덧셈과 쉬프트 연산을 사용한 MP3 IMDCT의 저전력 Systolic 구조

정회원 장 영 범*, 준회원 이 원 상*

A low-power systolic structure for MP3 IMDCT using addition and shift operation

Young-Beom Jang* *Regular Member*, Won-Sang Lee* *Associate Member*

요 약

이 논문에서는 MP3에 사용되는 32-point IMDCT 블록의 저전력 hard-wired 구조를 제안하였다 행렬의 재배열을 통하여 16, 8, 4, 2, 1 cycle에 동작하는 5개의 multirate block을 유도함으로서 저전력 systolic 구조를 제안하였다 각각의 sub-block들의 곱셈 구현은 덧셈기와 쉬프트로 구현하는 CSD(Canonic signed digit) 방식을 채택하여 덧셈의 수를 줄임으로서 전력소모를 감소시켰다 또한 각각의 sub-block들의 전력소모를 더욱 감소시키기 위하여 common sub-expression sharing 방식을 채용함으로서 덧셈의 연산량을 더욱 감소시킨 구조를 제안하였다 그 결과, 2의 보수형을 사용하는 구조와 비교하여 58 4%의 상대 전력소모를 줄일 수 있었다. 또한 하드웨어 구현을 Verilog-HDL 코딩을 통하여 시뮬레이션 함으로서 구조가 정확하게 동작함을 확인하였다.

Key Words MP3, MPEG audio; IMDCT, CSD, common subexpression sharing; systolic

ABSTRACT

In this paper, a low-power 32-point IMDCT structure is proposed for MP3 Through re-odering of IMDCT matrices, we propose the systolic structure operating with 16, 8, 4, 2, and 1 cycle, respectively To reduce power consumption, multiplication of each sub blocks are implemented by add and shift operation with CSD(Canonic signed digit) form coefficients To reduce, furthermore, the number of adders, we utilize the common sub-expression sharing techniques With these techniques, the relative power consumption of the proposed structure is reduced by 58.4% comparison to the conventional structure using only 2's complement form coefficient. Validity of the proposed structure is proved through Verilog-HDL coding

I 서 론

최근 인터넷의 발달로 오디오 파일의 인터넷 다운로드가 쉬어지면서 MP3 플레이어의 시장이 매우 빠르게 확대되고 있다. MP3는 MPEG1 audio layer III의 알고리즘을 사용하여 오디오를 부호화 또는 복호화한다[1] 이와 같은 MP3는 주로 DSP 프로세서를 사용하여 알고리즘을 코딩한 후에 ASIC으로 IC를 제작하는 것이 보편적인 방법이었다 그러나, 최근에는 DSP 프로세서를 사용하지 않고 hard wired로 구현하는 반도체 IC 제품이 출현하고 있다. MP3 복호화 과정에서 가장 많은 연산을 차지하는 부분이 IMDCT(Inverse Modified Discrete Cosine Transform)이다. 비디오 복호기에서는 DCT (Descrete cosine

* 상명대학교 정보통신공학전공(ybjang@smu ac kr)
논문번호 040062-0209, 접수일자 2004년 2월 10일

transform)와 IDCT를 사용하는데 반하여 MPEG audio에서는 MDCT와 IMDCT를 사용한다. 이는 변형된 DCT를 사용하기 때문이지만 기본적인 연산 방법은 거의 같다. 따라서 효율적인 MDCT 구조를 설계하기 위하여 DCT의 구조를 살펴볼 필요가 있다[2]. DCT는 쓰임새가 많아지면서 많은 고속 알고리듬이 연구되었다[3] [4]. 이와 같은 고속 알고리듬들은 DCT가 가지고 있는 수학적 성질을 이용하여 계산량을 감소시킨다. 또한 비디오 압축과 복원에 사용되는 표준들에서는 실시간 처리의 요구를 만족시키기 위한 DCT의 효과적인 VLSI 구조가 연구되었다[5]. 이와 같은 하드웨어 구조는 고속처리가 가능한 장점은 있지만 구현비용이 프로세서를 사용하는 구조보다 상대적으로 크다. VLSI의 발달로 프로세서의 속도가 빨라짐에 따라 DCT를 프로세서를 사용하여 구현하는 연구가 또한 진행되고 있다[6]. 그러나, 이와 같은 프로세서를 사용한 구현은 하드웨어는 줄일 수 있지만 실시간으로 처리되기 위해서는 프로세서의 고속 동작이 필수적이다. 따라서 지금까지 열거한 하드웨어 구조와 프로세서를 사용한 구조의 장점 만을 취하기 위하여 하이브리드 구조가 연구되고 있다[7]. 하이브리드 구조를 사용한 [7]에서는 곱셈기를 내장한 프로세서를 사용하지 않고 AU(Arithmetic Unit), 즉 덧셈기 1개만을 내장한 프로세서를 사용하여 2-D 8×8 DCT를 구현하기 위하여 1208개의 덧셈을 실행한다. 그러나 [7]의 방식은 한 개의 AU 프로세서를 사용하여 DCT를 수행하므로 거의 프로세서 방식에 가깝다. DCT와 더불어 디지털 필터의 VLSI 구현에서도 곱셈연산이 가장 중요한 변수가 된다. 대부분의 디지털 필터의 곱셈은 덧셈기와 쉬프트 연산을 사용하여 구현되며, 덧셈의 수를 감소시키기 위하여 CSD(Canonic Signed Digit) 형의 필터계수를 사용하고있다[8]. 이와 더불어 덧셈의 수를 더욱 감소하는 방법으로서 CSD형의 필터계수에서 공통패턴을 공유하는 방법이 연구되었다[9]-[10]. 본 논문에서는 필터 구현에 이용되는 CSD 형 계수의 공통패턴 공유기법을 이용하여 저전력 IMDCT 블록을 제안하고자 한다. 본 논문의 II장에서는 IMDCT 행렬 분해에 의한 mult-rate IMDCT 구조를 제안하며, III장에서는 II장에서 제안된 구조의 내부 블록들을 덧셈과 쉬프트 연산만을 사용하여 구현하는 구조를 제안한다. IV장에서는 시뮬레이션을 통하여 IC의 면적과 상대 전력 소모를 비교한다.

II. 곱셈 연산의 수를 최소화한 Systolic IMDCT 구조 설계

MPEG audio에서 사용되는 IMDCT의 행렬은 다음의 식으로 만들어진다.

$$\cos\left[\frac{\pi}{64}(2k+1)(r+16)\right] \quad k=0, \quad ,31, r=0,\cdots,63 \tag{1}$$

따라서 위의 식을 사용하여 IMDCT 변환을 32×64의 행렬로 나타내면 다음 식과 같다.

$$\begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_{63} \end{bmatrix} = \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,32} \\ a_{2,1} & a_{2,2} & a_{2,3} & \cdots & a_{2,32} \\ a_{3,1} & a_{3,2} & a_{3,3} & \cdots & a_{3,32} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{64,1} & a_{64,2} & a_{64,3} & \cdots & a_{64,32} \end{bmatrix} \begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ \vdots \\ X_{31} \end{bmatrix} \tag{2}$$

식(2)의 연산은 32×64의 행렬이므로 2048개의 곱셈연산이 필요하다. 이 절에서는 2048개의 곱셈연산을 342개로 감소시키며 또한 곱셈연산을 systolic 방식으로 처리하는 구조를 제안한다.

1단계로 행의 대칭성을 이용하면 2048개의 곱셈연산을 1024개의 곱셈연산으로 감소시킬 수 있다. 먼저 식 (2)의 행들을 살펴보면 17행을 기준으로 아래와 위의 행들이 부호만 반대이며 모두 대칭을 이루고 있다. 즉, 16행과 18행, 15행과 19행, 14행과 20행, …, 1행과 33행이 부호만 반대이고 같다. 또한 34행부터 64행에서도 대칭을 이루고 있다. 즉, 49행을 기준으로 48행과 50행, 47행과 51행, 46행과 52행, …, 34행과 64행이 같다. 따라서 이와 같은 행의 중복성을 제거하면 IMDCT 변환은 다음과 같이 32×32의 행렬로 모든 계산이 이루어진다.

$$\begin{bmatrix} x_{17} \\ x_{18} \\ x_{19} \\ \vdots \\ x_{48} \end{bmatrix} = \begin{bmatrix} a_{18,1} & a_{18\,2} & a_{18,3} & \cdots & a_{18,32} \\ a_{19,1} & a_{19,2} & a_{19,3} & \cdots & a_{19,32} \\ a_{20,1} & a_{20,2} & a_{20,3} & \cdots & a_{20,32} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{49,1} & a_{49,2} & a_{49,3} & \cdots & a_{49\ 32} \end{bmatrix} \begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ \vdots \\ X_{31} \end{bmatrix} \tag{3}$$

위의 식을 사용하여 x_{17}부터 x_{48}을 구하면 이를 사용하여 나머지 IMDCT 출력들은 추가의 계산 없이 간단히 구할 수 있다. 즉 출력 x_0부터 x_{16}은 출력 x_{17}부터 x_{32}에 -를 취해주면 구할 수 있고, x_{49}부터 x_{63}은 x_{33}부터 x_{48}까지의 출력과 같으므로 부가적인 계산이 필요 없다. 따라서 식(3)을 이용하면 32×32의 행렬 연산이므로 1024개의 곱셈으로 모든 IMDCT 출력을 구할 수 있다.

2단계로 식(3)의 행렬에서 열의 대칭성을 이용하면 곱셈의 수를 1024개에서 다시 342개로 감소시킬 수 있다 식(3) 행렬에서 각각의 행들을 살펴보면 행 내에서도 element들이 열 대칭을 이루고 있는 것을 발견할 수 있다. 즉, 1열과 32열, 2열과 31열, 3열과 31열, …, 16열과 17열이 부호를 제외하고 대칭을 이루고 있다. 따라서 32×32의 행렬은 2개의 16×16의 행렬로 분해된다. 그 중에서 다시 1개의 16×16의 행렬은 2개의 8×8의 행렬로 분해된다 이와 같이 행렬의 분해를 통하여 6개의 행렬 연산 블록이 만들어지며, 각 블록을 계산하기 위하여 필요한 곱셈의 수는 256+64+16+4+1+1 =342개가 된다. 이와 같이 1단계와 2단계의 행렬의 행과 열의 대칭성을 이용하여 2048개의 곱셈을 342개로 감소시켜 83 3%의 곱셈감소 효과를 얻을 수 있다 3단계로 342개의 곱셈연산을 병렬처리와 pipeline 처리가 가능한 systolic 구조를 제안한다 행렬 연산이 순차적으로 수행되도록 하기 위하여 행렬의 element 들의 위치를 조정하여 입력신호의 순서대로 계산하는 방법이다 이와 같은 방법으로 제안한 구조는 그림 1과 같다

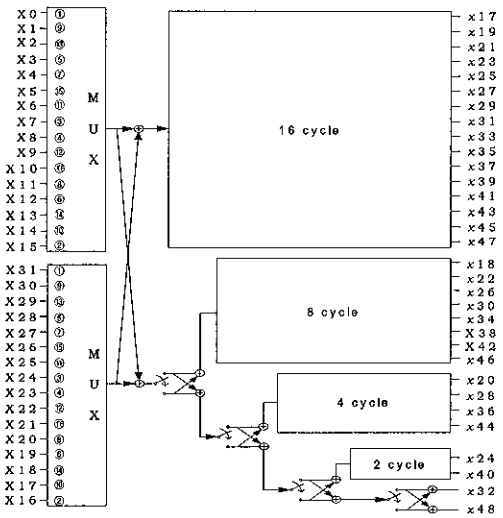


그림 1 행과 열의 대칭을 이용한 IMDCT 구조

그림 1의 구조를 살펴보면 두 개의 MUX를 사용하여 각각 16개의 입력순서를 제어한다 MUX의 입력순서는 번호로 표기하였다. 그림 1의 각각의 16 cycle 블록부터 1 cycle 블록에서 사용되는 행렬은 각각 다음 식(4)-(9)와 같다

$$\begin{bmatrix} x_{17} \\ x_{19} \\ x_{21} \\ x_{23} \\ x_{25} \\ x_{27} \\ x_{29} \\ x_{31} \\ x_{33} \\ x_{35} \\ x_{37} \\ x_{39} \\ x_{41} \end{bmatrix} = \begin{bmatrix} -.049 & .938 & .571 & -.740 & .336 & -.941 & -.427 & .903 & .146 & -.989 & -.595 & .808 & -.242 & .970 & .514 & -.857 \\ -.146 & -.989 & .808 & -.595 & .857 & .514 & -.970 & -.242 & .427 & .903 & -.941 & .336 & -.671 & -.740 & .988 & -.049 \\ -.242 & .970 & -.514 & .857 & .989 & .146 & -.808 & -.595 & .671 & -.740 & .049 & -.988 & -.941 & .336 & .427 & .903 \\ -.336 & -.941 & -.903 & .427 & .671 & -.740 & -.049 & .988 & .857 & .514 & .970 & .242 & -.989 & .146 & -.595 & -.808 \\ -.427 & .903 & .336 & -.941 & .049 & .988 & .740 & -.671 & .970 & -.242 & .514 & .857 & -.808 & -.595 & -.989 & .146 \\ -.514 & .857 & .970 & -.242 & -.595 & -.808 & .989 & .146 & .988 & -.049 & -.671 & -.740 & .427 & .903 & -.336 & .941 \\ -.595 & .808 & -.146 & .989 & -.970 & .242 & .514 & .857 & .941 & .336 & -.903 & -.427 & .049 & -.988 & .571 & -.740 \\ -.671 & -.740 & -.988 & .049 & -.903 & .427 & .336 & -.941 & .808 & -.595 & .146 & .989 & .514 & .857 & .970 & -.242 \\ -.740 & .671 & -.049 & -.988 & .427 & -.903 & -.941 & .336 & .595 & .808 & .989 & -.146 & .857 & -.514 & .242 & .970 \\ -.808 & -.595 & .989 & .146 & .242 & .970 & -.857 & .514 & .336 & -.941 & .427 & -.903 & .988 & .049 & -.740 & -.671 \\ -.857 & .514 & .242 & .970 & .808 & -.595 & -.146 & -.989 & .049 & .988 & -.740 & .671 & .903 & .427 & -.941 & -.336 \\ -.903 & -.427 & -.941 & -.336 & .988 & -.049 & .671 & .740 & -.242 & -.970 & -.857 & .514 & .595 & -.808 & -.146 & .989 \\ -.941 & .336 & .427 & -.903 & .740 & .671 & .988 & .049 & -.514 & .857 & .242 & -.970 & .146 & .989 & .808 & -.595 \\ -.970 & -.242 & .857 & .514 & .146 & -.989 & .595 & -.808 & -.740 & -.671 & .988 & .049 & -.336 & -.941 & .903 & .427 \\ -.989 & .146 & .595 & .808 & -.514 & .857 & -.242 & .970 & -.903 & .427 & .336 & .941 & -.740 & .671 & .049 & .988 \\ -.988 & -.049 & -.740 & -.671 & -.941 & -.336 & -.903 & -.427 & -.989 & -.146 & -.808 & -.595 & -.970 & -.242 & -.857 & -.514 \end{bmatrix} \begin{bmatrix} X_0-X_{31} \\ X_{15}-X_{16} \\ X_7-X_{24} \\ X_8-X_{23} \\ X_3-X_{28} \\ X_{12}-X_{19} \\ X_4-X_{27} \\ X_{11}-X_{20} \\ X_1-X_{30} \\ X_{14}-X_{17} \\ X_6-X_{25} \\ X_9-X_{22} \\ X_2-X_{29} \end{bmatrix} \quad (4)$$

$$\begin{bmatrix} x_{18} \\ x_{22} \\ x_{26} \\ x_{30} \\ x_{34} \\ x_{38} \\ x_{42} \\ x_{46} \end{bmatrix} = \begin{bmatrix} -0.098 & 0.995 & 0.634 & -0.773 & 0.290 & -0.956 & -0.471 & 0.881 \\ -0.290 & -0.956 & 0.881 & -0.471 & 0.773 & 0.634 & -0.995 & -0.098 \\ -0.471 & 0.881 & -0.290 & 0.956 & 0.995 & -0.098 & -0.634 & -0.773 \\ -0.634 & -0.773 & -0.995 & 0.098 & 0.881 & -0.471 & 0.290 & 0.956 \\ -0.773 & 0.634 & -0.098 & -0.995 & 0.471 & 0.881 & 0.956 & -0.290 \\ -0.881 & -0.471 & 0.956 & 0.290 & -0.098 & -0.995 & 0.773 & -0.634 \\ -0.956 & 0.290 & 0.471 & 0.881 & -0.634 & 0.773 & -0.098 & 0.995 \\ -0.995 & -0.098 & -0.773 & -0.634 & -0.956 & -0.290 & -0.881 & -0.471 \end{bmatrix} \begin{bmatrix} (X_0+X_{31})-(X_{15}+X_{16}) \\ (X_7+X_{24})-(X_8+X_{23}) \\ (X_3+X_{28})-(X_{12}+X_{19}) \\ (X_4+X_{27})-(X_{11}+X_{20}) \\ (X_1+X_{30})-(X_{14}+X_{17}) \\ (X_6+X_{25})-(X_9+X_{22}) \\ (X_2+X_{29})-(X_{13}+X_{18}) \\ (X_5+X_{26})-(X_{10}+X_{21}) \end{bmatrix} \quad (5)$$

$$\begin{bmatrix} x_{20} \\ x_{28} \\ x_{36} \\ x_{44} \end{bmatrix} = \begin{bmatrix} -0.195 & 0.980 & 0.555 & -0.831 \\ -0.555 & -0.831 & 0.980 & -0.195 \\ -0.831 & 0.555 & 0.195 & 0.980 \\ -0.980 & -0.195 & -0.831 & -0.555 \end{bmatrix} \begin{bmatrix} (X_0+X_{31}+X_{15}+X_{16})-(X_7+X_{24}+X_8+X_{23}) \\ (X_3+X_{28}+X_{12}+X_{19})-(X_4+X_{27}+X_{11}+X_{20}) \\ (X_1+X_{30}+X_{14}+X_{17})-(X_6+X_{25}+X_9+X_{22}) \\ (X_2+X_{29}+X_{13}+X_{18})-(X_5+X_{26}+X_{10}+X_{21}) \end{bmatrix} \quad (6)$$

$$\begin{bmatrix} x_{24} \\ x_{40} \end{bmatrix} = \begin{bmatrix} -0.382 & 0.923 \\ -0.923 & -0.382 \end{bmatrix} \begin{bmatrix} (X_0+X_{31}+X_{15}+X_{16}+X_7+X_{24}+X_8+X_{23})- \\ (X_3+X_{28}+X_{12}+X_{19}+X_4+X_{27}+X_{11}+X_{20}) \\ (X_1+X_{30}+X_{14}+X_{17}+X_6+X_{25}+X_9+X_{22})- \\ (X_2+X_{29}+X_{13}+X_{18}+X_5+X_{26}+X_{10}+X_{21}) \end{bmatrix} \quad (7)$$

$$x_{32} = [-.707] \begin{bmatrix} (X_0+X_{31}+X_{15}+X_{16}+X_7+X_{24}+X_8+X_{23}+ \\ X_3+X_{28}+X_{12}+X_{19}+X_4+X_{27}+X_{11}+X_{20})- \\ (X_1+X_{30}+X_{14}+X_{17}+X_6+X_{25}+X_9+X_{22}+ \\ X_2+X_{29}+X_{13}+X_{18}+X_5+X_{26}+X_{10}+X_{21}) \end{bmatrix} \quad (8)$$

$$x_{48} = [-1\ 0] \begin{bmatrix} (X_0+X_{31}+X_{15}+X_{16}+X_7+X_{24}+X_8+X_{23}+ \\ X_3+X_{28}+X_{12}+X_{19}+X_4+X_{27}+X_{11}+X_{20})+ \\ (X_1+X_{30}+X_{14}+X_{17}+X_6+X_{25}+X_9+X_{22}+ \\ X_2+X_{29}+X_{13}+X_{18}+X_5+X_{26}+X_{10}+X_{21}) \end{bmatrix} \quad (9)$$

Ⅲ CSD와 Common Sub-expression을 사용한 Sub-block의 구조 설계

1 CSS 16 cycle 구조

그림 1에서 제안된 16 cycle 구조는 16×16=256개의 곱셈연산이 필요한데, 16개의 곱셈기를 사용하여 16 cycle 후에 계산이 완료되도록 설계하였다 이 절

에서는 16개의 곱셈기를 덧셈과 쉬프트를 사용하여 효율적으로 구현하는 구조를 제안한다. 저전력 구조를 설계하기 위하여 두 가지 방식을 채택하였다. 즉, 덧셈의 연산을 줄이기 위하여 CSD(Canonic Signed Digit)형의 필터계수 방식과 공통패턴을 공유하는 CSS(Common Sub-expression Sharing) 방식을 채택하였다. 이와 같이 제안된 방식을 적용하기 위하여 식 (4)에서 사용되는 16개의 계수 0.049, 0.146, 0.242, 0.336, 0.427, 0 514, 0.595, 0 671, 0.740, 0 803, 0.857, 0.903, 0.941, 0.970, 0.989, 0.998를 18비트 정세도의 CSD형으로 나타내면 표 1과 같다.

표 1 1차 Odd IMDCT의 CSD형 계수

	0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10	-11	-12	-13	-14	-15	-16	-17
0049					1		N			1			1					N
0146				1		1		N		N			1					
0242			1					N			1		N		1			N
0336		1		N		N		N			1					N		
0427		1			N			N		N			N		1			
0514		1					1			N		1			N			
0595		1		1		N				1								N
0671	1		N		N		N						N		N		N	
0740	1		N					N			N		N			N		1
0803	1		N		1			N		N		1					N	
0857	1			N			N			N			1		1			
0903	1			N		1				N			N		N			N
0941	1				N				1					1			1	
0970	1					N					1		1		1			N
0989	1						N		1		1			N		1		1
0998	1										N		N				1	

위의 표에서 CSS 방식을 적용하기 위하여 먼저 공통패턴들을 2중 실선으로 표시하였다 표 1에서와 같이 101, 10N , 1001, 100N, 1000N, 10000N의 6개의 공통패턴이 있음을 알 수 있다 여기에서 N0N은 101과 같은 패턴이다 즉, 구현할 때에 101의 패턴에 -만 붙이면 N0N의 패턴이 되기 때문이다 따라서 표 1에서의 공통패턴은 다음과 같다.

$$x_2 = x_1 + x_1 \gg 2, \quad x_3 = x_1 - x_1 \gg 2,$$
$$x_4 = x_1 + x_1 \gg 3, \quad x_5 = x_1 - x_1 \gg 3$$
$$x_6 = x_1 - x_1 \gg 4 \quad x_7 = x_1 - x_1 \gg 5 \qquad (10)$$

표 1의 각각의 계수는 공통패턴을 공유하여 다음 식으로 나타낼 수 있다

$$\begin{aligned}
0.049 &= x_3 \gg 4 + x_4 \gg 9 - x_1 \gg 17,\\
0.146 &= x_2 \gg 3 - x_2 \gg 7 + x_1 \gg 12,\\
0.242 &= x_7 \gg 2 + x_3 \gg 10 + x_5 \gg 14,\\
0.336 &= x_3 \gg 1 - x_2 \gg 5 + x_7 \gg 10,\\
0.427 &= x_5 \gg 1 - x_2 \gg 7 - x_3 \gg 12,\\
0.514 &= x_1 \gg 1 + x_5 \gg 6 + x_5 \gg 11,\\
0.595 &= x_2 \gg 1 - x_6 \gg 5 - x_1 \gg 17,\\
0.671 &= x_3 - x_2 \gg 4 - x_2 \gg 12 - x_1 \gg 16,\\
0.740 &= x_3 - x_4 \gg 7 - x_4 \gg 12 + x_1 \gg 17,\\
0\ 803 &= x_1 - x_3 \gg 2 - x_2 \gg 7 + x_6 \gg 11,\\
0.857 &= x_5 - x_4 \gg 6 + x_2 \gg 12,\\
0\ 903 &= x_1 - x_3 \gg 3 - x_4 \gg 9 - x_4 \gg 14,\\
0.941 &= x_1 - x_6 \gg 4 + x_4 \gg 13,\\
0.970 &= x_7 + x_2 \gg 10 - x_1 \gg 14,\\
0\ 989 &= x_1 - x_3 \gg 6 + x_5 \gg 10 + x_2 \gg 15,\\
0.998 &= x_1 - x_2 \gg 10 + x_1 \gg 16.
\end{aligned} \qquad (11)$$

위의 식(11)에서 0.049는 공통패턴을 사용하여 2개의 덧셈으로 구현된다. 식(11)에서 보듯이 16개의 계수를 구현하기 위하여 37개의 덧셈을 사용하였다. 따라서 표 1을 구현하는 하는데 총 43개의 덧셈기가 사용되며 제안된 구조는 그림 2와 같다.

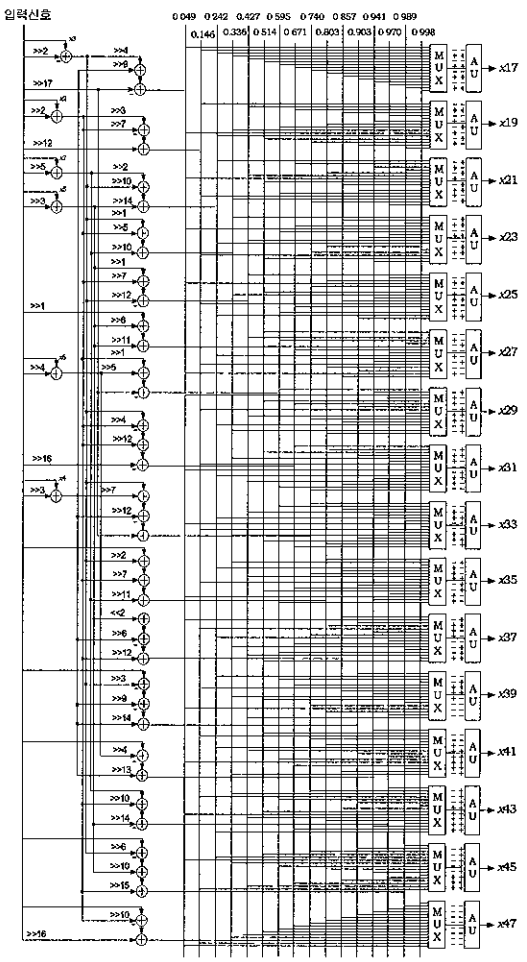


그림 2 제안된 저전력 16 cycle 구조

16 cycle 구조의 첫 번째 cycle에 들어온 입력신호 X_0-X_{31}은 위의 식(11)의 계산을 통하여 식(4) 행렬의 첫 번째 열의 element인 -0.049, -0.146, -0.242, -0.336, -0.427, -0.514, -0.595, -0.671, -0.740, -0.803, -0.857, -0.903, -0.941, -0.970, -0.989, -0.998 가 곱하여져서 16개의 출력신호용 AU에 저장된다 두 번째 cycle에 들어온 입력은 X_{15}-X_{16}이고 이때 곱해지는 계수는 식(4) 행렬의 두 번째 열이다 이 계수들은 부호만 제외하면 첫 번째 cycle에서 사용된 수들과 같다. 따라서 첫 번째 cycle에서 사용된 계수용 하드웨어를 두 번째 cycle에도 그대로 사용한다 즉, 부가적인 하드웨어가 필요 없고 단지 출력신호용 AU에서 +와 -를 선택하기만 하면 된다. 세 번째부터 16 번째 cycle에서도 마찬가지로 같은 계수들이 사용되므로 부가적인 하드웨어는 필요 없으며 출력신호용 AU의 입력단에는 입력을 선택하기 위한 MUX가 필요하다 즉 x_{17}의 계산을 위해서 각각의 cycle 마다 -0.049 0.998 0.671 -0.740 0.336 -0.941 -0.427 0.903 0.146 -0.989 -0.595 0.803 -0.242 0.970 0.514 -0.857의 계산이 필요하므로 이를 각 cycle마다 선택한다 MUX의 입력선택 순서는 위부터 각 cycle마다 한 개씩이다 그리고 AU의 동작은 AU의 좌측에 표기된 순서로 각 cycle 마다 덧셈기 또는 뺄셈기로 동작한다 뺄셈의 계산은 Sum과 Carry 대신에 Difference와 Borrow의 회로가 필요하고 구현비용은 덧셈과 같다. AU도 한 개의 덧셈기로 간주될 수 있으므로 위의 제안된 구조는 59개의 덧셈과 16 cycle이 필요하다 입력신호를 가공하는데 1개의 덧셈이 필요하므로 제안된 16 cycle 구조를 설계하기 위하여 총 60개의 덧셈연산을 사용하였다.

2. CSS 8 cycle 구조

식(5)의 행렬을 8 cycle에 계산하기 위하여 1 cycle 당 한 개의 입력신호를 사용한다. 따라서 첫 번째 cycle에 (X_0+X_{31})-$(X_{15}+X_{16})$의 입력신호와 곱해지는 모든 계산을 수행하도록 설계한다 두 번째, 세 번째 그리고 마지막 여덟 번째 cycle에서는 각각 (X_7+X_{24})-(X_8+X_{23}),(X_3+X_{28})-$(X_{12}+X_{19})$, (X_4+X_{27})-$(X_{11}+X_{20})$,(X_1+X_{30})-$(X_{14}+X_{17})$, (X_6+X_{25})-(X_9+X_{22}),(X_2+X_{29})-$(X_{13}+X_{18})$, (X_5+X_{26})-$(X_{10}+X_{21})$의 입력신호를 사용하는 계산을 수행한다. 첫 번째 cycle에서 필요한 계수는 식(5)행렬의 첫 번째 열인 -0.098, -0.290, -0.471, -0.634, -0.773, -0.881, -0.956, -0.995이다 이와 같은 8개의 계수들을 17비트 정세도의 CSD형으로 나타내면 다음의 표 2와 같다 표2에서도 역시 공통패턴들을 2중 실선으로 표현하였다. 표 6에서는 101, 10N, 100N, 1000N 4개의 공통패턴이 있으며 16 cycle 구조에서 정의한 공통패턴을 그대로 사용한다 공통패턴을 만들기 위하여 4개의 덧셈을 사용하였고 이와 같은 공통패턴을 사용하여 계수들을 나타내면 식(12)와 같다.

표 2 2차 Odd IMDCT의 CSD형 계수

	0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10	-11	-12	-13	-14	-15	-16	-17
0.098				1		N			1			1		N				N
0.290			1			1		1			1		1					
0.471		1				N			1		N		N		N		1	
0.634		1		1				1		1		N		1				N
0.773	1		N			1		N				N			1			N
0.881	1			N				1			N			1		N		N
0.956	1				N		1		1					N		1		
0.995	1								N		N				1			

$$
\begin{aligned}
0.098 &= x_3 \gg 3 + x_1 \gg 8 + x_3 \gg 11 - x_1 \gg 17,\\
0.290 &= x_1 \gg 2 + x_2 \gg 5 + x_2 \gg 10,\\
0.471 &= x_6 \gg 1 + x_3 \gg 8 - x_1 \gg 12 - x_3 \gg 14,\\
0.634 &= x_2 \gg 1 + x_2 \gg 7 - x_1 \gg 11 + x_6 \gg 13,\\
0.773 &= x_3 + x_3 \gg 5 - x_5 \gg 11 - x_1 \gg 17,\\
0.881 &= x_5 + x_5 \gg 7 + x_3 \gg 13 - x_1 \gg 17\\
0.956 &= x_6 + x_2 \gg 6 - x_3 \gg 13,\\
0.995 &= x_1 - x_2 \gg 8 + x_1 \gg 14.
\end{aligned}
\tag{12}
$$

위의 0.098, 0.290, 0.471, 0.634, 0.773, 0.881, 0.956, 0.995의 계수들을 덧셈과 쉬프트 연산을 사용하여 구현하면 다음 그림 3의 왼쪽 부분과 같다

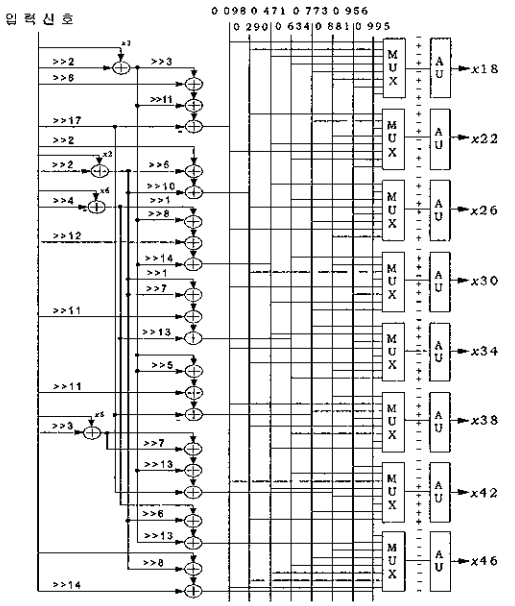


그림 3 제안된 저전력 8 cycle 구조

그림 3에서 25개의 덧셈으로 0.098, 0.290, 0.471, 0.634, 0 773, 0.881, 0.956, 0.995의 계수들을 구현하였다 첫 번째 cycle에 들어온 입력신호 (X_0+X_{31})-$(X_{15}+X_{16})$은 동시에 -0 098, -0 290, -0.471, -0.634, -0 773, -0.881, -0.956, -0.995가 곱해져서 8개의 출력신호용 AU에 저장된다. 즉, 이 출력용 AU에는 x_{18}, x_{22}, x_{26}, x_{30}, x_{34}, x_{38}, x_{42}, x_{46}의 첫 번째 cycle의 첫 번째 중간결과 값이 저장된다 두 번째 cycle에서 사용되는 입력신호는 (X_7+X_{24})-(X_8+X_{23})이고 이때 사용되는 계수는 식(5) 행렬의 두 번째 열이다. 이 계수들도 부호만 제외하면 첫 번째 cycle에서 사용된 계수들과 같다. 따라서 두 번째 cycle을 위한 부가적인 하드웨어는 필요 없고, 역시 출력신호용 AU에서 +와 -를 선택하면 된다. 기타의 하드웨어들의 동작원리도 16 cycle 구조와 동일하다. 계수 구현에 25개의 덧셈기가 필요하며, 입력신호를 만드는데 1개의 덧셈이 필요하므로 26개의 덧셈 연산이 필요하다. 그리고 출력용 AU가 8개 필요하므로 제안된 구조는 총 34개의 덧셈 연산이 필요하다

3 CSS 4 cycle 구조

식(6)의 행렬도 같은 동작원리의 하드웨어로 구현된다. 식(6) 행렬의 첫 번째 열의 계수들을 18비트 정세도의 CSD형으로 나타내면 다음 표 3과 같다

표 3 3차 Odd IMDCT의 CSD형 계수

	0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10	-11	-12	-13	-14	-15	-16	-17
0.195			1		N			1					N				1	
0.555		1			1			N			1			N		1		N
0.831	1		N		1		1		1			N			N		N	
0.980	1						N		N				1		1			1

위의 표 3에는 101, 10N, 100N의 공통패턴이 있으며, 공통패턴을 만들기 위하여 3개의 덧셈을 사용하였다 이와 같은 공통패턴을 사용하여 0 195, 0 555, 0.831, 0 980의 계수들을 나타내면 식 (13)과 같다

$$\begin{aligned} 0.195 &= x_3 \gg 2 + x_1 \gg 7 - x_1 \gg 12 + x_1 \gg 16, \\ 0\ 555 &= x_1 \gg 1 + x_5 \gg 4 + x_5 \gg 10 + x_3 \gg 15, \\ 0.831 &= x_3 + x_2 \gg 4 + x_5 \gg 8 - x_2 \gg 14, \\ 0.980 &= x_1 - x_2 \gg 6 + x_2 \gg 12 + x_1 \gg 17 \end{aligned} \quad (13)$$

식(13)은 12개의 덧셈과 공통패턴의 3개의 덧셈을 합쳐 총 15개의 덧셈으로 계수들을 구현하였다. 이와 같이 설계한 구조는 다음 그림 4와 같다. 제안된 4 cycle 구조는 그림 4에서도 알 수 있듯이 계

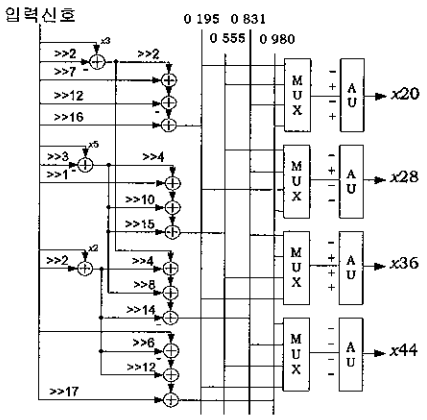


그림 4 제안된 저전력 4 cycle 구조

수 구현에 15개의 덧셈기가 필요하며, 입력신호를 만드는 1개의 덧셈기와 출력용 AU의 4개를 더해 총 20개의 덧셈연산이 필요하다

4. CSS의 2 cycle과 1 cycle 구조

식 (7)의 행렬도 같은 동작원리의 하드웨어로 구현된다. 식 (7) 행렬의 첫 번째 열의 계수들을 18비트 정세도의 CSD형으로 나타내면 다음 표 4와 같다.

표 4 4차 Odd IMDCT의 CSD형 계수

	0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10	-11	-12	-13	-14	-15	-16	-17
0.382		1		N				1						N				N
0.923	1				N		N			1					1		N	

식 (8)은 1개의 곱셈이 필요하며 식 (9)의 연산은 -1을 곱하므로 곱셈이 필요치 않다. 식 (8)의 계수를 CSD형으로 나타내면 표 5와 같다.

표 5 5차 Odd IMDCT의 CSD형 계수

	0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10	-11	-12	-13	-14	-15	-16	-17
0.707	1		N		N		1		1						1			1

공통패턴을 사용하여 표4와 표5의 0 382, 0.923, 0 707의 계수들을 나타내면 다음과 같다

$$\begin{aligned} 0\ 382 &= x_3 \gg 1 + x_1 \gg 7 - x_1 \gg 13 - x_1 \gg 17, \\ 0.923 &= x_1 - x_1 \gg 4 - x_1 \gg 6 + x_1 \gg 9 + x_3 \gg 14, \\ 0.707 &= x_1 - x_2 \gg 2 + x_2 \gg 6 + x_1 \gg 14 + x_1 \gg 17. \end{aligned} \quad (14)$$

제안된 2 cycle 구조와 1 cycle 구조는 각각 그

림 5와 6과 같다. 제안된 16, 8, 4, 2, 1 cycle 구조는 입력신호 가공용 덧셈기가 10개 필요하고 계수용 덧셈연산이 96개가 필요하고 출력 AU용 덧셈이 30개가 필요하다. 따라서 총 136개의 덧셈연산이 사용된다. 여기에서 30개의 출력 AU용 덧셈기는 뺄셈도 제공하여야 한다.

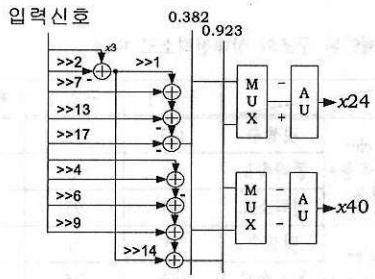


그림 5. 제안된 저전력 2 cycle 구조

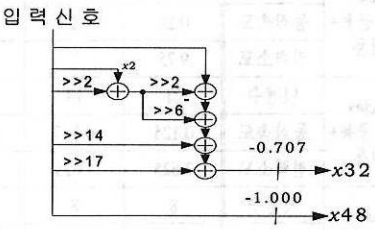


그림 6. 제안된 저전력 1 cycle 구조

Ⅳ. 성능 평가

1. HDL 코딩 실험

본 논문에서 제안된 구조의 동작 검증을 위해 그림 4, 5, 6의 구조를 C언어를 사용하여 시뮬레이션 하였다. 입력으로는 0.12, 0.11, 0.125, 0.092, 0.111, 0.099, 0.101, 0.1을 17비트 2의 보수형으로 변환하여 사용하였다. 표 6은 8개의 입력을 순서대로 입력시켜 시뮬레이션 한 출력신호의 결과를 나타내었다. 또한 FPGA 로직 구현을 위해 Altera사의 MAX-Plus II Ver10.2를 이용하여 Verilog-HDL 코딩하여 시뮬레이션 하였다. 4, 2, 1 cycle 구조에 대해서 각각의 내부 블록들이 16 clock동안 덧셈과 쉬프트 연산만을 사용하여 원하는 출력을 갖는지 확인하였다. 한 clock당 200ns의 시간이 소요되며 simulation 결과 총 3.2 μs 의 시간이 소요되었다. 그림 7은 Verilog -HDL 코딩의 결과를 나타낸 것이다. 그림 7에서 알 수 있듯이 16 clock 동안 4, 2, 1 cycle의 출력 x_{20}, x_{28}, x_{36}, x_{44}, x_{24}, x_{40}, x_{32}, x_{48}과 C언어를 사용하여 얻어낸 결과와 같음을 확인하였다.

표 6. 4, 2, 1cycle 구조의 출력

구분	출 력		
	출력	구분	2의 보수형 표현
4cycle	x_{20}	0.036219	0 0000100101000101
	x_{28}	-0.021768	1 1111101001101101
	x_{36}	0.013325	0 0000001101101001
	x_{44}	-0.026762	1 1111100100100110
2cycle	x_{24}	0.003341	0 0000000011011010
	x_{40}	-0.015437	1 1111110000001100
1cycle	x_{32}	-0.025452	1 1111100101111100
	x_{48}	-0.858	1 0010010001011010

2. 구현 면적 비교

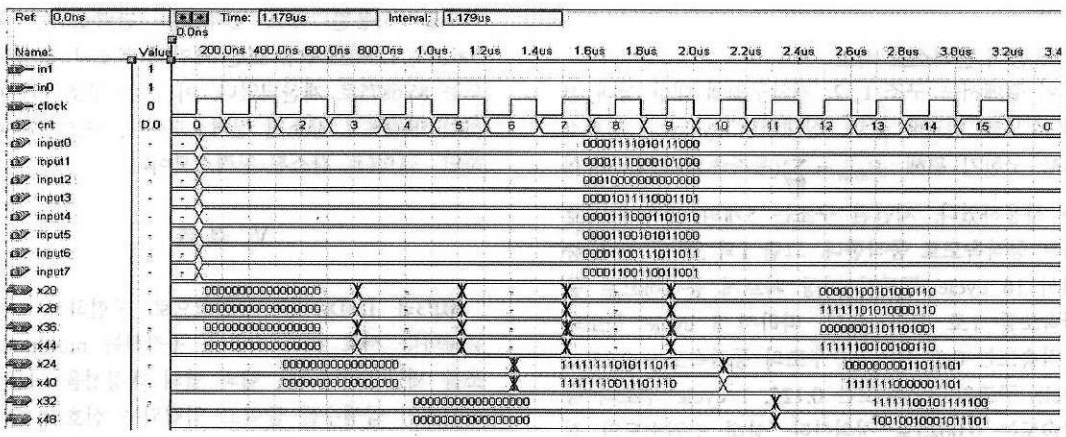


그림 7. 4, 2, 1 cycle의 simulation

Ⅱ절과 Ⅲ절에서 제안한 구조의 구현면적을 비교하여 구조의 효율성을 살펴보기로 한다 제안된 구조는 덧셈기와 쉬프터로 구성되는데, 쉬프터는 IC 구현에서 하드웨어가 필요치 않으므로 덧셈기의 사용수로 구현면적을 계산할 수 있다 따라서 제안된 구조의 덧셈기의 수로 구현면적을 계산하기로 한다 상대적인 비교를 위하여 3가지 구조의 덧셈수를 비교하기로 한다 3가지 구조 모두 Ⅱ절에서 제안된 systolic 구조를 사용한다 첫 번째 구조는 2의 보수형 계수를 사용하였으며, 두 번째 구조는 CSD형 계수를 사용하였다. 세 번째 구조는 제안된 구조로서 systolic 구조의 각 블록의 구현에 CSD형 계수와 CSS 방식을 사용하여 덧셈의 수를 비교한다 행렬 element의 비트 정세도는 17비트를 사용하였다. 이와 같은 3가지 구조의 사용된 덧셈 수를 비교하면 표 7과 같다. 구조 1, 2, 그리고 제안된 구조의 사용된 덧셈기의 수는 각각 298, 195, 136개이다 따라서 제안된 구조의 덧셈기 수는 구조 1과 비교하여 162개가 감소하였다. %로는 구조 1과 비교하여 54 36%를 감소시켰다

표 7 제안된 구조의 덧셈수 비교

구분	구조 1 (2의보수형)	구조 2 (CSD형)	제안 구조 (CSD형+CSS사용)
16 cycle 블록	139	77	43
8 cycle 블록	63	41	25
4 cycle 블록	33	22	15
2 cycle 블록	17	9	8
1 cycle 블록	6	6	5
입력신호가공용	10	10	10
출력AU용	30	30	30
총 덧셈수	298	195	136

3 상대 전력소모 비교

이 절에서는 구조 1, 2, 제안구조에 대한 상대 전력 소모를 비교해 본다. 상대적인 dynamic 전력소모를 구하기 위해 $P_{dyna}=\sum_{\text{블록}}(\text{동작속도}\times\text{면적})$의 식을 사용하였다 제안된 구조는 5개의 블록이 서로 다른 동작속도로 동작한다 그림 1의 5개의 블록 중에서 16 cycle 구조가 가장 빠르게 동작하므로 동작속도를 1로 정의하였다. 따라서 8 cycle 구조의 동작속도는 0 5, 4 cycle 구조의 동작속도는 0 25, 2 cycle 구조의 동작속도는 0 125, 1 cycle 구조의 동작속도는 0 0625로 정의한다 상대 전력소모의 식에서 각 블록의 동작속도가 정의되었으므로 면적만 구하면 상대전력소모를 구할 수 있다 면적으로서 덧셈의 수를 사용한다. 이와 같이 구한 덧셈의 수는 표 8과 같다. 표 7에서 보듯이 제안된 16 cycle 구조의 덧셈의 수는 43개이며 입력가공용으로 2개의 덧셈기, 출력용으로 16개의 덧셈기가 필요하므로 총 43+18=61개의 덧셈기가 필요하게 된다.

표 8 제안된 구조의 상대전력소모 비교

		구조 1	구조 2	제안구조
16 cycle+ 입력가공용+ 출력용	덧셈수	157	95	61
	동작속도	1	1	1
	전력소모	157	95	61
8 cycle+ 입력가공용+ 출력용	덧셈수	73	51	35
	동작속도	0 5	0 5	0 5
	전력소모	36 5	25 5	17 5
4 cycle+ 입력가공용+ 출력용	덧셈수	39	28	21
	동작속도	0 25	0 25	0 25
	전력소모	9 75	7	5 25
2 cycle+ 입력가공용+ 출력용	덧셈수	21	13	12
	동작속도	0 125	0 125	0 125
	전력소모	2 625	1 625	1 5
1 cycle+ 입력가공용+ 출력용	덧셈수	8	8	7
	동작속도	0 0625	0.0625	0 0625
	전력소모	0 5	0 5	0 4375
총 전력소모		206 375	129 625	85 6875

표 8에서 보듯이, 각 블록의 전력소모는 덧셈수와 동작속도를 곱한 값이다 그리고 각 블록의 전력소모를 모두 더하면 각 구조의 총 전력소모를 구할 수 있다. 2의 보수형 계수를 사용한 구조 1과 CSD형 계수를 사용한 구조 2의 총 전력소모는 각각 206.375, 129.625이다 또한 제안된 구조의 총 전력소모는 85 6875로 계산되었다. 이 값은 구조 1과 비교하여 58.48%로 감소된 전력소모이며, 구조 2와 비교하면 33.9%로 감소된 전력소모이다.

Ⅴ 결 론

MP3용 IMDCT를 저전력으로 구현하기 위하여 블록마다 서로 다른 속도로 동작하는 multirate 구조를 제안하였다 즉, 행과 열의 대칭성을 이용하여 전체적인 곱셈수를 줄이고 입력되는 신호의 순서를 변형시킴으로서 효과적인 저전력 systolic 구조를 제

안하였다. 이와 같이 설계된 각각의 sub block은 덧셈기와 쉬프트 연산을 사용하여 구현하였다. 전력소모, 즉 덧셈기의 수를 줄이기 위하여 CSD형 계수를 사용하였으며 계수들 사이의 공통패턴을 공유하는 CSS 방식을 채택함으로서 덧셈의 수를 더욱 감소시킬 수 있었다. 따라서 제안된 IMDCT 구조는 hard wired 방식으로 MP3를 구현하는 경우에 널리 사용될 수 있는 유용한 저전력 구조이다.

참 고 문 헌

[1] ISO/IEC 11172-3 MPEG-1 Committee Draft Part 3 : Audio, Layer I, II, III

[2] N. Ahmed, T. Natarajan, and K. R. Rao, "Discrete cosine transform," *IEEE Trans. Comput.*, vol. C-23, pp. 90-93, Jan. 1974.

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

[4] M. Vetterli, and H. J. Nussbaumer, "Simple FFT and DCT algorithm with reduced number of operations," *Signal Process.*, vol. 6, No. 4, pp. 267-278, 1984.

[5] M. T. Sun, L. Wu, and M. L. Liou, "A concurrent architecture for VLSI implementation of discrete cosine transform," *IEEE Trans. Circuits and Systems*, vol. CAS-34, pp. 992-994, Aug. 1987.

[6] M. Yoshida, H. Ohtomo, and I. Kuroda, "A new generation 16-bit general purpose programmable DSP and its video rate application," in *IEEE Workshop on VLSI Signal Processing*, pp. 93-101, 1993.

[7] T. S. Chang, C. S. Kung, and C. W. Jen, "A simple processor core design for DCT/IDCT," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 10, No. 3, Apr. 2000.

[8] K. Hwang, Computer Arithmetic: Principles, Architecture, and Design, New York: Wiley, 1979.

[9] R. I. Hartley, "Subexpression sharing in filters using canonic signed digit multipliers," IEEE Trans. Circuits and Systems-II: Analog and Digital Signal Processing, vol. 43, No. 10, pp. 677-688, Oct. 1996.

[10] 장영범, 양세정, "수직 공통패턴을 사용한 고속/저전력 CSD 선형위상 FIR 필터 구조," 한국통신학회논문지, 27권 4A호, pp. 324-329, 2002년 4월.

장 영 범 (Young-Beom Jang) 정회원

1981년 : 연세대학교 전기공학과 졸업(공학사)

1990년 : Polytechnic University 대학원 졸업(공학석사)

1994년 : Polytechnic University 대학원 졸업(공학박사)

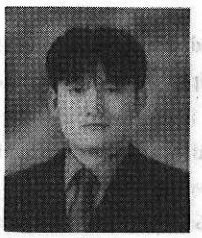
1981년~1999년 : 삼성전자 System LSI 사업부 수석연구원

2000년~2002년 : 이화여자대학교 정보통신학과 연구교수

2002년~현재 : 상명대학교 정보통신공학전공 교수

<주관심분야> 통신신호처리, 음성/오디오 신호처리, 통신신호처리용 SOC 설계

이 원 상 (Won-Sang Lee) 준회원



2004년 2월 : 상명대학교 컴퓨터 시스템 졸업(공학사)

2004년 2월 ~현재 : 상명대학교 대학원 컴퓨터 정보 통신공학 석사과정

<주관심분야> 통신신호처리, 통신신호처리용 SOC 설계