Common sub-expression sharing을 이용한 고속/저전력 DCT 구조

정회원 장 영 범*, 준회원 양 세 정**

Low-power/high-speed DCT structure using common sub-expression sharing

Young-Beom Jang* ***Regular Member,*** **Se-Jung Yang**** ***Associate Member***

요 약

이 논문에서는 곱셈기를 사용하지 않고 덧셈기 만을 사용하여 DCT를 효과적으로 수행하는 저전력 구조를 제안하였다. 고속처리가 가능하면서도 구현 하드웨어의 크기를 최소화하기 위하여 8-point DCT를 4 cycle에 수행하는 구조를 사용하였다. 즉, 첫 번째 cycle에서 사용한 계수용 하드웨어를 두 번째부터 네 번째까지의 계산에서도 공통으로 사용할 수 있는 구조를 채택하였다. 덧셈기 만을 사용하는 기존의 구조들은 CSD(Canonic signed digit)형의 계수를 사용하여 덧셈의 수를 줄이고 있다. 본 논문에서는 Common subexpression sharing 방식을 채용함으로서 하드웨어를 더욱 감소시킬 수 있는 구조를 제안하였다. 그 결과 8-point DCT의 경우에 CSD 만을 사용한 구조와 비교하여 19.5%의 덧셈 수 감소 효과를 달성하였다

Key Words : DCT, CSD, common sub-expression sharing

ABSTRACT

In this paper, a low-power 8-point DCT structure is proposed using add and shift operations. Proposed structure adopts 4 cycles for complete 8-point DCT in order to minimize size of hardware and to enable high-speed processing. In the structure, hardware for the first cycle can be shared in the next 3 cycles since all columns in the DCT coefficient matrix are common except sign. Conventional DCT structures implemented with only add and shift operation use CSD(Canonic Signed Digit) form coefficients to reduce the number of adders. To reduce the number of adders further, we propose a new structure using common sub-expression sharing techniques. With this techniques, the proposed 8-point DCT structure achieves 19.5% adder reduction comparison to the conventional structure using only CSD coefficient form.

Ⅰ. 서 론

Descrete cosine transform(DCT)[1]은 여러 가지 데이터 압축 표준들에서 매우 중요한 역할을 수행한다. DCT는 쓰임새가 많아지면서 많은 고속 알고리듬이 연구되었다[2]-[4]. 이와 같은 고속 알고리듬들은 DCT가 가지고 있는 수학적 성질을 이용하여 계산 량을 감소시킨다. 예를 들어 8-point DCT를 수행하는데 [2]에서는 16개의 곱셈과 26개의 덧셈이 필요하고 [3]에서는 12개의 곱셈과 29개의 덧셈이 필요하다. 또한 비디오 압축과 복원에 사용되는 표준들에서는 실시간 처리의 요구를 만족시키기 위한 DCT의 효과적인

* 상명대학교 정보통신공학전공(ybjang@smu.ac.kr), **EE Dept., University of Southern California
논문번호 : 030195-0509, 접수일자 : 2003년 5월 9일

VLSI 구조가 연구되었다[5]-[6]. 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[7]-[8]. 그러나, 이와 같은 프로세서를 사용한 구현은 하드웨어는 줄일 수 있지만 실시간으로 처리되기 위해서는 프로세서의 고속동작이 필수적이다. 또한 지금까지 열거한 하드웨어 구조와 프로세서를 사용한 구조의 장점 만을 취하기 위하여 하이브리드 구조가 연구되고 있다[9]. 하이브리드 구조를 사용한 [9]에서는 곱셈기를 내장한 프로세서를 사용하지 않고 AU(Arithmetic Unit), 즉 덧셈기 1개만을 내장한 프로세서를 사용하여 2-D 8 × 8 DCT를 구현하기 위하여 1208개의 덧셈을 수행하여야한다. 그러나 [9]의 방식은 한 개의 AU 프로세서를 사용하여 DCT를 수행하므로 거의 프로세서 방식에 가깝다. 본 논문에서는 8-point DCT 계산을 위하여 DCT 계수의 계산은 덧셈기를 사용하는 하드웨어가 담당하고, 완전한 출력신호 계산은 4 cycle의 AU가 담당하는 하이브리드 방식을 채용하였다. 기존의 DCT 구조들도 이와 같은 하이브리드 방식을 사용하고 있으며, CSD(Canonic Signed Digit) 형의 DCT 계수를 사용하고 있다. 본 논문에서는 CSD 형의 DCT 계수를 사용하는 기존의 하이브리드 구조보다 덧셈의 수를 더욱 감소시킬 수 있는 새로운 기법과 구조를 제안한다.

II. Transposed direct form을 사용한 기본 DCT 구조

이 절에서는 곱셈기를 사용하는 기존의 DCT 구조를 소개한다. 이 구조가 본 논문이 제안하는 DCT 구조의 기본 구조가 되기 때문이다. 1차원의 8-point DCT는 다음의 식으로 나타낸다.

$$X_k = \frac{1}{4} c_k \sum_{i=0}^{7} x_i \cos[(2i+1)\frac{\pi}{16} k] \quad (1)$$

여기에서 $c_0 = \frac{1}{\sqrt{2}}$, 그 외의 $c_k = 1$.

우리가 제안하게 될 DCT 구조를 효과적으로 표현하기 위하여 위의 8-point DCT를 행렬식으로 나타내면 편리하다. 따라서 위의 DCT를 행렬식으로 표현하면 다음과 같다.

$$\begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \\ X_4 \\ X_5 \\ X_6 \\ X_7 \end{bmatrix} = \begin{bmatrix} d & d & d & d & d & d & d & d \\ a & c & e & g & -g & -e & -c & -a \\ b & f & -f & -b & -b & -f & f & b \\ c & -g & -a & -e & e & a & g & -c \\ d & -d & -d & d & d & -d & -d & d \\ e & -a & g & c & -c & -g & a & -e \\ f & -b & b & -f & -f & b & -b & f \\ g & -e & c & -a & a & -c & e & -g \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} \quad (2)$$

위의 행렬식에서 사용되는 7개의 DCT 계수는 다음과 같다.

$$\begin{aligned} &a = 0.2451963201,\ b = 0.2309698831, \\ &c = 0.2078674031,\ d = 0.1767766953, \\ &e = 0.1388925583,\ f = 0.0956708581, \\ &g = 0.0487725805 \end{aligned} \quad (3)$$

위의 식(2)를 보면 64개의 곱셈이 필요하다. 그런데 식(2)의 계수 행렬은 어느 행을 보더라도 부호만 빼면 좌우 대칭임을 알 수 있다. 따라서 중복되는 곱셈을 다음과 같이 줄일 수 있다.

$$\begin{bmatrix} X_0 \\ X_2 \\ X_4 \\ X_6 \end{bmatrix} = \begin{bmatrix} d & d & d & d \\ b & f & -f & -b \\ d & -d & -d & d \\ f & -b & b & -f \end{bmatrix} \begin{bmatrix} x_0 + x_7 \\ x_1 + x_6 \\ x_2 + x_5 \\ x_3 + x_4 \end{bmatrix} \quad (4)$$

$$\begin{bmatrix} X_1 \\ X_5 \\ X_3 \\ X_7 \end{bmatrix} = \begin{bmatrix} a & c & e & g \\ c & -g & -a & -e \\ e & -a & g & c \\ g & -e & c & -a \end{bmatrix} \begin{bmatrix} x_0 - x_7 \\ x_1 - x_6 \\ x_2 - x_5 \\ x_3 - x_4 \end{bmatrix} \quad (5)$$

위의 식 (4)와 (5)를 보면 32개의 곱셈이 필요하므로 식(2)의 64개와 비교하여 곱셈의 수가 반으로 감소되었다. 위의 식(4)를 구현하는 구조를 Even DCT 구조라 하고 식(5)의 구조를 Odd DCT 구조라고 명명한다. 먼저 Even DCT 구조를 설계하는 방법은 다음과 같다. 식 (4)의 계산을 위하여 입력신호를 식 (4)의 맨 오른쪽 벡터와 같이 가공한다. 즉, 8개의 입력신호를 4개의 입력신호 x0+x7, x1+x6, x2+x5, x3+x4로 변환시킨다. 식(4)의 16개의 곱셈을 4 cycle에 수행하기 위한 방법은 Direct form과 Transposed direct form의 2가지가 가능하나, Transposed direct form이 주로 사용된다. Even DCT를 Direct form을 사용할 경우에는 첫 cycle에서 d(x0+x7), d(x1+x6), d(x2+x5), d(x3+x4)의 계산을 수행한다. 즉 식(4)의 4x4 DCT 계수 행렬에서 첫 번째 행의 계수 d들을 사용하여 계산한다. 이 경우에 각각의 cycle에서 사용하는 DCT 계수들이 공통이 아니므로 하드웨어의 구현비용이 증가하는 단점이 있다. 즉 첫 번째 cycle에서는 4개의 d를 곱하는 회로가 필요하며, 만일 1개의 d를 곱하는 회로만을 사용하려면 4배의 속도로 계산하여야 한다. 두 번째 cycle에서는 2개의 b와 역

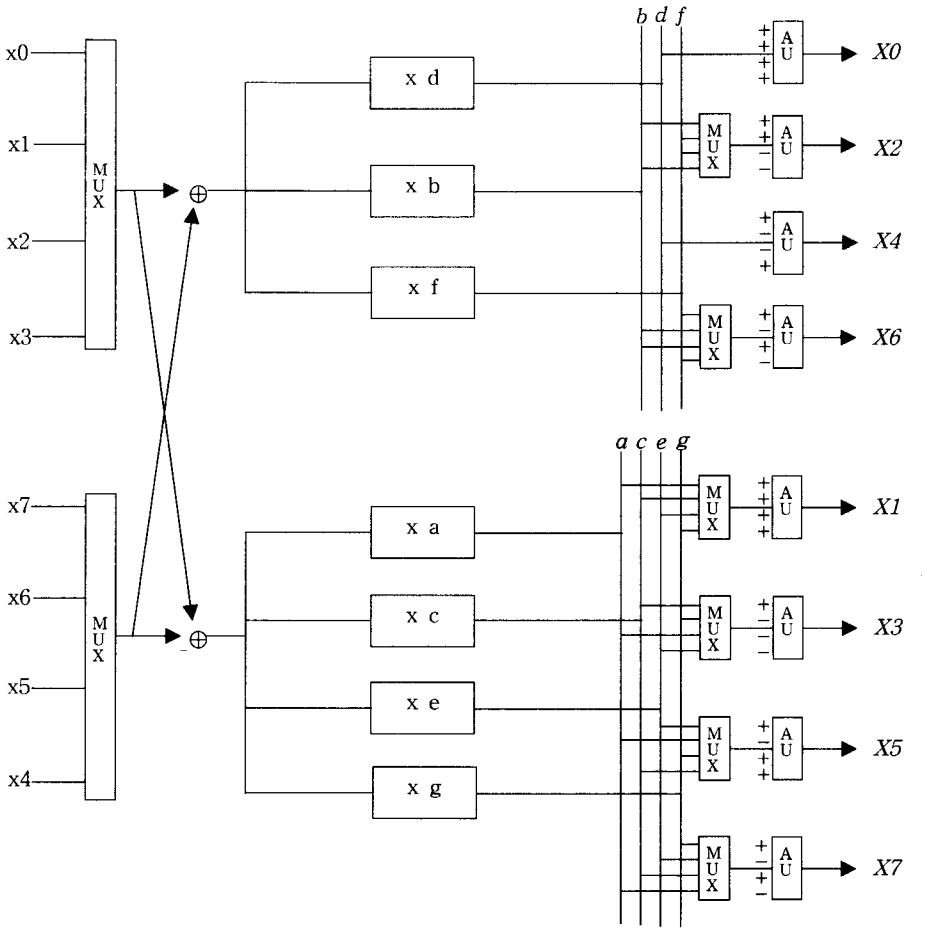


그림 1. Transposed direct form계수와 출력용AU를 사용한 저전력의 4 Cycle DCT 구조

시 2개의 f를 곱하는 회로가 필요하다. 따라서 Direct form을 사용하면 4개의 d와 2개의 f와 b를 곱하는 회로를 구성하여야 한다. 즉, 8개의 곱셈용 하드웨어가 필요하다. 그러나 Transposed direct form을 사용하면 하드웨어의 구현비용을 감소시킬 수 있다. 즉, 첫 번째 cycle에서 x0+x7에 필요한 계산을 모두 수행한다. 다시 말하면 4x4 DCT 계수 행렬의 첫 번째 열 d, b, d, f에 해당하는 곱셈을 첫 번째 cycle에 수행하도록 한다. 4x4 DCT 계수 행렬의 2, 3, 4번째 열을 살펴보면 모두 공통으로 d, b, f 만이 사용되고 있음을 알 수 있다. 부호가 -인 것은 출력용 AU에서 뺄셈의 계산을 수행하면 되므로 문제가 되지 않는다. 이와 같이 각각의 4개의 cycle에서 공통의 계수가 사용되므로 Hard-wired 구현의 경우에 Transposed direct form은 Direct form과 비교하여 구현비용의 감소라는 장점을 갖는 구조이다. Transposed direct form과 8개의 AU를 사용한 구조는 그림 1과 같다. Direct form을 사용하면 계수용 곱셈회로가 12개가 필요하나 그림 1과 같이 Transposed direct form을 사용하면 7개로 감소함을 알 수 있다. Odd DCT의 경우에는 Transposed direct form이 Direct form과 비교하여 큰 장점을 갖지 않는다고 볼 수도 있다. 이는 식 (5)의 DCT 계수 행렬에서 보듯이 열과 행이 모두 공통 계수 a, c, e, g를 사용하고 있기 때문에 Transposed direct form을 사용하나 Direct form을 사용하나 공통 계수를 모두 이용할 수 있기 때문이다. 그림 1 구조를 보면, 2개의 덧셈과 7개의 곱셈, 그리고 8개의 AU가 사용됨을 알 수 있다. 그리고 4 cycle 에 DCT가 완료된다.

III. 기존의 CSD를 사용한 DCT 구조

고속/저전력의 DCT가 요구되는 경우에, 그림 1의 DCT 계수들의 곱셈을 덧셈과 shift 만으로 수행할 수 있다. 이와 같이 곱셈을 덧셈과 shift를 사용하여 구현하는 경우, shift는 하드웨어로 구현할 때에는 비용이 거의 들지 않으므로 덧셈의 수가 곧 하드웨어 구현비용이 된다. DCT 계수들의 binary 표현에서 1의 수가 덧셈의 수가 되므로 표현되는 1의 수가 곧 구현 비용이 된다. 따라서 2의 보수형 계수보다 1의 수가 적게 사용되는 CSD형 계수를 사용하는 것이 구현비용이 적게 든다[10]-[11]. 모든 n-bit의 2의 보수형의 수는 n-bit의 CSD 형의 수로 나타낼 수 있다. 2의 보수형의 수에 비하여 CSD 형의 수는 (N+1)/2 이상의 nonzero bit를 갖지 않는 장점을 갖고 있으며, 이는 덧셈을 사용하여 구현할 때에 덧셈의 수를 줄일 수 있음을 의미한다. 식 (4)에서 사용되는 3개의 계수 d, b, f를 16비트 정세도의 CSD형으로 나타내면 표 1과 같다.

표 1. Even DCT의 CSD형 계수

	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10	-11	-12	-13	-14	-15	-16	-17
d		1		N		N		1		1						1	
b		1				N		N			1						1
f			1		N				1						N		N

위의 표에서 빈칸은 0을 나타낸다. 즉, 각 계수들의 binary 표현에서 0은 표시하지 않았다. 그리고 N은 -1을 의미한다. 표 1을 덧셈을 사용하여 구현하면 다음과 같다. 즉, d, b, f에서 사용되는 1 또는 N의 수는 각각 6개, 5개, 5개이다. 따라서 필요한 덧셈의 수는 각각 5개, 4개, 4개로서 총 13개의 덧셈이 필요하다. 위의 d, b, f의 CSD형 계수들을 덧셈을 사용하여 구현하면 다음 그림 2의 왼쪽 부분과 같다. 그림 2에서 볼 수 있듯이 13개의 덧셈으로 d, b, f의 계수들을 구현하였다. 입력신호 x0+x7은 위의 하드웨어를 통하여 동시에 d, b, d, f가 곱하여져서 4개의 출력신호용 AU에 저장된다. 즉, 이 출력용 AU에는 X0, X2, X4, X6의 중간결과 값이 저장된다. 두 번째 cycle의 입력은 x1+x6이고 이때 사용되는 계수는 식 (4) 행렬의 두 번째 열인 d, f, -d, -b이다. 따라서 두 번째 cycle에서는 첫 번째 cycle에서 사용한 계수용 하드웨어를 그대로 사용한다. 즉, 부가적인 하드웨어가 필요 없고 단지 출력신호용 AU에서 +와 -를 선택하기만 하면 된다. 세 번째와 네 번째 cycle에서도 같은 계수들이 사용된다. 다만 X2와 X6 의 출력신호용 AU의 입력 단에는 입력을 선택하기 위한 MUX

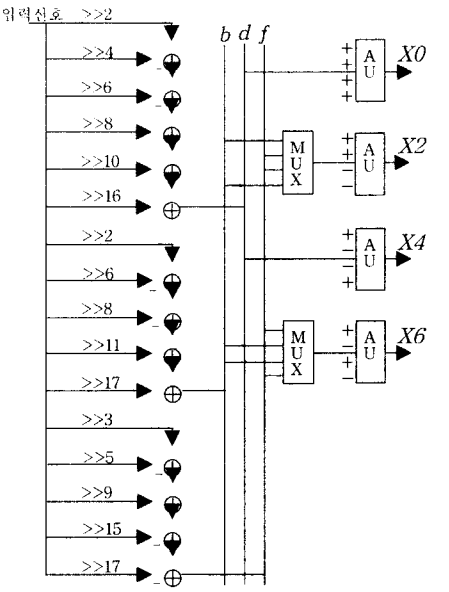


그림 2. CSD를 사용한 기존의 Even DCT 구조

가 필요하다. 즉 X2의 계산을 위해서 각각의 cycle 마다 b, f, -f, -b 의 계산이 필요하므로 이를 각 cycle 마다 선택한다. 마찬가지로 X6의 계산을 위해서 각각의 cycle 마다 f, -b, b, -f 의 계산이 필요하므로 이를 각 cycle마다 선택한다. 이와 같은 방법으로 설계된 기존의 CSD를 사용한 Even DCT의 구조는 그림 2와 같다. 그림 2에서 -로 표시된 것은 뺄셈의 계산이다. AU도 한 개의 덧셈 혹은 뺄셈으로 간주될 수 있으므로 이 구조는 17개의 덧셈과 4 cycle이 필요하다. 입력신호를 만드는데 1개의 덧셈기가 필요하므로 총 Even DCT 구조를 설계하기 위하여 18개의 덧셈이 필요하다. CSD형 계수를 사용하여 Odd DCT 구조를 설계하는 방식은 다음과 같다. 먼저 입력신호를 식(5)와 같이 가공한다. 그리고 식(5)의 16개의 곱셈을 4 cycle에 계산하기 위하여 Even DCT 구조와 마찬가지로 1 cycle 당 한 개의 입력신호를 사용한다. 따라서 첫 번째 cycle에 x0-x7의 입력신호와 곱해지는 모든 계산을 수행한다. 두 번째, 세 번째 그리고 마지막 네 번째 cycle에서는 각각 x1-x6, x2-x5, x3-x4의 입력신호를 사용하는 계산을 수행한다. 첫 번째 cycle에서 필요한 계수는 식(5) 계수행렬의 첫 번째 열인 a, c, e, g이다. 이와 같은 4개의 계수를 16비트 정세도의 CSD형으로 나타내면 다음의 표 2와 같다.

표 2. Odd DCT의 CSD형 계수

	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10	-11	-12	-13	-14	-15	-16	-17
a		1						N		N				1		1	
c		1		N		1		1		1			N		N		1
e			1			1			N			1			N		
g				1		N			1					N			

위의 표에서 나타낸 CSD형의 a, c, e, g의 계수들을 덧셈을 사용하여 구현하면 그림 3의 왼쪽 부분과 같다. 그림 3에서 보듯이 18개의 덧셈으로 a, c, e, g의 계수들이 구현된다. 첫 번째 cycle에서 입력신호 x0-x7은 위의 하드웨어를 통하여 a, c, e, g 가 곱하여져서 4개의 출력신호용 AU에 저장된다. 즉, 이 출력용 AU에는 X1, X3, X5, X7의 첫 번째 cycle의 중간결과 값이 저장된다. 두 번째 cycle에서 사용되는 입력신호는 x1-x6이고 이때 사용되는 계수는 식(5) 행렬의 두 번째 열인 c, -g, -a, -e이다. 이 계수들도 부호만 제외하면 첫 번째 cycle에서 사용된 수들과 같다. 따라서 두 번째 cycle을 위한 부가적인 하드웨어는 필요 없고, 역시 출력신호용 AU에서 +와 -를 선택하는 구조가 된다. 세 번째와 네 번째 cycle에서도 같은 방법으로 처리된다. 다만 X1, X3, X5, X7의 출력신호용 AU의 입력 단에는 입력을 선택하기 위한 MUX가 필요하다. 이와 같이 설계한 CSD형의 계수를 사용하는 기존의 Odd DCT의 구조는 그림 3과 같다. 그림 3에서도 MUX의 입력선택 순서는 위부터 각 cycle마다 한 개씩이다. 그리고 AU의 동작은 AU의 좌측에 표기된 순서로 각 cycle마다 동작한다. 위의 구조는 22개의 덧셈기와 4 cycle이 필요하다. 입력신호를 만드는데 1개의 덧셈이 필요하므로 총 Odd DCT 구조를 설계하기 위하여 23개의 덧셈이 필요하다. 지금까지 살펴 본 CSD형 계수를 사용한 기존의 DCT 구조는 총 41개의 덧셈을 사용하여 구현됨을 알 수 있다.

Ⅳ. 제안된 Common sub-expression sharing을 사용한 DCT 구조

기존의 CSD형의 계수를 사용하는 DCT 구조는 단지 DCT 계수들을 CSD형을 사용함으로서 덧셈의 수를 감소시키는 방식이다. 디지털 필터 구조에서 Common sub-expression sharing(CSS)이라는 방식을 사용하여 덧셈의 수를 더욱 감소시키는 구조가 사용된다[12]-[13]. 이 절에서 우리는 이 CSS 방식을 DCT 계

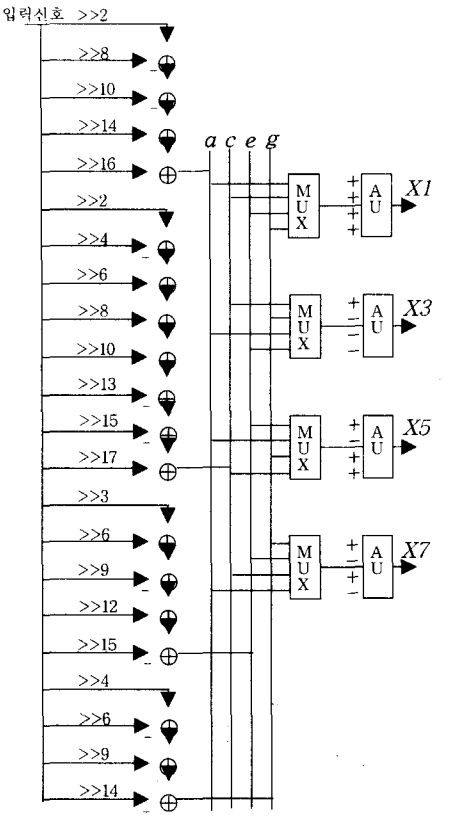


그림 3. CSD를 사용한 기존의 Odd DCT 구조

수를 곱하는 데에 사용함으로서 덧셈의 수를 더욱 감소시키는 구조를 제안한다. 먼저 Even DCT 구조를 제안하기 위하여, 식 (4)에서 사용되는 3개의 계수 d, b, f를 16비트 정세도의 CSD형 계수와 common sub-expression으로 나타내면 표 3과 같다.

표 3. Even DCT의 CSD형 계수와 common sub-expression

	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10	-11	-12	-13	-14	-15	-16	-17
d		1		N		N		1		1						1	
b		1				N		N			1						1
f			1		N				1						N		N

표 3에서 보듯이 CSS 방식을 적용하기 위하여 먼저 공통패턴들을 2중 실선으로 표시하였다. 표 3에서와 같이 101과 10N의 공통패턴이 있음을 알 수 있다. 여기에서 N0N은 101과 같은 패턴이다. 즉, 구현할 때에 101의 패턴에 -만 붙이면 N0N의 패턴이 되기 때문이다. 따라서 공통패턴은 다음과 같이 나타낼 수 있다.

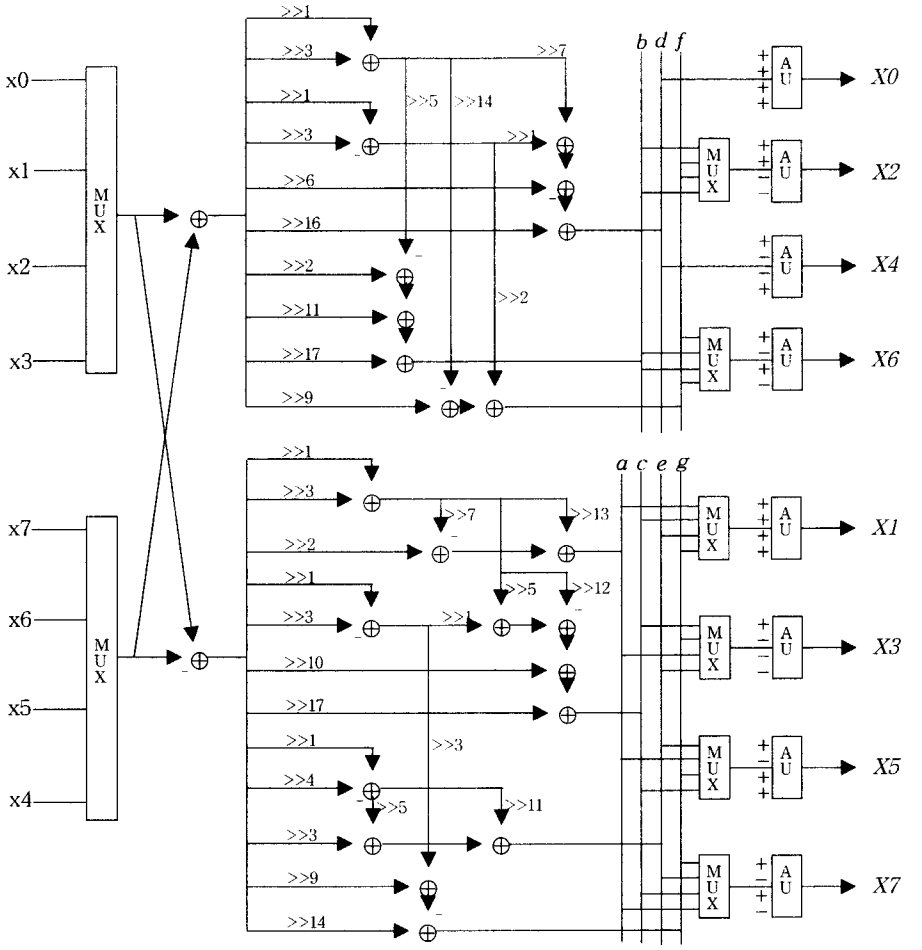


그림 4. CSD형의 계수와 CSS를 사용한 제안된 4 Cycle DCT 구조

$$x_2 = x_1 + x_1 \gg 2, \quad x_3 = x_1 - x_1 \gg 2 \tag{6}$$

위의 공통패턴을 사용하여 d, b, f의 계수들을 공통패턴을 사용하여 나타내면 다음과 같다.

$$\begin{aligned} d &= x_3 \gg 1 - x_1 \gg 5 + x_2 \gg 7 + x_1 \gg 15, \\ b &= x_1 \gg 1 - x_2 \gg 5 + x_1 \gg 10 + x_1 \gg 16, \\ f &= x_3 \gg 2 + x_1 \gg 8 - x_2 \gg 14. \end{aligned} \tag{7}$$

위의 d, b, f의 계수들을 덧셈을 사용하여 구현하면 다음 그림 4의 윗쪽 부분과 같다. 그림 4에서 볼 수 있듯이 공통패턴을 공유하였으므로 10개의 덧셈으로 d, b, f의 계수들을 구현하였다. 즉, 공통패턴 공유방식으로 13개에서 10개로 3개의 덧셈을 줄일 수 있다. 입력신호 x0+x7은 위의 하드웨어를 통하여 d, b, d, f가 곱하여져서 4개의 출력신호용 AU에 저장된다. 즉, 이 출력용 AU에는 X0, X2, X4, X6의 중간결과 값이 저장된다. 이제 두 번째 cycle을 설계해 보기로 한다. 두 번째 cycle의 입력은 x1+x6이고 이 때 사용되는 계수는 식(4) 행렬의 두 번째 열인 d, f, -d, -b이다. 이 계수들은 부호만 제외하면 첫 번째 cycle에서 사용된 수들과 같음을 알 수 있다. 따라서 첫 번째 cycle에서 사용된 계수용 하드웨어가 두 번째 cycle에도 공유할 수 있는 구조이다. 즉, 부가적인 하드웨어가 필요 없고 단지 출력신호용 AU에서 +와 -를 선택하기만 하면 된다. 세 번째와 네 번째 cycle에서도 마찬가지로 같은 계수들이 사용되므로 부가적인 하드웨어는 필요 없다. 다만 X2와 X6 의 출력신호용 AU의 입력 단에는 입력을 선택하기 위한 MUX가 필요하다. 즉 X2의 계산을 위해서 각각의 cycle 마다 b, f, -f, -b 의 계산이 필요하므로 이를 각

cycle마다 선택하여야 한다. 마찬가지로 X6의 계산을 위해서 각각의 cycle 마다 f, -b, b, -f 의 계산이 필요하므로 이를 각 cycle마다 선택한다. 이와 같이 설계한 Even DCT의 구조는 그림 4와 같다. 위의 그림 4에서 MUX의 입력선택 순서는 위부터 각 cycle마다 한 개씩이다. 그리고 AU의 동작은 AU의 좌측에 표기된 순서로 각 cycle마다 동작한다. 그림 1에서 -로 표시된 것은 뺄셈의 계산이다. 뺄셈의 계산은 Sum과 Carry 대신에 Difference와 Borrow의 회로가 필요하고 구현비용은 덧셈과 같다. 그리고 AU도 한 개의 덧셈 혹은 뺄셈으로 간주될 수 있다. 따라서 위의 제안된 구조는 14개의 덧셈과 4 cycle이 필요하다. 입력신호를 만드는데 1개의 덧셈기가 필요하므로 총 Even DCT 구조를 설계하기 위하여 15개의 덧셈이 필요하다.

이제 식(5)를 사용하여 Odd DCT 구조를 설계하는 방법을 제안한다. 먼저 입력신호를 식(5)와 같이 가공한다. 그리고 식(5)의 16개의 곱셈을 4 cycle에 계산하기 위하여 Even DCT 구조와 마찬가지로 1 cycle 당 한 개의 입력신호를 사용한다. 따라서 첫 번째 cycle에 x0-x7의 입력신호와 곱해지는 모든 계산을 수행하여야 한다. 두 번째, 세 번째 그리고 마지막 네 번째 cycle에서는 각각 x1-x6, x2-x5, x3-x4의 입력신호를 사용하는 계산을 수행하여야 한다. 첫 번째 cycle에서 필요한 계수는 식(5) 계수행렬의 첫 번째 열인 a, c, e, g이다. 이와 같은 4개의 계수들을 16비트 정세도의 CSD형으로 나타내면 다음의 표 4와 같다.

표 4. Odd DCT의 CSD형 계수와 common sub-expression

	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10	-11	-12	-13	-14	-15	-16	-17
a		1						N		N				1		1	
c		1		N		1		1		1			N		N		1
e			1			1			N			1			N		
g				1		N			1					N			

표 4에서도 역시 공통패턴들을 2중 실선으로 표현하였다. 표 2에서는 10N, 101, 100N의 공통패턴이 있음을 알 수 있다. 따라서 식 (6) 외의 공통패턴 100N은 다음과 같이 나타낼 수 있다.

$$x_4 = x_1 - x_1 \gg 3 \tag{8}$$

즉, 공통패턴을 만들기 위하여 3개의 덧셈을 사용하였다. 이와 같은 공통패턴을 사용하여 a, c, e, g의 계수들을 나타내면 다음과 같다.

$$\begin{aligned} a &= x_1 \gg 1 - x_2 \gg 7 + x_2 \gg 13 \\ c &= x_3 \gg 1 + x_2 \gg 5 + x_1 \gg 9 - x_2 \gg 12 + x_1 \gg 16 \\ e &= x_1 \gg 2 + x_4 \gg 5 + x_4 \gg 11 \\ g &= x_3 \gg 3 + x_1 \gg 8 - x_1 \gg 13. \end{aligned} \tag{9}$$

위의 a, c, e, g의 계수들을 덧셈을 사용하여 구현하면 그림 5의 아랫쪽 부분과 같다. 그림 4에서 보듯이 13개의 덧셈으로 a, c, e, g의 계수들을 구현하였다. 첫 번째 cycle에서 입력신호 x0-x7은 위의 하드웨어를 통하여 a, c, e, g 가 곱하여져서 4개의 출력신호용 AU에 저장된다. 즉, 이 출력용 AU에는 X1, X3, X5, X7의 첫 번째 cycle의 중간결과 값이 저장된다. 두 번째 cycle에서 사용되는 입력신호는 x1-x6이고 이때 사용되는 계수는 식(5) 행렬의 두 번째 열인 c, -g, -a, -e이다. 이 계수들도 부호만 제외하면 첫 번째 cycle에서 사용된 수들과 같다. 따라서 두 번째 cycle을 위한 부가적인 하드웨어는 필요 없고, 역시 출력신호용 AU에서 +와 -를 선택하는 구조를 제안한다. 세 번째와 네 번째 cycle에서도 마찬가지이다. 다만 X1, X3, X5, X7의 출력신호용 AU의 입력 단에는 입력을 선택하기 위한 MUX가 필요하다. 이와 같이 설계한 Odd DCT의 구조는 그림 5와 같다. 그림 5에서도 MUX의 입력선택 순서는 위부터 각 cycle마다 한 개씩이다. 그리고 AU의 동작은 AU의 좌측에 표기된 순서로 각 cycle마다 동작한다. 위의 구조는 17개의 덧셈기와 4 cycle이 필요하다. 입력신호를 만드는데 1개의 덧셈이 필요하므로 총 Odd DCT 구조를 설계하기 위하여 18개의 덧셈이 필요하다. 지금까지 제안한 Even DCT와 Odd DCT 구조를 사용하여 8-point DCT의 전체 구조를 설계하면 그림 6과 같다. 그림 6에서 보듯이 입력신호 가공용 덧셈이 2개가 필요하고 a, b, c, d, e, f, g의 계수용 덧셈이 23개가 필요하고 출력 AU용 덧셈이 8개가 필요하다. 따라서 총 33개의 덧셈(혹은 뺄셈)이 매 cycle마다 필요하다. 여기에서 8개의 출력 AU용 덧셈기 중에서 6개는 뺄셈도 제공하여야 한다. 그 외의 덧셈기는 는 덧셈 또는 뺄셈중 한가지만을 수행하면 된다. 그리고 4개의 입력 중에서 1개를 선택하는 MUX가 8개 필요하다. 지금까지 이 절에서 제안한 구조를 통하여 볼 수 있듯이, CSD 형의 계수와 CSS 방식을 채용하여 덧셈의 수를 매우 감소시킬 수 있다. 즉, 33개의 덧셈기를 사용하여 4 cycle에 8-point DCT를 수행할 수 있는 경제적인 저전력 구조이다. 4 cycle

에 필요한 총 덧셈의 수는 33개 × 4 cycle=132개이다.

V. 성 능 평 가

기존의 구조들과의 계산량 비교를 위하여 8-point DCT를 사용하였다. 기본 구조로서 Transposed direct form을 사용하는 4 cycle 구조를 채택하였으며 곱셈기를 사용하지 않고 add & shift로 DCT를 수행하는 구조이다. 기존 구조 1은 필터계수를 2의 보수형을 사용하였으며, 기존 구조 2는 필터계수를 CSD형을 사용하였다. 그리고 본 논문이 제안하는 구조는 CSD형의 계수와 CSS를 사용하였다. 먼저 2의 보수형 계수를 사용한 기존 구조 1의 덧셈의 수를 알아보기로 하자. DCT 계수들을 2의 보수형으로 나타내면 다음의 표 5와 같다. 이 표에서 보듯이 필터계수의 비트 정세도는 17비트를 사용하였다. 이 정세도가 높아질수록 CSS의 효과가 커지므로 본 논문이 제안하는 구조는 더욱 계산량의 감소율이 높아진다. 표 5에서 각 계수를 구현하기 위한 덧셈의 수를 맨 오른쪽 열에 나타냈다. 즉 계수 d는 6개의 1로 표현되므로 이를 덧셈을 사용하여 구현하면 5개가 필요하다. 이와 같이 하여 표 3에서 보듯이 47개의 덧셈이 필요하다. 입력신호 가공용으로 2개와 출력 AU용으로 8개의 덧셈이 필요하므로 총 57개의 덧셈이 필요하다.

이번에는 CSD형 계수를 사용하는 기존 구조 2의 덧셈의 수를 구해보기로 하자. DCT 계수들을 CSD형으로 나타내면 표 6과 같다. 표 6에서도 맨 오른쪽 열에 필요한 덧셈의 수를 표기하였다. 표 6에서 보듯이 표 5와 비교하여 1의 수가 현저히 적음을 알 수 있다. 표 6에서도 맨 우측 열에 각각의 계수를 구현하는데 필요한 덧셈의 수를 표기하였다. 표 6의 맨

표 5. 8-point DCT의 2의 보수형 계수

	-3	-4	-5	-6	-7	-8	-9	-10	-11	-12	-13	-14	-15	-16	-17	덧셈수
d	1		1	1		1		1						1		5
b	1	1	1		1	1			1						1	6
f		1	1					1	1	1	1	1		1	1	8
a	1	1	1	1	1		1	1				1		1		8
c	1	1		1		1			1	1		1	1		1	8
e	1				1	1	1			1	1	1	1			6
g			1	1				1	1	1	1	1				6

우측 열을 모두 더하면 계수용 덧셈의 수는 31개가 필요하다. 그 외에는 역시 마찬가지로 10개의 덧셈이 필요하므로 총 41개의 덧셈이 사용된다. 본 논문이 제안한 구조는 IV절에서 구한 것과 같이 33개의 덧

표 6. 8-point DCT의 CSD형 계수

	-2	-3	-4	-5	-6	-7	-8	-9	-10	-11	-12	-13	-14	-15	-16	-17	덧셈수
d	1		N		N		1		1						1		5
b	1				N		N			1						1	4
f		1		N				1						N		N	4
a	1						N		N				1		1		4
c	1		N		1		1		1			N		N		1	7
e		1			1			N			1			N			4
g			1		N			1					N				3

표 7. 8-point DCT의 덧셈의 수 비교

구분	기존구조 1 (2의보수형 계수)	기존구조 2 (CSD형 계수)	제안된 구조 (CSD형+ CSS사용)
계수용	47	31	23
신호가공용	2	2	2
출력AU용	8	8	8
계	57	41	33
%	100	71.9	57.9

셈이 필요하다. 따라서 기존 구조 1, 2와 제안된 구조의 덧셈의 수를 종합하면 표 7과 같다. 표 7에서 보듯이 제안된 구조는 2의 보수형 구조와 비교하여 덧셈의 수를 57개에서 33개로 감소시켰다. 즉, 42.1%의 감소를 달성하였다. CSD형의 계수를 사용한 것과 비교하면 41개에서 33개로 8개를 감소시켰다. 이와 같은 감소효과는 DCT의 크기가 커질수록, 또한 계수의 정세도가 높아질수록 더욱 효과가 커진다. 즉 8-point에서 16-point로 DCT 크기를 늘리거나, 17비트의 계수 정세도를 20비트로 늘리면 공통패턴이 더 많이 생기므로 덧셈의 수 감소 효과가 더욱 커진다.

VI. 결 론

8-point DCT 변환에는 4cycle의 Transposed direct form 구조가 널리 사용되고 있다. 기존의 구조들은 add & shift와 AU를 사용하여 8-point DCT를 수행하는 구조를 많이 사용하고 있다. 이와 같은 기존의 구조들은 2의 보수형이나 CSD형의 DCT 계수들을 사용하여 덧셈을 수행하고 있다. 본 논문에서는 CSD형의 계수와 CSS를 사용하여 덧셈의 수를 최소화하는 구조를 제안하였다. 8-point DCT의 경우에 2의 보수형을 사용한 기존 구조와 비교하여 42.1%의 덧셈 감소 효과를 달성하였으며, CSD형을 사용한 기존 구조와 비교하여도 19.5%의 감소 효과를 달성하였다. 본

논문이 제안한 구조의 덧셈 감소효과는 DCT의 크기가 커질수록, 또한 계수의 정세도가 높아질수록 더욱 효과가 커짐을 볼 수 있었다. 즉 8-point에서 16-point로 DCT 크기를 늘리거나, 17비트의 계수 정세도를 20비트로 늘리면 공통패턴이 더 많이 생기므로 덧셈의 수 감소 효과가 더욱 커지게 된다. 따라서 제안된 구조는 1-D DCT가 core로 사용되는 JPEG이나 MPEG등의 응용에서 널리 사용될 수 있다.

참 고 문 헌

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

[2] 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.

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

[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. Kovac and N. Ranganathan, "JAGUAR: A VLSI architecture for JPEG image compression standard," *Proc. IEEE,* vol. 83, pp. 247-258, Feb. 1995.

[7] 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.

[8] J. Golston, "Single-chip H.324 videoconferencing," *IEEE Micro.*, vol. 16, pp. 21-33, Aug. 1996.

[9] 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.

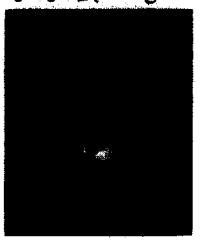
[10] R. W. Reitwiesner, "Binary arithmetic," in *Advances in Computers,* New York: Academic, vol. 1, pp. 231-308, 1966.

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

[12] 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.

[13] M. Yagyu, A. Nishihara, and N. Fujii, "Fast FIR digital filter structures using minimal number of adders and its application to filter design," *IEICE Trans. Fundamentals of Electronics Communications & Computer Sciences,* vol. E79-A No. 8, pp. 1120-1129, Aug. 1996.

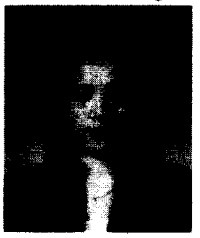
장 영 범(Young-Beom Jang) 정회원



1981년 2월 : 연세대학교 전기공학과 졸업, 공학사
1990년 1월 : Polytechnic University 전기공학과 졸업, 공학석사
1994년 1월 : Polytechnic University 전기공학과 졸업, 공학박사
1981년~1999년 : 삼성전자 System LSI 사업부 수석연구원
2000년~2002년 : 이화여자대학교 정보통신학과 연구교수
2002년~현재 : 상명대학교 컴퓨터정보통신공학부 교수

<주관심분야> 통신신호처리, 음성/오디오 신호처리

양 세 정(Se-Jung Yang) 준회원



2001년 2월 : 이화여자대학교 정보통신학과 졸업
2003년 2월 : 이화여자대학교 정보통신학과 석사
2003년 9월~현재 : EE Dept., University of Southern California 대학원

<주관심분야> 통신신호처리