한국음향학회지 제26권 제5호 pp. 214~219 (2007)

AAC 디코더의 IMDCT를 위한 고속 IFFT 알고리즘

A Fast IFFT Algorithm for IMDCT of AAC Decoder

지 화 준*, 김 태 훈*, 박 주 성*
(Hua-jun Chi*, Tae-hoon Kim*, Ju-sung Park*)
*부산대학교 전자전기통신공학부

(접수일자: 2007년 5월 22일, 수정일자: 2007년 7월 6일, 채택일자: 2007년 7월 23일)

본 논문은 MPEG-2 AAC (Advanced Audio Coding) 디코더에 필요한 IMDCT (Inverse Modified Discrete Cosine Transform)를 고속으로 처리하기 위한 새로운 IFFT (Inverse Fast Fourier Transform) 구현 방식을 제안한다. 기존 방식 중에서 2^n (N-point) type IMDCT가 성능이 가장 우수하지만 많은 계산을 요구하는 N/4-point complex IFFT 과정을 포함하고 있다. 본 연구는 2^n (N-point) type IMDCT에 포함된 N/4-point complex IFFT의 연산량을 줄이는 방법을 고안하였다. N/4-point complex IFFT는 입력 데이터를 bit-reverse 방식을 사용하여 정렬하지만 본 연구에서는 새로운 입력 데이터 정렬방식과 $N/4^{n+1}$ 형태의 IFFT 고안하여 곱셈, 덧셈, ROM 용량을 줄였다.

핵심용어: IFFT, 비트 리버스, 디지털 오디오, IMDCT

투고분야: 전기 음향 분야 (3.2)

This paper proposes a new IFFT (Inverse Fast Fourier Transform) algorithm, which is proper for IMDCT (Inverse Modified Discrete Cosine Transform) of MPEG 2 AAC (Advanced Audio Coding) decoder. The 2^n (N-point) type IMDCT is the most powerful among many IMDCT algorithms, however it includes IFFT that requires many calculation cycles. The IFFT used in 2^n (N-point) type IMDCT employ the bit-reverse data arrangement of inputs and N/4-point complex IFFT to reduce the calculation cycles. We devised a new data arrangement method of IFFT input and $N/4^{n+1}$-type IFFT and thus we can reduce multiplication cycles, addition cycles, and ROM size.

Key words: Fast IFFT, Bit-reverse, Digital audio, IMDCT

ASK subject classification: Electro-Acoustics (3.2)

I. 서 론

MPEG-2 AAC는 "MPEG-2 Advance Audio Coding"의 약자로, backward compatibility를 없애고, 적은 비트율로 고음질의 사운드 데이터를 부호화 (encoding)와 복호화 (decoding)하는 ISO 13818-7 규격이다. [1] AAC는 MP3 (MPEG-1 layer 3)와 마찬가지로 인간의 귀 특성을 반영한 심리음향 모델을 이용하는 transform coding 방식이다. [2] MPEG-2 AAC 디코더의 입력신호는 주파수 영역의 신호이므로 시간영역의 신호로 변환하는 과정이 반드시 필요하다. 주파수 영역의 데이터를 시간영역의 데이터로 변환하기 위하여 IMDCT 기법이 많이 사용되고 있다.[3]

IMDCT를 구현하는 방식으로는 데이터 재정렬 방식, type-II DCT 방식, type-IV DCT 방식, 2^n (N-point) type IMDCT 등과 같은 방식이 있다. [4-6] 본 논문에서는 MPEG-2 AAC 디코더에 사용되는 IMDCT 구현을 위하여 다양한 방식을 분석하여 효율적인 구현방법을 고안하였다. 본 논문은 2장에서 AAC 디코더 알고리즘에 대해서 설명하고, 3장에서는 종래의 IMDCT 방식을 분석하고, 4장에서는 새로운 고속 IMDCT 방식을 소개한다. 5장에서는 기존 방식과 비교분석을 하였으며 마지막으로 6장에서 결론을 맺는다.

책임저자: 박 주 성 (juspark@pusan.ac.kr)
부산시 금정구 장전동 산30 부산대학교 전자공학과
(전화:051-510-2444; 팩스: 051-515-5190)

II. AAC 디코더 알고리즘

AAC는 main, low complexity (LC), scale-able sampling rate (SSR)와 같은 3종의 프로파일을 가진다. [1] 본 논문은 휴대오디오기기에 많이 응용되는 LC 프로파일의 복호화를 위한 IMDCT에 관한 내용을 다룬다.

AAC LC의 디코더 과정을 그림 1에 나타내었다. MPEG-2 AAC 부호화기는 시간영역의 데이터를 심리음향 모델을 이용하여 여러 주파수 밴드 신호로 변환하여 압축한 후 허프만 부호화를 통하여 또 다시 압축을 한다.

AAC 디코더는 엔코더의 역과정을 수행해야 하므로 입력 비트열을 허프만 디코딩한 후 역양자화를 수행한다. 역양자화 된 샘플을 허프만 디코더 과정에서 얻은 스케일 팩터로 스케일링하고 시간영역 잡음 변형을 수행한 후 필터뱅크를 거쳐 시간영역의 신호로 변환된다.

MPEG-2 AAC 디코더의 마지막 과정인 필터뱅크는 주파수 영역신호를 시간영역의 신호로 변환시키는 과정인 IMDCT와 windowing & block switching, 중첩덧셈 과정으로 구성되며 최종적으로 이득을 조절함으로써 시간영역의 신호를 얻는다. AAC 디코더에서 허프만 디코더와 필터 뱅크의 연산량이 전체의 80% 이상 차지하고 있고 IMDCT의 연산량은 전체 연산량의 28.4%을 차지한다. [7]

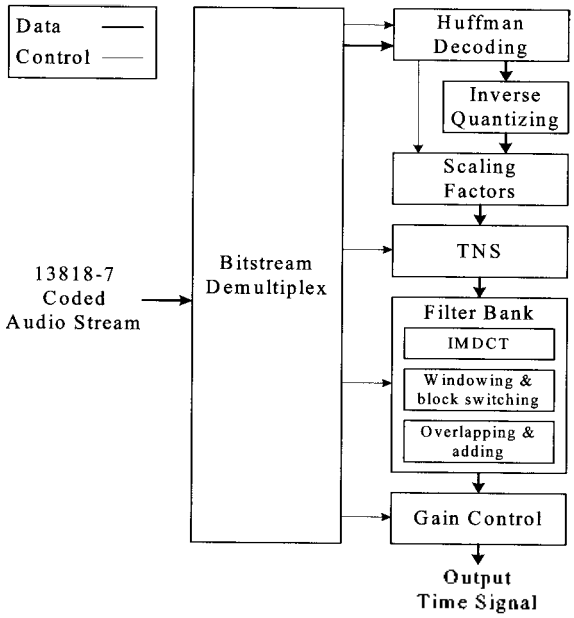


그림 1. AAC LC 프로파일의 디코딩 순서
Fig. 1. Decoding flow of AAC LC profile.

III. IMDCT 알고리즘 분석

3.1. 데이터 재정렬을 이용한 알고리즘

주파수 영역의 신호를 시간영역으로 변환하기 위한 IMDCT는 식 (1)과 같이 표시된다. [8]

$$x[n]=\sum_{k=0}^{\frac{N}{2}-1}X[k]\cos\left[\frac{2\pi}{N}(n+p)\left(k+\frac{1}{2}\right)\right],\quad 0\le n<N \tag{1}$$

여기서 X[k]는 spectral coefficient, n는 샘플 인덱스, k는 spectral 계수 인덱스, N은 윈도우 길이, p=(N/2+1)/2을 나타낸다. 식 (1)의 결과인 시간 영역 샘플 x[n]은 대칭적인 특성이 있으므로 N-point의 x[n]을 구할 필요가 없이 N/2-point의 x[n]만 구하여 결과를 재정렬하여 사용하면 되므로 연산량이 반으로 줄어들게 된다.

3.2. DCT를 적용한 알고리즘

DCT 방식의 대표적인 알고리즘인 Type-II 형태의 N-point DCT는 식 (2)와 같이 나타낼 수 있는데 이 식에서 x[n]과 X[k]이 어떤 성분을 나타내는 가에 따라 DCT 또는 IDCT가 결정된다.

$$x[n]=\sum_{k=0}^{N-1}X[k]\cos\left(\frac{\pi}{2N}(2k+1)n\right),\quad 0\le n<N \tag{2}$$

식 (2)을 변형하여 x[n]의 짝수성분 (C[2n])과 홀수성분 (D[2n+1])을 구하기 위하여 식 (3a)과 식 (3b)과 같이 정의한다.

$$C[n]=x[2n] \tag{3a}$$

$$D[n]=x[2n+1],\ 0\le n<N/2 \tag{3b}$$

짝수성분은 식 (2)과 식 (3a)을 이용하여 식 (4)과 같이 나타낼 수 있다.

$$\begin{aligned}C[n]&=\sum_{k=0}^{N-1}X[k]\cos\left(\frac{\pi}{2N}(2k+1)\times 2n\right)\\&=\sum_{k=0}^{\frac{N}{2}-1}P[k]\cos\left(\frac{\pi}{2\times(N/2)}(2k+1)n\right)\end{aligned} \tag{4}$$

여기서 P[k]=X[k]+X[N-1-k] 이다. 또한 x[n]의 홀수 성분을 이용하여 D'[n]을 식 (5)과 같이 정의한다. 식 (3b)과 식 (5)을 이용하여 D'[n]을 구하면 식 (6)과 같다.

$$D'[n] = D[n] + D[n-1] \tag{5}$$

$$\begin{aligned} D'[n] &= \sum_{k=0}^{N-1} X[k]\cos\left(\frac{\pi}{2N}(2k+1)(2n+1)\right) \\ &\quad + \sum_{k=0}^{N-1} X[k]\cos\left(\frac{\pi}{2N}(2k+1)(2n-1)\right) \\ &= \sum_{k=0}^{\frac{N}{2}-1} 2Q[k]\cos\left(\frac{\pi}{2\times(N/2)}(2k+1)n\right) \end{aligned} \tag{6}$$

여기서

$$Q[k] = \sum_{k=0}^{\frac{N}{2}-1} 2\{X[k] - X[N-1-k]\}\cos\left(\frac{\pi}{2N}(2k+1)\right) \tag{7}$$

이다.

한편 D(0) = D(-1)으로 정의하면 식 (5)에 의하여 D(0)=D'[0]/2가 되고 IDCT x[n]의 홀수성분 (D[n])은 D'[n] - D[n-1] 로 나타낼 수 있다. 홀수 성분의 경우 D'[n]과 D[n-1]을 recursive하게 사용하여 구할 수 있다. 따라서 N-point Type-II DCT는 2개의 N/2-point Type-II DCT를 이용하여 구현할 수 있다.

3.3. 2^n (N-point) type IMDCT 방식

2^n (N-point) type 방식은 식 (1)을 식 (8)과 같이 변형한 것이다. [6]

$$\begin{aligned} x[n] &= \sum_{k=0}^{N-1} X[k]\times\cos\left(\frac{\pi}{2N}(2k+1)(2n+1+\frac{N}{2})\right) \\ &= -\sum_{k=0}^{\frac{N}{4}-1} X[2k]\times e^{j\frac{\pi}{2N}(4k+1)(4n+1)} + j\sum_{k=0}^{\frac{N}{4}-1} X\left[\frac{N}{2}-2k-1\right]\times e^{j\frac{\pi}{2N}(4k+1)(4n+1)} \\ &= \left[\sum_{k=0}^{\frac{N}{4}-1}\left\{\left(-X[2k]+jX\left[\frac{N}{2}-2k-1\right]\right)\times e^{j\frac{2\pi}{N}(k+\frac{1}{8})}\right\}\times e^{j\frac{2\pi}{N/4}nk}\right]\times e^{j\frac{2\pi}{N}\left(n+\frac{1}{8}\right)} \end{aligned} \tag{8}$$

이 방식은 X[k]의 2개의 샘플 데이터를 이용하여 -X[2k]+ jX[N/2-2k-1] 형태의 복소수를 만든다. 여기서 k가 odd이면 허수부로, even 이면 실수부를 구성하게 된다. 이렇게 만들어진 복소수를 pre-rotation 시킨 후 bit-reverse 시켜 N/4-point complex IFFT를 수행한다. 이렇게 계산된 결과를 post-rotation하여 N/4 개의 complex 데이터를 구하고 이 데이터를 이용하여 de-interleaving을 수행하면 시간영역의 N-point 데이터 x[n]을 얻을 수 있다. 2^n (N-point) type IMDCT 에서 N/4-point complex IFFT 과정이 전체 연산량의 70% 정도 차지하고 있다.

IV. 제안하는 고속 IMDCT 방식

4.1. 기본 개념

제안하는 방식은 2^n (N-point) type IMDCT의 N/4-point complex IFFT를 다른 형태로 수행하는데 그 수식은 먼저 식 (9)과 같이 짝수와 홀수 항으로 나눈 다음 다시 4의 배수 성분으로 분해 하여 식 (10)과 같이 재배열한다.

$$\begin{aligned} x[n] &= \sum_{k=0}^{N-1} X[k]W_N^{nk}, \quad 0 \le n < N \\ &= x[2n] + x[2n+1], \quad 0 \le n < N/2 \\ &= x[4n] + x[4n+2] + x[4n+1] + x[4n+3], \; 0 \le n < N/4 \end{aligned} \tag{9}$$

$$\begin{aligned} x[n] &= \sum_{k=0}^{N/4-1} H_1[k] + H_2[k]W_N^{2k} + H_3[k]W_N^{k} + H_4[k]W_N^{3k} \\ x[n+\frac{N}{4}] &= \sum_{k=0}^{N/4-1} H_1[k] + H_2[k]W_N^{2k} - H_3[k]W_N^{k} - H_4[k]W_N^{3k} \\ x[n+\frac{N}{2}] &= \sum_{k=0}^{N/4-1} H_1[k] - H_2[k]W_N^{2k} + H_3[k]W_N^{k}W_N^{N/4} - H_4[k]W_N^{3k}W_N^{N/4} \\ x[n+\frac{3N}{4}] &= \sum_{k=0}^{N/4-1} H_1[k] - H_2[k]W_N^{2k} - H_3[k]W_N^{k}W_N^{N/4} + H_4[k]W_N^{3k}W_N^{N/4} \end{aligned}$$

$$.\; 0 \le n < N/4 \tag{10}$$

여기서

$$\begin{aligned} H_1[k] &= \sum_{k=0}^{N/4-1} X[4k]W_{N/4}^{nk}, \qquad H_2[k] = \sum_{k=0}^{N/4-1} X[4k+2]W_{N/4}^{nk} \\ H_3[k] &= \sum_{k=0}^{N/4-1} X[4k+1]W_{N/4}^{nk}, \quad H_4[k] = \sum_{k=0}^{N/4-1} X[4k+3]W_{N/4}^{nk} \end{aligned} \tag{11}$$

이다.

제안하는 방식은 먼저 식 (11)과 같이 N/16-point의 $H_1[k]$, $H_2[k]$, $H_3[k]$, $H_4[k]$를 먼저 구한 다음 IFFT EXPAND 구조를 이용하여 N/4-point의 x[n]을 구하게 된다. 또한 제안하는 방식은 식 (10), (11)과 같이 변형된 IFFT의 계산량을 줄이기 위하여 bit reverse 대신 표 1, 2와 같은 형태로 입력을 정렬한다. AAC 디코더에 필요한 complex 64, 512 포인터 IFFT의 경우 기본단위 IFFT가 각각 16, 32 포인터 이므로 표 1, 2와 같이 배열한다. 표 1, 2는 pre-rotation의 출력 (PR[ ])이 IFFT의 어떤 입력 (IF[ ])에 인가되는 가를 설명하고 있다.

표 1. 64-point IFFT 입력정렬 방식
Table 1. Input data arrangement for 64-point IFFT.

m = 0, 1, 2, ··· ··· , 14	m = 15
IF[16 * 0 + m] = PR[4m+0] IF[16 * 1 + m] = PR[4m+1] IF[16 * 2 + m] = PR[4m+2] IF[16 * 3 + m+1] = PR[4m+3]	IF[16 * 0 + m] = PR[4m+0] IF[16 * 1 + m] = PR[4m+1] IF[16 * 2 + m] = PR[4m+2] IF[16 * 2 + m+1] = PR[4m+3]

표 2. 512-point IFFT 입력정렬 방식
Table 2. Input data arrangement for 512-point IFFT.

m = 0, 1, 2, ··· ··· , 30	m = 31
IF[32 * 0 + m] = PR[16m+0] IF[32 * 4 + m] = PR[16m+1] IF[32 * 8 + m] = PR[16m+2] IF[32 * 13 + m] = PR[16m+3] IF[32 * 1 + m] = PR[16m+4] IF[32 * 5 + m] = PR[16m+5] IF[32 * 9 + m] = PR[16m+6] IF[32 * 14 + m] = PR[16m+7] IF[32 * 2 + m] = PR[16m+8] IF[32 * 6 + m] = PR[16m+9] IF[32 * 10 + m] = PR[16m+10] IF[32 * 15 + m+1] = PR[16m+11] IF[32 * 3 + m+1] = PR[16m+12] IF[32 * 7 + m+1] = PR[16m+13] IF[32 * 11 + m+1] = PR[16m+14] IF[32 * 12 + m+1] = PR[16m+15]	IF[32 * 0 + m] = PR[16m+0] IF[32 * 4 + m] = PR[16m+1] IF[32 * 8 + m] = PR[16m+2] IF[32 * 13 + m] = PR[16m+3] IF[32 * 1 + m] = PR[16m+4] IF[32 * 5 + m] = PR[16m+5] IF[32 * 9 + m] = PR[16m+6] IF[32 * 14 + m] = PR[16m+7] IF[32 * 2 + m] = PR[16m+8] IF[32 * 6 + m] = PR[16m+9] IF[32 * 10 + m] = PR[16m+10] IF[32 * 14 + m+1] = PR[16m+11] IF[32 * 2 + m+1] = PR[16m+12] IF[32 * 6 + m+1] = PR[16m+13] IF[32 * 10 + m+1] = PR[16m+14] IF[32 * 11 + m+1] = PR[16m+15]

새로운 방식으로 정렬된 데이터와 기본단위 IFFT (M-point 라 가정)를 구한 후, IFFT의 샘플 포인터 수를 4xM, 16xM point 형태로 확장해간다. 기본단위 IFFT는 $M=N/4^{n+1}$ 식으로 부터 구한다. 이 식에서 N은 IMDCT point 수, M은 16 또는 32인 정수이고, n은 데이터 포인터를 확장해야 할 stage 수를 나타낸다.

4.2. 고안된 방식의 구현

AAC 디코더의 2048-point IMDCT의 경우에는 32-point가 기본 단위 IFFT 가 되며 그 입력은 표 2와 같이 정렬된 것을 사용한다. 고안한 입력 정렬방식에 맞는 32-point IFFT 수행 과정은 그림 2와 그림3과 같다. 그림 2, 3 및 본 논문에서 twiddle factor의 표시 방법은 W n m 형식으로 표시하며 그 의미는 W_n_m = EXP (jπ m/n)와

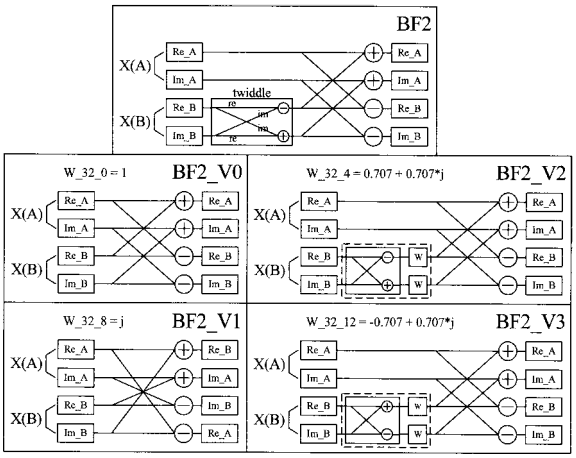


그림 2. IFFT_32 블록의 기능
Fig. 2. Functions of blocks used in IFFT 32.

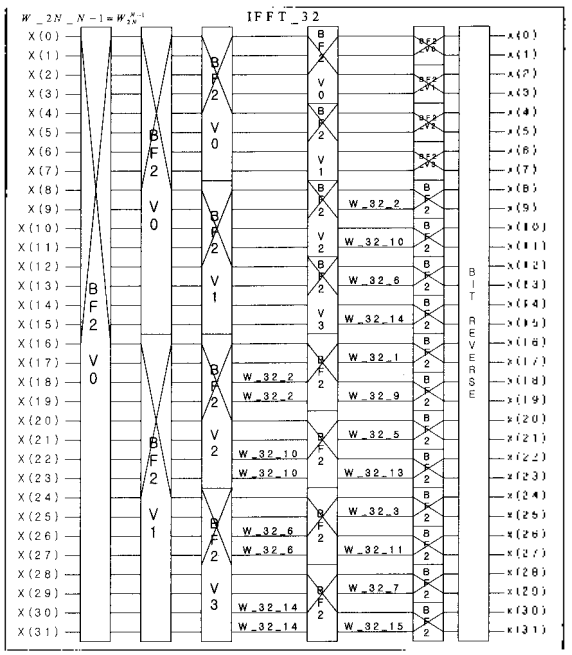


그림 3. IFFT 32 블록 구조
Fig. 3 Architecture of IFFT_32 block.

같다. IFFT의 데이터 포인터를 확장하는 EXPAND의 구조는 그림 4와 같으며, 2쌍의 동일한 데이터 포인터를 받아들이기 때문에 IFFT 포인트 수에 상관없이 N/16- point complex IFFT를 받아 N/4-point complex IFFT를 만들 수 있다.

IFFT EXPAND는 식 (11)을 이용하여 구한 N/16-point IFFT에 해당되는 $H_1[k]$, $H_2[k]$, $H_3[k]$, $H_4[k]$에 식 (10)과 같이 적절한 twiddle factor와 곱셈, 덧셈 연산을 통하여 N/4 point IFFT를 수행한다.

AAC 디코더에 필요한 2,048 point IMDCT를 구현하기 위해서 필요한 512-point IFFT는 4개의 32-point IFFT (IFFT_32 블록)의 결과를 IFFT EXPAND를 거쳐 128-point IFFT를 만든 다음 다시 IFFT EXPAND를 거치면

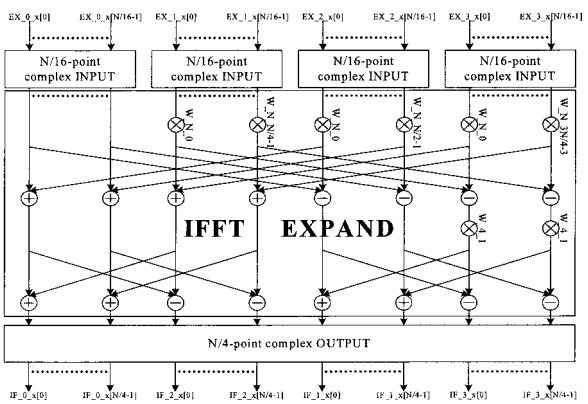


그림 4. IFFT EXPAND 블록 구조
Fig. 4 Architecture of IFFT EXPAND.

된다. 256-point IMDCT의 경우는 16-point IFFT를 기본 단위로 하여 1번의 IFFT EXPAND를 시키면 된다.

V. 비교 분석

여러 방식의 IMDCT 성능을 곱셈 횟수(N_M), 덧셈 횟수(N_A), twiddle factor를 저장하는 ROM 용량 (N_{ROM}), temporary data를 저장하는 RAM 용량 (N_{RAM}) 측면에서 비교한다. 3장에서 소개된 여러 방식의 성능을 비교해 보면 표 3과 같다.

표 3. 기존 방식의 특성[4][6]
Table 3. Characteristics of conventional algorithm[4][6].

항목	재정렬[4]	DCT[6]	2^n(N-point)
N_M	$N^2/4$	$N/2*\log_2 N$	$N/2*\log_2 N/4+2N$
N_A	$N^2/4-N/2$	$3N/4*(2*\log_2 N-1)$	$3N/4*\log_2 N/4+N$
N_{ROM}	$N^2/4$	$N/2*\log_2 N$	$N/4*\log_2 N/4+N$
N_{RAM}	$N^2/4+N$	$2*(3N/2*\log_2 N-N)$	$N*\log_2 N/4+4N$

(주) N은 IMDCT data point 를 의미함

새로운 방식의 연산량을 2048-point IMDCT의 경우를 적용한 예를 들어 분석한다. IFFT_32블록은 그림 3과 같은 구조를 가지므로 곱셈 횟수는 $[(32/2)*\log_2(32) - [(16+8+4+2+1) + (8+4+2+1) + (4+2+1)]]*4*16 = 1,728$ 이다. 여기서 (16+8+4+2+1)*4, (8+4+2+1)*4, (4+2+1)*2, (4+2+1)*2은 각각 BF_V0, BF_V1, BF V2, BF_V3을 사용함으로 해서 연산량이 감소되는 부분이다. 덧셈 횟수는 $[(32/2)*\log_2(32)*6 - [(16+8+4+2+1) + (8+4+2+1)]*2]*16 = 6,208$ 이다. 여기서 $[(16+8+4+2+1) + (8+4+2+1)]*2$은 새로운 형태의 IFFT를 사용함으로 해서 연산량이 감소되는 부분이다.

IFFT EXPAND블록은 그림 4와 같은 구조를 가지는데 이 블록을 실행할 때 곱셈 횟수는 (32*4+128)*12 = 3,072 이고, 덧셈 횟수는 (32*4+128)*22 = 5,632 이다. IFFT_32 블록은 2^n (N-point) type IMDCT의 IFFT 보다 곱해야 할 twiddle factor 수가 작아 곱셈과 덧셈 연산을 각각 66.3%와 19.2%만큼 줄일 수 있다. IFFT EXPAND 블록은 기존의 radix-2 방식의 IFFT에 비하여 곱셈과 덧셈 연산을 25%와 8.3% 만큼 줄일 수 있다. 이는 2^n (N-point) type IMDCT 경우는 radix-2 포맷인 N/4-point IFFT 를 2-stage를 거치게 되어 N*4*4 만큼의 곱셈과 N*6*4 만큼의 덧셈 횟수를 필요로 하지만, IFFT EXPAND는 N*12 만큼의 곱셈과 N*22 의 덧셈만 필요로 하기 때문이다.

표 4. 제안하는 방식의 특성
Table 4. Characteristics of the proposed algorithm.

항목	pre/post rotation	IFFT	De-interl eaving	Total
N_M	N	S1+S2		S1+S2+2N
N_A	N/2	P1+P2		P1+P2+N
N_{ROM}	N/2	(S1+S2)/2		(S1+S2)/2+N
N_{ROM}	N	$N*\log_2 N/4$	2N	$N*\log_2 N/4+4N$

제안된 방식을 사용할 경우 RAM 용량은 $2,048*\log_2(256)$, ROM 용량은 (1728+3,072)/2=2,400 이다. 256, 2048-point IMDCT의 연산량, RAM, ROM 용량를 일반화 하여 표 4에 나타내었다.

(주) S1, S2, P1, P2 는 아래와 같다

$$S1=\begin{cases}\frac{N}{2}\log_2(N/64)-\frac{7}{4}N+192, & N=2048\\ \frac{N}{2}\log_2(N/16)\ 25\times16, & N=256\end{cases}\qquad S2=\begin{cases}(N/8)\times12=3N/2, & N=2048\\ (N/16)\times12=3N/4, & N=256\end{cases}$$

$$P1=\begin{cases}\frac{3N}{4}\log_2(N/64)-92\times16, & N=2048\\ \frac{3N}{4}\log_2(N/16)\ 44\times4, & N=256\end{cases}\qquad P2=\begin{cases}(N/8)\times22=11N/4, & N=2048\\ (N/16)\times22=11N/8, & N=256\end{cases}$$

표 3, 4에 의하면 2048-point IMDCT 경우 새로운 방식이 2^n (N-point) type IMDCT 방식보다 곱셈, 덧셈, ROM 용량을 각각 33.2%, 12.5%, 33.2% 만큼 감소시킬 수 있다. 256-point인 경우는 곱셈, 덧셈, ROM에서 각각 36.3%, 14.8%, 36.3% 만큼 줄일 수 있다.

새로운 IMDCT 방식을 이용하여 AAC 디코더를 구현하여 정상적으로 동작하는 것을 확인하였다. 24bit DSP을 이용하여 10초 분량의 AAC 데이터를 2^n (N-point) type과 제안하는 방식으로 IMDCT를 구현했을 때 1 프레임 당 평균 연산량을 표 5에 비교하였다.

표 5. 2048-point AAC 디코더 사이클 수 비교
Table 5. Comparison of calculation cycles for 2048-point AAC decoder.

Block	Proposed IMDCT	2^n(N-point) IMDCT
Huffman decoder	418,830	418,830
Inverse quantizer	100,410	100,410
Scale factors	24,378	24,378
Filter bank	423,572	470,580
Others	53,325	53,325
Total	1,020,515	1,067,523

(주) AAC 1 frame 평균 처리 사이클 수

AAC 디코더 전체적인 연산량 측면에서 보면 새로운 방식이 2^n (N-point) type IMDCT의 경우의 95.6 %가 되었으며 필터 뱅크는 90.0%가 되었다. IMDCT와 IFFT에서 곱셈과 덧셈이 차지하는 비중이 data load와 store보다

상대적으로 낮아 덧셈이나 곱셈이 줄어든 비율에 비하여 전체적인 연산량 개선효과가 작게 나왔다.

VI. 결 론

본 논문에서 제안하는 방식은 기존방식에서 가장 효율적인 2^n (N-point) type IMDCT에서 사용되는 IFFT 과정을 개량한 것이다. 기존 IFFT에서 사용하는 bit- reverse와 N/4-point complex IFFT 대신 새로운 데이터 정렬방식, $N/4^{n+1}$-IFFT, IFFT EXPAND를 고안하여 twiddle factor와 곱셈, 덧셈, ROM 용량을 줄일 수 있었다.

새로운 방식을 이용하여 AAC 디코더에 필요한 256-point IMDCT를 구현했을 경우는 2^n (N-point) type IMDCT 방식보다 곱셈에서 36.3%, 덧셈에서 14.8%, ROM 용량에서 36.3% 만큼 감소시킬 수 있었다. 2048-point 경우에는 곱셈에서 33.2%, 덧셈에서 12.5%, ROM 용량에서 33.2% 만큼 감소시킬 수 있었다. 새로운 방식의 IMDCT를 채택함으로써 AAC 디코더의 전체 연산량에서 4.4% 개선이 있었다.

본 논문에서 제안하는 IFFT 입력 재정렬 방식과 $N/4^{n+1}$-IFFT를 이용하여 N-point IFFT를 구현하는 방식은 AAC 디코더가 아닌 다른 분야에서도 응용 가능할 것으로 생각된다.

감사의 글

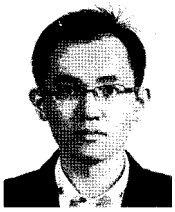
이 논문은 부산대학교 자유과제 학술연구비 (2년)에 의하여 연구되었음

참 고 문 헌

1. ISO/IEC IS 13818-7, "Information Technology - Generic Coding of Moving Pictures and Associated Audio, Part7: Advanced Audio Coding, AAC," 1997.
2. John Gordon, "Psychoacoustics.", in John Strawn, Curtis Abbott, John Gordon, and Philip Greespun, eds. The Computer Music Tutorial, The MIT Press, Cambridge, Massachusetts, 1053-1068, 1998
3. J. P. Princen and A. B. Bradley, "Analysis/Synthesis Filter Bank Design Based on Time Domain Aliasing Cancellation," IEEE Trans. on ASSP-34, (5) 1986, 1153-1161.
4. Mu-Huo Cheng and Yu-Hsin Hsu, "Fast IMDCT and MDCT Algorithms A Matrix Approach", IEEE Trans. on Signal Processing 51 (1) Jan. 2003.
5. Che-Hong Chen, Bin-Da Liu, Jar-Ferr Yang, and Jiun-Lung Wang "Efficient Recursive Structures for Forward and Inverse Discrete Cosine Transform", IEEE Trans. on Signal Processing 52 (9) Sept. 2004.
6. Che-Hong Chen, Bin-Da Liu, and Jar-Ferr Yang, "Recursive Architectures for Realizing Modified Discrete Cosine Transform and Its Inverse", IEEE Trans. on Circuits and system-II: Analog and Digital Signal Processing, 50 (1) Jan. 2003
7. Do-Hee Kim "The research on configuration of Fixed-point MPEG-2 AAC Decoder with 24 bit DSP core", Pusan National University, Master Thesis, Feb. 2006.
8. Davis Yen Pan, "Digital Audio Compression", Digital Tech. Journal, 5 (2) 1993.

저자 약력

• **지 화 준 (Hua-Jun Chi)**



2005년 2월 : 연변과학기술대학교 학사
2007년 2월 : 부산대학교 전자공학과 석사

• **김 태 훈 (Tae-Hoon Kim)**

1995년 2월 : 연변과학기술대학교 학사
1997년 2월 : 부산대학교 전자공학과 석사
2002년 2월 : 부산대학교 전자공학과 박사

• **박 주 성 (Ju-Sung Park)**

1976년 2월 : 부산대학교 전자공학과 학사
1978년 6월 : KAIST 전자공학과 석사
1978년~1985 : ETRI 실장
1989년 6월 : Univ. of Florida 전자공학 박사
1991년~현재 : 부산대학교 전자공학과 교수