},\ m)_i=\begin{cases}0, & 0\le i<\boldsymbol{N}/2\\ (1+\delta(i-1/2\boldsymbol{N}))\sigma^2{}_{R(b)}/\boldsymbol{N}, & \boldsymbol{N}/2\le i<\boldsymbol{N},\end{cases} \tag{25c}$$

여기서 $S(N,\ k)$는 $k-1$번째 단에서 k번째 단으로의 행렬이므로 $A(i)$는 i번째 단에서 출력단 (m번째단)까지의 행렬, $S(\cdot)$의 곱이 된다. 행렬 B(i)는 분산 계산을 위하여 행렬 각 원소를 제곱한 형태이다. 그러므로 이들 행렬은 다음과 같이 정의할 수 있다.

$$\boldsymbol{A}(i)=\prod_{j=0}^{i}\boldsymbol{S}(\boldsymbol{N},\ m-j), \tag{26a}$$

$B(i)=A(i)\circledcirc A(i)$, ◎는 두 행렬에서 같은 위치의 항 끼리의 곱이다. 즉, $B=A\circledcirc C$는 $B_{i,j}=A_{i,j}\times B_{i,j}$로 정의된다.

$$\boldsymbol{S}(\boldsymbol{N},\ \boldsymbol{k})=\boldsymbol{S}(1/2\boldsymbol{N},\ \boldsymbol{k})\oplus\boldsymbol{S}(1/2\boldsymbol{N},\ \boldsymbol{k}), \tag{26b}$$

$$\boldsymbol{S}(\boldsymbol{N},\ \boldsymbol{m})=\boldsymbol{I}(1/2N)\oplus\boldsymbol{k}(1/2N),\ \ \boldsymbol{N}=2^m \tag{26c}$$

이러한 방법은 물론 오차 평균의 계산이나 다른 알고리듬에 모두 적용할 수 있는 오차를 분석하는 일반적인 방법이지만 복잡하기 때문에 Lee알고리듬에서는 사용할 필요가 없다.

3.4 Vetterli 알고리듬의 오차분석

Vetterli는 실수 데이타에 대한 고속 DFT (discrete fourier transform) 알고리듬과 고속 DCT 알고리듬을 제안하였다. 본 논문에서는 이중에서 고속 DCT 알고리듬에 대한 오차 분석을 하였다. 그 기본적인 구조는 그림 5와 같다.

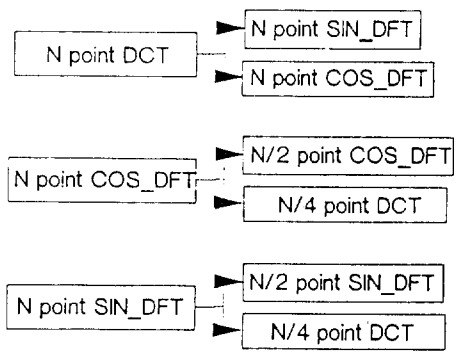

그림 5 Vetterli알고리듬의 블럭선도
Fig. 5 A Block diagram of Vetterli algorithm

N-point DFT결과의 실수 성분은 $N/2$-point DFT결과의 실수 성분과 $N/4$-point DCT결과의 합으로 표시되고 비슷한 방법으로 N-point DFT 결과의 허수 성분을 $N/2$-point DFT 결과의 허수 성분과 $N/4$-point DCT결과의 합으로 표시된다. Narasimha와 Peterson이 제안한[4] N-point DCT는 N-point DFT의 실수 성분과 허수 성분의 합으로 나타내는 알고리듬과의 결합으로 이루어진다. 이 관계를 정리하면 다음과 같다.

$$\mathrm{DCT}(k,\ N,\ x)=\cos\left[\frac{k\pi}{2N}\right]\mathrm{cos\_DFT}(k,N,x_4)-\sin\left[\frac{k\pi}{2N}\right]\mathrm{sin\_DFT}(k,N,x_4) \tag{27a}$$

$$\mathrm{cos\ DFT}(k,N,x)=1/2\mathrm{cos\_DFT}(k,N/2,x_1)+1/2\mathrm{DCT}(k,\mathrm{N}/4,x_2), \tag{28a}$$

$$\mathrm{sin\ DFT}(k,N,x)=1/2\mathrm{sin\_DFT}(k,N/2,x_1)+1/2\mathrm{DCT}(N/4-k,\ N/4,\ x_3), \tag{28b}$$

$$k=0,\cdots,N-1$$

여기서 $x_4(n)=x(2n)$,

$$x_4(N-n-1)=x(2n+1),\quad n=0,\cdots,N/2-1, \tag{29a}$$

$$x_1(n)=x(2n),\quad n=0,\cdots,n/2-1, \tag{29b}$$

$$x_2(n)=1/2x(2n+1)+1/2x(N-2n-1),\quad n=0,\cdots,N/4-1, \tag{29c}$$

$$x_3(n)=(-1)^n(1/2x(2n+1)-1/2x(N-2n-1)),\quad n=0,\cdots,N/4-1 \tag{29d}$$

실제 알고리듬에서는 곱셈수를 줄이기 위하여 식 (27)을 N이 $N/2$보다 큰 경우와 작은 경우로 나누어서 4번의 곱셈과 2번의 덧셈 대신 3번의 곱셈과 3번의 덧셈이 가능한 구조로 실현되지만 오차분석의 측면에서는 동일한 성능을 가지므로 식 (27)을 그대로 사용하였다. sin_DFT와 cos_DFT의 특징중에 하나가 변환 차수보다 변환 결과의 차수가 더 크다는 것인데 cos_DFT를 예로 설명하면, N-point cos_DFT를 계산하는데 $N/2$-point cos_DFT와 $N/4$-point DCT결과를 이용하므로 출력 인덱스 k가 $N/4$이상일 때에는 위의 식이 약간 변경되어야 한다.

Scaling방법은 식 (28)에 표시된 것과 같이 sin DFT와 cos_DFT의 계산에서 덧셈을 하기 전에 1/2로 나누고 식 (29a-d)에 표시된 것과 같이 DCT입력 x_4의 계산에서 미리 1/2로 나누어 입력하는 방법을 이용하였다. 따라서 절단오차는 cos_DFT와 sin_DFT에서만 발생되며 그 종류는 sin_DFT나 cos_DFT의 계산에서 발생하는 것과 DCT 입력 x_4의 계산에서 발생하는 것으로 나누어진다. 후자의 경우 오차가 발생한 후에 $N/4$-point DCT를 한 후에 더해지는 차이가 있다. 반면에 곱셈에 의한 반올림 오차는 DCT 계산식인 식 (27)에서만 발생하는 특징을 갖는다.

오차분석에 앞서서 평균과 분산이 각각 μ_x, σ_x^2로 일정하고 서로 상관 관계가 없는 $\boldsymbol{N}$개의 확률변수들에 대한 cos_DFT결과에 대한 평균과 분산을 각각 구하면 다음과 같다.

$$\boldsymbol{E}\{y(i)\}=\delta(i)\mu_x, \tag{30}$$

$$\mathrm{Var}\{y(i)\}=1/2(1+\delta(i)+\delta(i-\boldsymbol{N}/2))\ \sigma_x^2/\boldsymbol{N} \tag{31}$$

또한 같은 확률 분포를 갖는 x에 대하여 sin_DFT 결과에 대한 평균과 분산을 구하면

$$\boldsymbol{E}\{y(i)\}=0, \tag{32}$$

$$\mathrm{Var}\{y(i)\}=1/2(1-\delta(i)-\delta(i-\boldsymbol{N}/2))\ \sigma_x^2/N \tag{33}$$

끝으로 DCT결과에 대한 확률분포는 Hou 알고리듬의 분석에서 유도한 식 (20), (21)을 그대로 이용하였다.

우선, DCT에 대한 오차 분석을 하면 앞에서 언급한 바와 같이 Vetterli DCT 알고리듬에서는 반올림 오차만 발생함으로 절단 오차의 평균 $\boldsymbol{tm}(N)$은 식 (27)에서 sin_DFT의 절단 오차의 평균 $\boldsymbol{tsm}(N)$과 cos_DFT의 절단오차의 평균 $\boldsymbol{tcm}(N)$을 그대로 대입한 꼴이다. 분산인 경위에는 $\cos(\cdot)$이나 $\sin(\cdot)$를 곱하는 대신에 $\cos^2(\cdot)$이나 $\sin^2(\cdot)$을 곱하는 식이 되므로 이를 정리하면,

$$\boldsymbol{tm}(N_k)=\cos\left[\frac{k\pi}{2N}\right]\cdot\boldsymbol{tcm}(N)_k-\sin\left[\frac{k\pi}{2N}\right]\cdot\boldsymbol{tsm}(N)_k \tag{34}$$

$$\boldsymbol{tv}(N_k)=\cos^2\left[\frac{k\pi}{2N}\right]\cdot\boldsymbol{tcv}(N)_k-\sin^2\left[\frac{k\pi}{2N}\right]\cdot\boldsymbol{tsv}(N)_k \tag{35}$$

$$k=0,\cdots,N-1,\quad \boldsymbol{tm}(2)=[2\quad 0]^T\mu_{T(0)},\quad \boldsymbol{tv}(2)=[2\quad 1]^T\sigma^2_{T(0)}.$$

DCT에서의 반올림 오차의 분석은 위의 절단 오차를 분석하는 식에다 곱셈이 일어나는 횟수만큼 반올림 오차의 평균이나 분산을 더하는 형태가 된다. k가 영인 경우에는 곱셈이 없고 $N/2$인 경우에는 곱셈이 한번 있게 되므로 이를 식으로 정리하면

$$\boldsymbol{rm}(N_k)=\cos\left[\frac{k\pi}{2N}\right]\cdot\boldsymbol{tcm}(N)_k-\sin\left[\frac{k\pi}{2N}\right]\cdot \tag{34}$$

$$\boldsymbol{rsm}(N)_k + sd(k)\,\mu_{R(b)},$$

$$\boldsymbol{rv}(N_k) = \cos^2\left[\frac{k\pi}{2N}\right]\cdot \boldsymbol{tcv}(N)_k - \sin^1\left[\frac{k\pi}{2N}\right]\cdot \boldsymbol{rsv}(N)_k + sd(k)\,\sigma^2_{R(b)}, \quad (37)$$

$k=0,\cdots,N-1$, $\boldsymbol{rm}(2)=[0\ 1]^T\mu_{R(b)}$, $\boldsymbol{rv}(2)=[0\ 1]^T\sigma^2_{R(b)}$,

여기서 $sd(k)=(2-\delta(k-N/2))\cdot(1-\delta(k))$이다.

N-point cos_DFT의 오차 분석에서는 반올림 오차는 발생하지 않으므로 반올림 오차평균은 $N/2$-point cos_DFT의 오차 평균 $\boldsymbol{rcm}(N/2)$와 $N/4$-point DCT의 오차 평균을 1/2로 나눈 후에 더하고 분산도 같은 방법으로 $\boldsymbol{rcv}(N/2)$와 $\boldsymbol{rv}(N/4)$를 1/4로 나누어서 더한다. 이 관계를 식으로 표현하면

$$\boldsymbol{rcm}(N)_k = 1/2(\boldsymbol{rcm}(N/2)_{k2} + c(N,\ k)\boldsymbol{rm}(N/4)_{k3}), \quad (38)$$

$$\boldsymbol{rcv}(N)_k = 1/4(\boldsymbol{rcv}(N/2)_{k2} + \boldsymbol{rv}(N/4)_{k3}), \quad (39)$$

$k=0,\cdots,N-1$, $\boldsymbol{rcm}(2)=[0\ \ 0]^T$, $\boldsymbol{rcv}(2)=[0\ \ 0]^T$

여기서 $c(N,\ k)$와 k_2, k_3는 표 2와 같다.

다음으로 N-point cos_DFT의 절단 오차 분석을 하였는데 인덱스 k의 변화에 따라서 절단 오차의 발생 방법이 다양하게 변하였다. k가 $N/4$에서 $3N/4$ 사이인 경우 즉 $c(N,\ k)$가 -1인 경우에는 식 (28b)에서 덧셈 대신에 뺄셈을 하여야 하므로 이 때에는 절단 오차의 평균이 영이된다. 또한 $N/4$-point DCT의 입력인 $x_3(n)$의 스케일링으로 발생한 절단 오차가 k_3가 $N/4$인 경우에는 DCT$(N,\ N,\ x)$가 영이므로 이 경우를 제외시켜야 한다. 이를 식으로 정리하면

$$\boldsymbol{tcm}(N)_k = 1/2(\boldsymbol{tcm}(N/2)_{k2} + c(N,\ k)\boldsymbol{tm}(N/4)_{k3}) + (1+c(N,\ k)(1+\delta(k_3)) - c(N,\ k)\delta(k_3-N_4))\mu_{T(1)}, \quad (40)$$

$$\boldsymbol{tcv}(N)_k = 1/4(\boldsymbol{tcv}(N/2)_{k2} + \boldsymbol{tv}(N/4)_{k3}) + ((1+\delta(k+3))/N + 2 - \delta(k_3-N/4))\,\sigma^2_{T(1)}, \quad (41)$$

$\boldsymbol{tcm}(2)=[2\ \ 0]^T\mu_{T(1)}$, $\boldsymbol{tcv}(2)=[2\ \ 2]^T\sigma^2_{T(1)}$

표 1 $c(N,\ k)$, $s(N,\ k)$, k_2, k_3, k_4의 정의
Table 1 The definition of $c(N,\ k)$, $s(N,\ k)$, k_2, k_3, k_4

k의 범위	$c(N,k)$	$s(N,k)$	k_2	k_3	k_4
$0\le k<N/4$	1	1	k	k	$N/4-k$
$N/4\le k\le N/2$	-1	1	k	$N/2-k$	$k-N/4$
$N/2\le k<3N/4$	-1	-1	$k-N/2$	$k-N/2$	$3N/4-k$
$3N/4\le k<N$	1	-1	$k-N/2$	$N-k$	$k-3N/4$

또한 N-point sin_DFT의 오차 분석을 하였다. sin_DFT의 계산에서도 역시 반올림 오차는 발생하지 않으므로 반올림 오차 평균은 $N/2$ point sin_DFT의 오차 평균 $\boldsymbol{rsm}(N/2)$와 $N/4$-point DCT의 오차 평균을 1/2로 나눈 후에 더하고 분산은 같은 방법으로 $\boldsymbol{rsv}(N/2)$와 $\boldsymbol{rv}(N/4)$를 1/4로 나누어서 더하였다. 이 관계를 식으로 표현하면

$$\boldsymbol{rsm}(N)_k = 1/2(\boldsymbol{rsm}(N/2)_{k2} + s(N,\ k)\boldsymbol{rm}(N/4)_{k4}), \quad (42)$$

$$\boldsymbol{rsv}(N)_k = 1/4(\boldsymbol{rsm}(N/2)_{k2} + \boldsymbol{rv}(N/4)_{k4}) \quad (43)$$

$\boldsymbol{rsm}(2)=[0\ \ 0]^T$, $\boldsymbol{rsv}(2)=[0\ \ 0]^T$

여기서 $s(N,\ k)$와 k_2, k_4는 표 1과 같다.

마지막으로 N-point sin_DFT의 절단 오차 분석을 하였다. cos_DFT의 경우와 마찬가지로 출력 인덱스 k의 변화에 따라서 절단 오차의 발생 방법이 다양하게 변한다. 즉 $s(N,\ k)$가 -1일때에는 절단오차의 평균은 영이 되었고 $N/4$-point DCT의 입력인 $x_4(n)$의 스케일링으로 발생한 절단 오차 평균은 절단 오차간의 뺄셈이므로 항상 영이된다. 또한 cos_DFT의 경우에서와 같이 k_4가 $N/4$인 경우에는 DCT$(N,\ N,\ x)$가 영이므로 이 경우를 제외 시켜야 한다. 이를 식으로 정리하면

$$\boldsymbol{tsm}(N)_k = 1/2(\boldsymbol{tsm}(N/2)_{k2} + s(N,\ k)\boldsymbol{tm}(N/4)_{k3}) + (1+s(N,\ k))(1-\delta(k_2)) - \delta(k_2-N/4))\mu_{T(1)}, \quad (44)$$

$$\boldsymbol{tsv}(N)_k = 1/4(\boldsymbol{tsv}(N/2)_{k2} + \boldsymbol{tv}(N/4)_{k3}) (2(1-\delta(k_2)) + (1-(N-1)\ \delta(k-N/4))/N)\sigma^2_{T(1)},$$

$\boldsymbol{tsm}(2)=[0\ \ 0]^T$, $\boldsymbol{tsv}(2)=[0\ \ 0]^T$

4. 모의 실험 결과

3장에서 유도한 오차의 평균과 분산의 타당성을 검토하기 위하여 각 알고리듬에 대하여 모의 실험을 하였다. 본 연구에서 정의하는 고정 소수점 연산에 의한 N point DCT의 오차 벡터 $\boldsymbol{err}(N)$, 오차의 평균 벡터 $\boldsymbol{em}(N)$, 분산 벡터 $\boldsymbol{ev}(N)$, 전력(power) 벡터 $\boldsymbol{ep}(N)$은 다음과 같다.

$$\boldsymbol{err}(N) = [\boldsymbol{y}(N)]_Q - \boldsymbol{y}(N), \quad (46)$$

$$\boldsymbol{em}(N) = \boldsymbol{E}\{\boldsymbol{err}(N)\} = \boldsymbol{rm}(N) + \boldsymbol{tm}(N), \quad (47)$$

$$\boldsymbol{ev}(N) = \mathrm{Var}\{\boldsymbol{err}(N)\} = \boldsymbol{rv}(N) + \boldsymbol{tv}(N), \quad (48)$$

$$\boldsymbol{ep}(N)_i = E\{\boldsymbol{err}(N)_i^2\} = \boldsymbol{em}(N)_i^2 + \boldsymbol{ev}(N)_i \quad (49)$$

또한 각 오차 벡터들의 인덱스 변화에 대한 평균은 끝에 알파벳 "a"를 덧붙여서 표시하였다. 예를

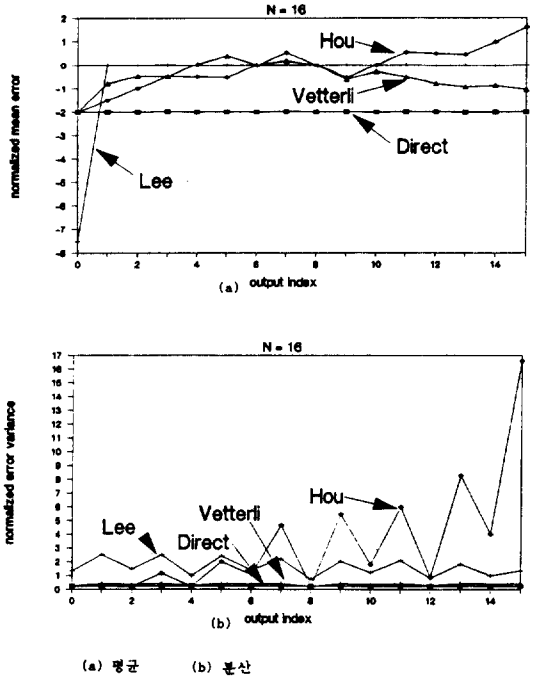

그림 6 $N=16$인 경우의 오차 평균과 분산
(a) 평균 (b) 분산

Fig. 6 The mean and variance of error in case of $N=16$

들어 오차 전력 벡터의 평균을 epa라 하고 다음과 같이 정의한다.

$$epa=\frac{1}{N}\sum_{i=0}^{N-1} ep(N)_i \tag{50}$$

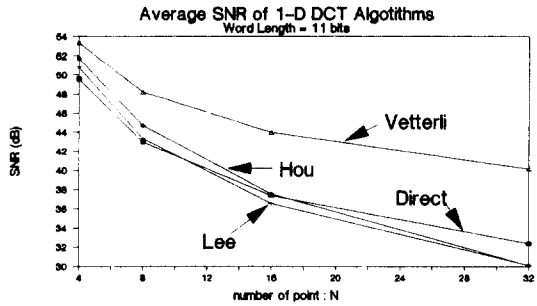

그림 7 DCT차수의 변화에 대한 신호대 잡음비의 변화

Fig. 7 A SNR as a function of DCT order

모의 실험은 DCT차수를 4, 8, 16, 32로 변화시키면서 서로 다른 10,000개의 입력데이타에 대하여 수행하였다. 이 때에 입력 데이타의 폭과 연산결과의 폭은 10비트로 하였고 기준이 되는 출력은 64비트 부동소수점으로 하였다. 그 결과 출력 인덱스의 변화에 따른 오차의 특성을 알기 위하여 DCT차수 16에 대한 오차의 평균과 분산을 그림 6(a), (b)에 나타내었다. 이 때에 오차의 평균은 데이타 폭에 무관하도록 양자화 폭 $Q(=2^{-b})$로 나누어준 값이고 오차의 분산도 같은 이유로 그 값을 Q^2으로 나눈 값이다. DCT차수가 변할 때 각 알고리듬에 대한 평균 신호대 잡음비 SNRA 그림 7에 나타내었다. 이 때 신호대 잡음비, $SNR(N)_i$과 평균 신호대 잡음비 SNRA는 각각 다음과 같은식 (51), (52)으로 정의된다.

표 2 모의 실험과 오차 분석 결과의 비교

Table 2 A comparison of computer simulation and analysis result

		4-point		8-point		16-point		32-point	
		mean	var	mean	var	mean	var	mean	var
drt	sim	−0.999	0.201	−1.498	0.226	−1.998	0.240	−2.500	0.245
	anl	−1.000	0.203	−1.500	0.228	−2.000	0.239	−2.500	0.245
lee	sim	−0.381	0.342	−1.444	0.753	−0.460	1.622	−0.494	3.462
	anl	−0.374	0.341	−0.436	0.755	−0.467	1.617	−0.481	3.408
hou	sim	−0.255	0.357	−0.187	1.063	−1.123	3.316	−0.079	9.955
	anl	−0.250	0.357	−0.187	1.071	−0.125	3.294	−0.078	10.026
vtr	sim	−0.371	0.222	−0.452	0.300	−0.505	0.359	−0.545	0.398
	anl	−0.365	0.225	−0.449	0.298	−0.505	0.355	−0.544	0.400

$$\mathrm{SNR}(N)_i = 10\log_{10}\left[\frac{xp(N)_i}{ep(N)_i}\right], \tag{51}$$

$$\mathrm{SNRA} = 10\log_{10}\left[\frac{xpa}{epa}\right], \tag{52}$$

여기서 출력 신호의 전력 $xp(N)_i$를 구하기 위해서는 신호의 DCT 변환후의 전력을 구하여야한다. 입력 신호 $x(n)$가 -1과 1사이에 균일하게 분포하므로 입력 신호의 전력은 그 신호의 분산인 1/3이 된다. 이러한 신호에 대한 DCT후의 분산은 식 (20)을 이용하여 직류성분의 출력 전력은 $2/N$이고 그 이외의 성분은 $1/N$임을 쉽게 알 수 있다. 그러므로 출력 신호의 인덱스 변화에 대한 평균, xpa는 $(N+1)/(3N)$임을 알 수 있다.

모의 실험 결과 3장에서 분석한 오차의 예상과 거의 같음을 알 수 있었는데 DCT차수를 변화시키면서 각 차수에서의 오차의 평균과 분산에 대한 실험치와 이론치를 비교하여 나타내면 표 2과 같다. 그 결과는 2장에서의 오차에 대한 가정과 3장에서의 분석 방법이 타당함이 입증된다.

신호대 잡음비의 측면에서 분석 결과를 살펴보면, 모든 차수에서 Vetterli 알고리듬의 오차가 직접 구현법과 Lee와 Hou 알고리듬에 비하여 오차가

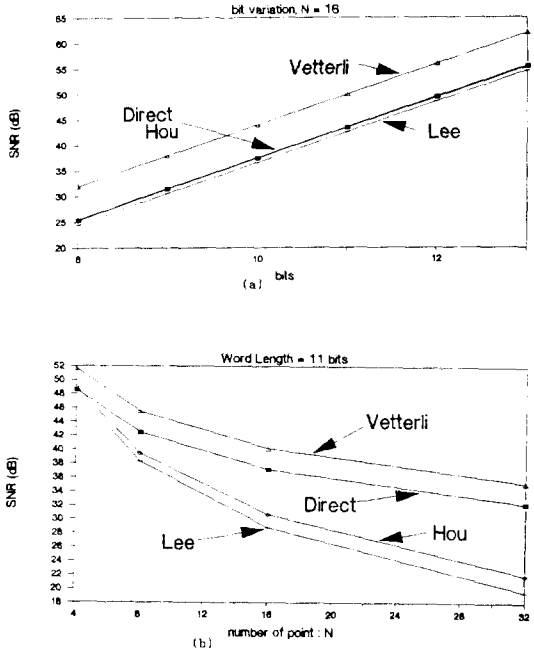

그림 8 데이타 폭을 변화시키는 경우에 신호대 잡음비의 변화
(a) 평균 SNR (b) 최소 SNR

Fig. 8 A SNR as a function of data width

작은 것을 알 수 있다. Hou의 알고리듬은 평균 신호대 잡음비가 DCT차수 N이 16보다 작은 경우에는 직접 구현법이나 Lee알고리듬 보다 크지만 16-point 이상에서는 직접 구현방법의 신호대 잡음비가 더 커지고 N이 32보다 커지는 경우에는 Lee 알고리듬의 신호대 잡음비가 더 커짐을 알 수 있다. 이것은 Hou 알고리듬은 DCT 차수가 증가함에 따라 오차가 급격히 커지는 특징을 갖는데 이는 3장에서 언급한 바와 같이 행렬 $\boldsymbol{K}(\cdot)$에서 전단에서 발생한 오차를 많은 출력으로 전하기 때문이다. 그러나 Hou의 알고리듬은 영상 코딩에서 중요시하는 저주파 성분 즉, 낮은 인덱스에서 오차가 작다는 장점이 있다. 그 밖에 모든 고속 알고리듬의 경우 인덱스가 홀수인 경우가 짝수인 경우에 비하여 오차의 분산이 더 큰 특징이 있는데 이것은 홀수항에서 반올림에 의한 오차, 즉 곱셈에 의한 오차가 많기 때문이다. 또한, 최소 신호대 잡음비의 경우에서는 그 크기가 Vetterli, 직접 구현법, Hou, Lee 알고리듬의 순인데 직접 구현법은 평균 신호대 잡음비에 대하여 큰 변화가 없었다. 차수의 증가에 대하여 신호대 잡음비의 감소는 약 6dB 정도였는데 이는 차수 증가에 따른 잡음비의 감소를 보상하기 위해서는 내부 데이타의 크기를 1비트 증가시키면 된다는 것을 뜻한다.

오차 평균의 특징을 보면, 절단에 의한 영향이 반올림에 의한 영향에 비하여 절대적으로 크다. 알고리듬별로 그 특징을 보면, 직접 구현 방법은 인덱스의 변화에 대하여 균일하고 DCT차수의 증가에 대하여 로그함수적으로 증가하고, Vetterli 알고리듬은 오차 평균이 전반적으로 가장 작으면서 차수 증가에 따른 변화도 가장 작고, Lee알고리듬의 경우는 스케일링을 처음에 한번만 하기 때문에 직류 성분에만 큰 오차가 있으며, Hou알고리듬의 경우는 인덱스가 증가함에 따라 음에서 양으로 바뀌는 특징이있다.

알고리듬 오차의 성능 평가에서 오차의 평균보다는 오차의 분산이 더 비중이 크다고 할 수 있는데, 이는 오차의 평균은 일종의 출력 신호에 대한 바이어스(bias)이므로 변환 후에 적절히 처리할 수 있기 때문이다. 오차 분산의 특징을 살펴보면 모든 차수에 대하여 직접 구현법이 그 크기가 가장 작고 그 다음 Vetterli, Lee, Hou알고리듬의 순이다. Hou알고리듬은 역시 N의 증가에 대하여 오차의 증가가 크고 Vetterli알고리듬과 직접 구현법이 증가가 작음을 알 수 있다.

끝으로 데이타 폭 b의 변화에 따른 신호대 잡음비의 변화를 조사한 결과를 그림 8에 제시하였다.

이론적으로는 식 (4-7)과 식 (51)에서 데이타 폭 1비트의 증가에 따라 SNR이 $(20\log_{10}2)=6.02$dB씩 증가하는데 실험 결과도 이에 일치하였다.

5. 결 론

본 연구에서 DCT알고리듬의 고정 소수점 오차를 분석하는 간단하고 새로운 기법을 DCT에 적용하여서 DCT의 직접 구현방법과 고속 알고리듬중 Hou와 Lee, Vetterli의 알고리듬을 분석하였다. 분석 결과 오차 분산의 측면에서는 직접 구현방법이, 오차평균의 측면에서는 Vetterli 알고리듬이 좋았으나 신호대 잡음비의 측면에서 볼 때에 Verterli 알고리듬이 오차가 가장 적었음을 알 수 있었다. 그러나 Vetterli 알고리듬의 경우에 알고리듬 자체가 복잡하다는 단점이 있다. Lee알고리듬은 알고리듬 중에 1보다 큰 계수를 곱하게 되어 스케일링 측면에서 불리하므로 이에 대한 적절한 하드웨어 구조와 직류 성분 오차에 대한 사후 처리과정이 필요하다. Hou알고리듬은 차수의 증가에 불리하므로 작은 차수에서만 사용하는 것이 좋다. Hou 알고리듬의 분석에 이용된 방법은 다른 알고리듬의 분석에 비하여 복잡하지만 순환식으로 정의되는 모든 변환의 분석에 이용될 수 있다. 물론 직접 구현방법도 오차 성능도 좋고 그 구조가 간단하지만 곱셈의 수가 많으므로 실제 구현하는 데는 불리하다. 이 결과로 영상 코딩등에 이용할 DCT chip의 설계에서 고속 알고리듬의 구현을 고려할 수 있을 것이다. 결론적으로 본 논문에서의 오차 분석은 어떤 알고리듬을 하드웨어로 구현할 때 필요한 최소의 내부 데이타 폭을 구할 수 있으므로 최적의 하드웨어 설계에 이용될 수 있을 것이다.

참 고 문 헌

[1] N. Ahmed, T. Natarajan and K.R. Rao, "Discrete cosine transform," *IEEE Trans. Commun.*, vol. COM-23, pp. 90~93, Jan. 1974.

[2] Alan V. Oppenheim and R. W. Schafer, *Digital Siginal Processing*. Prentice Hall, 1985.

[3] W.H. Chen, C.H. Smith and S.C. Fralick, "A fast computational algorithm for discrete cosine transform," *IEEE Trans. Commun.*, vol. COM-25, pp. 1004~1009, Nov. 1977.

[4] M.J. Narashimha and A.M. Peterson, "On the computation of the discrete cosine transform," *IEEE Trans. Commun.*, vol. COM-26, pp. 934~936, June, 1978.

[5] N.I. Cho and S.U. Lee, "DCT Algorithms for VLSI Parallel mplementations," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-28, pp. 121~127, Jan. 1990.

[6] B.G. Lee, "A new algorithm to compute the discrete cosine transform," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-32, pp. 1243~1245, Dec. 1984.

[7] H.S. Hou, "A fast recursive algorithms for comptuing the discrete cosine transform," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-35, No. 10, pp. 1455~1461, Oct., 1987.

[8] M. Vetterli and H. Nussbaumer, "Simple FFT and DCT Algorithms with Reduced number of operations," *Signal Processing*, vol. 6, no. 4, July 1984.

[9] C.W. Barnes and B.T. Tran and S.H. Leung, "On the Statistics of Fixed-Point Roundoff Error," *IEEE Trans. Acoust., Speech, Signal Processing. vol. ASSP-33, pp. 595~606, June.* 1985.

[10] *K.R. Rao and P. Yip, Discrete Consine Transform Algorithms, Advantages, Applications*, Academic Press, 1990.