새로운 연산 공유 승산기를 이용한 1차원 DCT 프로세서의 설계

이 태 욱[†] · 조 상 복[††]

요 약

DCT 알고리즘은 내적을 효율적으로 처리할 수 있는 하드웨어 구조가 필수적이다. 내적 연산을 위한 기존의 방법들은 하드웨어 복잡도가 높기 때문에, 이를 줄이기 위한 방법으로 연산 공유 승산기가 제안되었다. 하지만, 기존의 연산 공유 승산기는 전처리기 및 선택기의 비효율적 구조로 인한 성능저하의 문제점을 가지고 있다. 본 논문에서는 새로운 연산 공유 승산기를 제안하고 이를 1차원 DCT 프로세서에 적용하여 구현하였다. 연산 공유 승산기의 구조 및 논리 합성 비교 시 새로운 승산기는 기존에 비해 효율적인 하드웨어 구성이 가능함을 확인하였고, 1차원 DCT 프로세서 설계 시 기존 구현 방식들에 비해 우수한 성능을 나타내었다.

Design of 1-D DCT processor using a new efficient computation sharing multiplier

Tae-Wook Lee[†] · Sang-Bock Cho[††]

ABSTRACT

The DCT algorithm needs efficient hardware architecture to compute inner product. The conventional methods have large hardware complexity. Because of this reason, a computation sharing multiplier was proposed for implementing inner product. However, the existing multiplier has inefficient hardware architecture in precomputer and select units. Therefore it degrades the performance of the multiplier. In this paper, we proposed a new efficient computation sharing multiplier and applied it to implementation of 1-D DCT processor. The comparison results show that the new multiplier is more efficient than an old one when hardware architectures and logic synthesis results were compared. The designed 1-D DCT processor by using the proposed multiplier is more high performance than typical design methods.

키워드 : 승산기(Multiplier), 연산 공유 승산기(Computation Sharing Multiplier), DCT 프로세서(DCT Processor)

1. 서 론

최근 휴대용 멀티미디어 시스템에 대한 수요가 증가하면서 고속의 영상 데이터 처리에 대한 필요성이 점차 증가하고 있다. 영상처리에 기본적으로 사용되는 DSP(Digital Signal Processing) 알고리즘은 계산의 복잡성으로 인해 효율적인 하드웨어 구조가 필수적이다. 특히, 영상 처리 기본 블록의 하나인 DCT(Discrete Cosine Transform)의 경우 내적(inner product) 연산이 많기 때문에 이를 어떠한 구조로 설계하느냐에 따라서 프로세서의 성능 및 하드웨어 크기가 좌우된다[1, 2]. ROM 기반 분산 산술(ROM based DA, Distributed Arithmetic) 연산은 이러한 내적 연산에 있어 승산기를 사용하는 것보다 소비전력 및 하드웨어 크기를 효과적으로 줄일 수 있고, 고속 동작이 가능하며 회로 구현이 쉬운 장점이 있기 때문에 대부분의 DCT 프로세서(DCT processor)설계에 많이 이용되고 있다[3-5]. 하지만 ROM 기반 분산 산술 연산은 입력의 개수 및 연산의 정확도가 증가할수록 더 큰 크기의 ROM을 필요로 하므로 하드웨어 복잡도 및 동작 속도가 증가하는 단점을 가지고 있다[1]. 최근 들어 내적 연산에 ROM 기반 분산 산술 연산 대신 연산 공유 승산기를 사용한 연구들이 발표되고 있다[1, 2, 6, 7]. 연산 공유 승산 알고리즘은 벡터 스케일링 연산(vector scaling operation)시 하드웨어의 크기를 줄이고자 제안된 알고리즘으로 내적 연산을 상수 벡터와 스케일러(scalar)의 승산으로 분해하여 수행한다. 연산 공유 승산기를 사용한 DCT 프로세서는 기존에 널리 사용되던 ROM 기반 분산 산술 연산보다 소비 전력 및 하드웨어 복잡도를 효과적으로 줄일 수 있고, 회로 구현이 간단한 장점이 있다[1]. 하지만 기

※ 본 연구는 한국과학재단 지정 울산대학교 네트워크기반 자동화연구센터의 지원(NARC)과 산업자원부 반도체 설계 교육센터(IDEC)의 지원으로 수행되었음
† 준 회 원 : 울산대학교 대학원 전자공학과
†† 정 회 원 : 울산대학교 전기전자정보스템공학부 교수
논문접수 : 2003년 1월 27일, 심사완료 : 2003년 9월 5일

존의 연산 공유 승산기의 상수 벡터는 4비트씩 처리되므로 전처리기에 사용되는 가/감산기의 개수가 필요 이상 많고 특정 알파벳 승산에서 지연시간이 커지는 단점을 가진다. 또한 선택기에 Shifter 및 Ishifter의 부가 회로를 필요로 하여 하드웨어 용량이 증가한다. 따라서 본 논문에서는 기존 연산 공유 승산기의 장점을 살리면서 하드웨어 복잡도 및 동작 속도 측면에서 우수한 성능을 가지는 새로운 구조의 연산 공유 승산기를 제안한다. 새로운 연산 공유 승산기는 상수 벡터를 기존의 4비트 대신 2비트씩 나누어 처리함으로써 전처리기에 사용되는 가/감산기 개수 및 지연 시간을 줄이고 선택기에 사용된 Shifter와 Ishifter를 제거하여 승산기 구조를 간단화 하였다.

본 논문의 구성은 다음과 같다. 2장에서는 기존의 연산 공유 승산 알고리즘에 대해 살펴보고 효율적인 하드웨어 구성을 위한 새로운 연산 공유 승산 알고리즘을 제시한다. 3장에서는 연산 공유 승산기의 구조에 대해 살펴보고 기존의 승산기와 비교한다. 그리고 4장에서는 연산 공유 승산기를 이용한 DCT 프로세서의 구조에 대해 살펴보고, 5장에서 효용성 검증을 위한 성능비교 및 시뮬레이션을 수행한다. 마지막으로 6장에서는 연구 결과에 대한 결론을 맺는다.

2. 연산 공유 승산 알고리즘

2.1 기존의 연산 공유 승산 알고리즘

연산 공유 승산 알고리즘은 영상처리 및 DSP에서 많이 사용되는 내적 연산을 효율적인 하드웨어 구조로 설계 하기위해 제안된 알고리즘이다[1, 2, 6, 7]. 벡터 스케일링 연산은 상수 벡터($c=[c_0, c_1, c_{M-1}]$)와 스케일러(x)의 승산을 의미하는 것으로 식 (1)과 같이 표현된다.

$$y = x \cdot c = x \cdot \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_{M-1} \end{bmatrix} = \begin{bmatrix} c_0 x \\ c_1 x \\ \vdots \\ c_{M-1} x \end{bmatrix} \tag{1}$$

연산 공유 승산 알고리즘은 벡터 스케일링 연산 시 상수 벡터를 상위와 하위 4비트씩 나누어 계산하며, 4비트 상수에 대한 스케일러의 승산 값을 공유함으로써 하드웨어의 크기를 줄이는데 목적을 두고 있다. 예를 들어, 상수 벡터 $c=[c_0, c_1, c_2]$와 스케일러 x에 대한 승산을 한다고 가정하자. 이때 상수의 값은 각각 $c_0 = 01100111$, $c_1 = 10001011$, $c_2 = 11100100$이다. 상수 벡터를 상위 4비트와 하위 4비트로 나누어 승산 하면 그 결과는 식 (2)와 같은 형태로 표현된다.

$$x \cdot \begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} c_0 x \\ c_1 x \\ c_2 x \end{bmatrix} = \begin{bmatrix} 2^4(0110)x + (0111)x \\ 2^4(1000)x + (1011)x \\ 2^4(1110)x + (0100)x \end{bmatrix} = \begin{bmatrix} 2^4\{(0011)x << 1\} + (0111)x \\ 2^4\{(0001)x << 3\} + (1011)x \\ 2^4\{(0111)x << 1\} + (0111)x << 2x\} \end{bmatrix} \tag{2}$$

Alpabet* = *0001, 0011, 0111, 1011
***Alpabet set* = { *0001, 0011, 0111, 1011* }**

이 중 $(0001)x$, $(0011)x$, $(0111)x$, $(1011)x$의 승산 결과가 계산되어 있다면 벡터 스케일링 연산은 단지 시프트와 가산 연산으로 수행될 수 있다.

여기서, 0001, 0011, 0111, 1011을 알파벳(alphabet)이라 정의하고, 이 알파벳의 묶음 {0001, 0011, 0111, 1011}을 알파벳 셋(alphabet set)이라 한다.

2.2 제안된 연산 공유 승산 알고리즘

기존의 알고리즘은 벡터 스케일링 연산 시 상수 벡터를 상위와 하위 4비트씩 나누어 계산하나 제안된 알고리즘은 2비트씩 나누어 처리한다. 임의의 상수가 n비트 크기를 갖는다면 상수 c_i는 식 (3)과 같이 정의될 수 있다.

$$c_i = \sum_{k=0}^{\frac{n}{2}-1} 2^{2k} a_j^i \tag{3}$$

여기서, n은 짝수라 가정하고, a_j^i는 00, 01, 10, 11 중 하나의 값을 가진다.

식 (1)를 스케일러 x와 상수 c_i의 승산으로 나타내면 식 (4)와 같다.

$$x \cdot c_i = x \cdot \sum_{k=0}^{\frac{n}{2}-1} 2^{2k} a_j^i \tag{4}$$

이를 M개의 상수를 가지는 상수 벡터와 스케일러의 승산으로 확장하면 식 (5)와 같다.

$$x \cdot c = x \cdot \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_{M-1} \end{bmatrix} = \begin{bmatrix} \sum_{k=0}^{\frac{n}{2}-1} 2^{2k} a_j^0 \cdot x \\ \sum_{k=0}^{\frac{n}{2}-1} 2^{2k} a_j^1 \cdot x \\ \vdots \\ \sum_{k=0}^{\frac{n}{2}-1} 2^{2k} a_j^{M-1} \cdot x \end{bmatrix} \tag{5}$$

식 (5)의 a_j^0, a_j^1, $\cdots$, a_j^{M-1}은 00, 01, 10, 11 중 하나의 값으로 구성되는데, 00은 0의 값을 가지고 10은 01을 왼쪽으로 1비트 시프트하여 구현할 수 있으므로 01과 11 2개의 알파벳만으로 벡터 스케일링 연산이 가능하다. 예를 들어, 상수의 값이 각각 $c_0 = 10100111$, $c_1 = 10010010$, $c_2 = 01101011$, $c_3 = 11100101$이라면, 벡터 스케일링 연산은 식 (6)과 같이 나타내어 진다.

식 (6)에서 알 수 있듯이 제안된 알고리즘은 상수 벡터를 2비트씩 처리함으로써 벡터 스케일링 연산시 사용되는 알파벳 수를 두개로 줄일 수 있다.

$$x \cdot \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} c_0 x \\ c_1 x \\ c_2 x \\ c_3 x \end{bmatrix} = \begin{bmatrix} \sum_{k=0}^{3} 2^{2k} a_j^0 \cdot x \\ \sum_{k=0}^{3} 2^{2k} a_j^1 \cdot x \\ \sum_{k=0}^{3} 2^{2k} a_j^2 \cdot x \\ \sum_{k=0}^{3} 2^{2k} a_j^3 \cdot x \end{bmatrix} \quad (6)$$

$$= \begin{bmatrix} 2^6(10)x + 2^4(10)x + 2^2(01)x + (11)x \\ 2^6(10)x + 2^4(01)x + 2^2(00)x + (10)x \\ 2^6(01)x + 2^4(10)x + 2^2(10)x + (11)x \\ 2^6(11)x + 2^4(10)x + 2^2(01)x + (11)x \end{bmatrix}$$

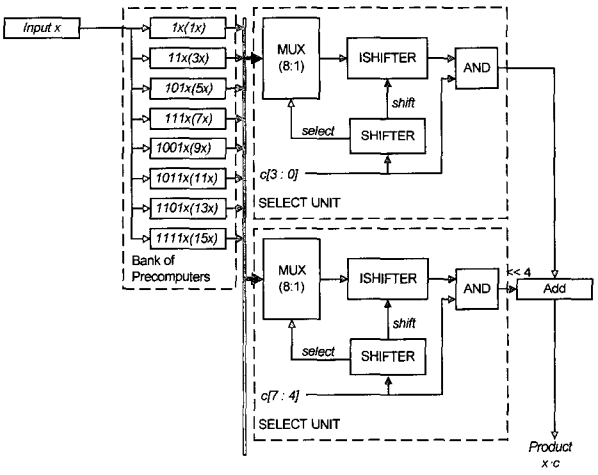
$$= \begin{bmatrix} 2^6\{(01)\ll 1\} + 2^4\{(01)\ll 1\} + 2^2(01)x + (11)x \\ 2^6\{(01)\ll 1\} + 2^4(01)x + \{(01)x \ll 1\} \\ 2^6(01)x + 2^4\{(01)x \ll 1\} + 2^2\{(01)x \ll 1\} + (11)x \\ 2^6(11)x + 2^4\{(01)x \ll 1\} + 2^2(01)x + (01)x \end{bmatrix}$$

Alpabet = 01, 11
Alpabet set = {01, 011}

3. 연산 공유 승산기 구조

3.1 기존의 연산 공유 승산기 구조

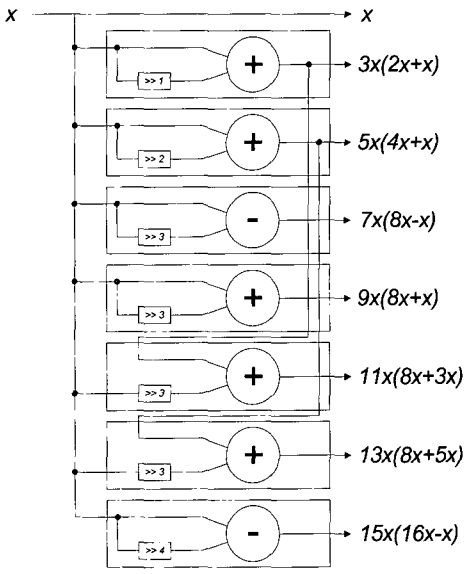
기존의 연산 공유 승산기는 크게 전처리기, 선택기 그리고 가산기의 3블록으로 구성된다. (그림 1)은 연산 공유 승산기의 구조를 나타낸 것으로 각 블록의 동작은 다음과 같다.



(그림 1) 연산 공유 승산기 구조

① 전처리기 : 스케일러와 알파벳간의 승산을 수행하여 벡터 스케일링 연산의 결과를 미리 계산하는 부분으로 (그림 2)의 구조를 가진다. 8비트 상수를 상위와 하위 4비트씩 나누었을 때 승산 값은 $1x \sim 15x$이다. 그러나, 이들 중 $1x$는 ($2x$, $4x$, $8x$), $3x$는 ($6x$, $12x$), $5x$는 $10x$, 그리고 $7x$는 $14x$의 값을 나타낼 수 있기 때문에 실제 필요한 알파벳 셋은 {1, 3, 5, 7, 9, 11, 13, 15}가 된다.

② 선택기 : 전처리기에서 나온 승산 결과를 상수의 상위 4비트와 하위 4비트에 따라 선택하고 시프트 시키는 부분으로 멀티플렉서, Shifter, Ishifter, 그리고 And로 구성된다. 멀티플렉서는 Shifter의 select 신호에 따라 전처리기의 값을 선택한다. <표 1>은 멀티플렉서의 동작을 나타낸다.



(그림 2) 기존의 전처리기의 구조

〈표 1〉 Select 신호에 따른 멀티플렉서 출력

Select input	Mux output
000	1x
001	3x
010	5x
011	7x
100	9x
101	11x
110	13x
111	15x

〈표 2〉 상수에 따른 select와 shift 출력

Coefficient	Select signal	Shift signal
0000	ddd	dd
0001	000	00
0010	000	01
0011	001	00
0100	000	10
0101	010	00
0110	001	01
0111	011	00
1000	000	11
1001	100	00
1010	010	01
1011	101	00
1100	001	10
1101	110	00
1110	011	01
1111	111	00

Shifter는 멀티플렉서의 select 신호 및 Ishifter의 shift 신호를 생성한다. <표 2>는 상수에 따른 select와 shift 출력을 나타낸다. Ishifter는 shift신호에 따라 멀티플렉서의 값을 이동시키며 And 게이트는 상수가 0000인 경우 출력을 정의한다

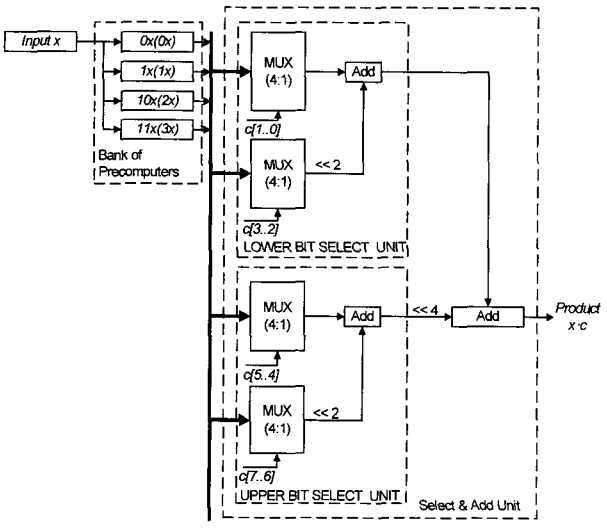
③ 가산기 : 선택기 출력의 상위 4비트를 왼쪽으로 4비트 시프트 시킨 후 가산하여 최종 승산 결과를 구한다.

3.2 제안된 연산 공유 승산기 구조

기존의 연산 공유 승산 알고리즘은 하드웨어 구현시 다음 3가지 문제점을 가진다.

① 4비트 상수에 대해 8개의 알파벳을 필요로 하며, 전처리기 구현시 5개의 가산기와 2개의 감산기가 필요하므로 하드웨어 복잡도가 크다.
② 전처리기의 특정 출력 $11x$, $13x$의 경우 $3x$, $5x$를 입력받아 계산하므로 승산기의 지연시간이 커질 수 있다.
③ 선택기의 Shifter 및 Ishifter 구현을 위한 하드웨어 용량이 크다.

이러한 문제점을 해결하기 위해 하드웨어 복잡도 및 처리속도를 향상시킨 새로운 구조의 연산 공유 승산기를 제안한다. 제안된 승산기 구조는 (그림 3)과 같다. 전처리기 및 선택기를 기존 승산기와 비교 설명하면 다음과 같다.



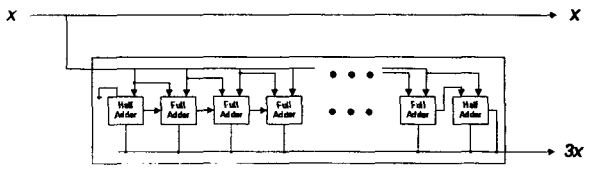
(그림 3) 제안된 연산 공유 승산기 구조

① 전처리기 : 제안된 전처리기는 상수를 2비트씩 처리하므로 2개의 알파벳과 1개의 가산기 만으로 구현 가능하다. 특히, $11x$, $13x$와 같은 출력을 배제하므로 기존 전처리기 보다 간단한 하드웨어 구조를 가지며 동작 속도 또한 향상된다. <표 3>은 기존 전처리기와 제안된 전처리기의 하드웨어를 비교한 것이다.

<표 3> 기존 전처리기와 제안된 전처리기의 하드웨어 비교

	기존 전처리기	제안된 전처리기
Alphabet set	{0001, 0011, 0101, 0111, 1001, 1011, 1101, 1111}	{01, 11}
Adder	5	1
Subtracter	2	-

(그림 4)는 제안된 전처리기의 구조로 동작 속도의 향상을 위해 CLA(Carry Look-ahead Adder)나 CSA(Carry Save Adder)를 사용할 수 있으나 하드웨어 복잡도가 증가하므로 본 논문에서는 가장 간단한 구조를 가진 RCA(Ripple Carry Adder)를 사용하여 구현하였다.



(그림 4) 제안된 전처리기의 구조

② 선택기 : <표 4>는 기존 선택기와 제안된 선택기의 하드웨어를 비교하여 나타낸 것이다. 제안된 선택기는 기존 선택기의 8×1 멀티플렉서를 2개의 4×1 멀티플렉서로 대체하고, Shifter및 Ishifter를 제거한 대신 1개의 가산기를 사용한다.

<표 4> 기존 선택기와 제안된 선택기의 하드웨어 비교

	기존 선택기	제안된 선택기
8×1 MUX	1	-
4×1 MUX	-	2
Shifter/Ishifter	1/1	-
Adder	-	1

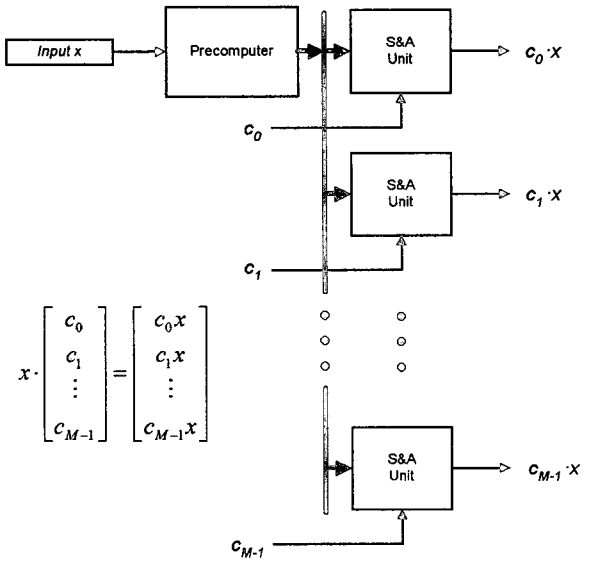
새로운 선택기는 Shifter와 Ishifter를 사용하지 않으므로 하드웨어 구현이 간단한 장점이 있으나 가산기를 추가하므로 선택기의 동작 시간이 증가되는 단점이 있을 수 있다. 그러나 추가되는 가산기는 4비트 연산을 하므로 Shifter와 Ishifter의 제거로 얻을 수 있는 동작 속도 향상으로 충분히 보상 가능하다. <표 5>는 select신호에 따른 4×1 멀티플렉서의 동작을 나타낸 것으로, 8비트의 상수는 2비트씩 나누어져 멀티플렉서의 select 입력으로 사용된다.

<표 5> Select 신호에 따른 41 멀티플렉서의 출력

Select input	MUX output
00	0
01	$1x$
10	$2x\,(x<<1)$
11	$3x$

3.3 벡터 스케일러 구조

벡터 스케일러(vector scalar)는 스케일러(x)와 M개의 상수($c=[c_0, c_1, \cdots, c_{M-1}]$)에 대한 승산을 위한 회로이다. 벡터 스케일러는 하나의 전처리기를 M개의 선택 & 가산기(Select & Adder unit)가 공유하는 구조를 갖는다. (그림 5)는 식 (1)을 수행하기 위한 벡터 스케일러 구조를 나타낸다.



(그림 5) 벡터 스케일러 구조

4. 연산 공유 승산기를 이용한 DCT 프로세서 구조

4.1 DCT 알고리즘

$N \times N$ 화소 블록 $x(i, j)$에 대한 2차원 DCT 변환 계수 $y(u, v)$는

$$y(u, v) = \frac{2}{N} C(u) C(v) \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} x(i, j) \cos\left[\frac{(2i+1)v\pi}{2N}\right] \cos\left[\frac{(2j+1)u\pi}{2N}\right] \quad (7)$$

$$C(k) = \begin{cases} \frac{1}{\sqrt{2}} & k = o \\ 1 & otherwise \end{cases}$$

로 정의되고, 이를 행-열 분해(row-column decomposition)를 이용하여 행렬 형태로 표현하면 식 (8)과 같다.

$$y = C x C^T = C[C x^T]^T \quad (8)$$

여기서 $w = C x^T$라 하면, $y = C[C x^T]^T = C w^T$가 된다. 즉, 입력 x에 대하여 1차원 DCT 수행 후 행렬의 전치, 그리고 1차원 DCT의 과정을 거침으로서 2차원 DCT의 결과를 얻을 수 있다.

식 (8)에서 행렬 C의 기저벡터 원소 C_{ij}는

$$C_{ij} = \sqrt{\frac{2}{N}} A(i) \cos\frac{(2j+1)\pi i}{2N} \quad (9)$$

$$A(w) = \begin{cases} \frac{1}{\sqrt{2}} & w = o \\ 1 & otherwise \end{cases}$$

이고, $N=8$일 때 1차원 DCT를 행렬식으로 나타내면 식 (10)과 같다. 여기서 x_i는 입력 화소 값, y_i는 DCT 계수 값, $C_k = \cos\frac{\pi k}{16}$ 코사인 계수 값을 나타낸다.

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{pmatrix} = \frac{1}{2} \begin{pmatrix} C_4 & C_4 & C_4 & C_4 & C_4 & C_4 & C_4 & C_4 \\ C_1 & C_3 & C_5 & C_7 & -C_7 & -C_5 & -C_3 & -C_1 \\ C_2 & C_6 & -C_6 & -C_2 & -C_2 & -C_6 & C_6 & C_2 \\ C_3 & -C_7 & -C_1 & -C_5 & C_5 & C_1 & C_7 & -C_3 \\ C_4 & -C_4 & -C_4 & C_4 & C_4 & -C_4 & -C_4 & C_4 \\ C_5 & -C_1 & C_7 & C_3 & -C_3 & -C_7 & C_1 & -C_5 \\ C_6 & -C_2 & C_2 & -C_6 & -C_6 & C_2 & -C_2 & C_6 \\ C_7 & -C_5 & C_3 & -C_1 & C_1 & -C_3 & C_5 & -C_7 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix} \quad (10)$$

위 식을 두 개의 4×4 행렬 분해하면 식 (11)과 같다.

$$\begin{pmatrix} y_0 \\ y_2 \\ y_4 \\ y_6 \end{pmatrix} = \begin{pmatrix} d & d & d & d \\ b & f & -f & -b \\ d & -d & -d & d \\ f & -b & b & -f \end{pmatrix} \begin{pmatrix} x_0 + x_7 \\ x_1 + x_6 \\ x_2 + x_5 \\ x_3 + x_4 \end{pmatrix} \quad (11a)$$

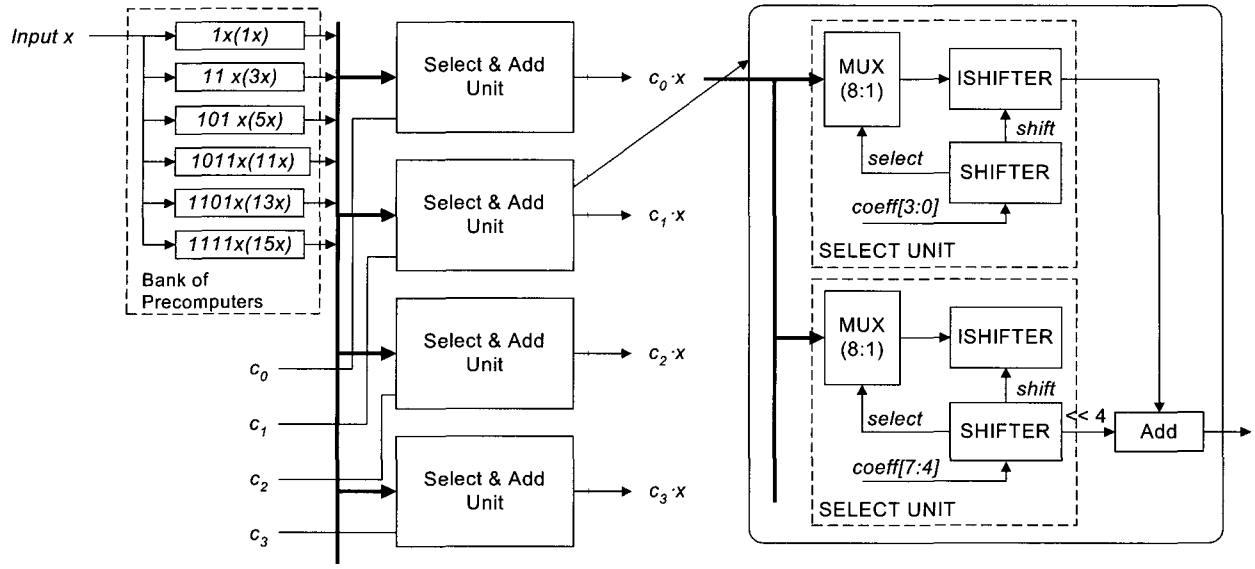
$$\begin{pmatrix} y_1 \\ y_3 \\ y_5 \\ y_7 \end{pmatrix} = \begin{pmatrix} a & c & e & g \\ c & -g & -a & -e \\ e & -a & g & c \\ g & -e & c & -a \end{pmatrix} \begin{pmatrix} x_0 - x_7 \\ x_1 - x_6 \\ x_2 - x_5 \\ x_3 - x_4 \end{pmatrix} \quad (11b)$$

(단, $a = \frac{1}{2} C_1$, $b = \frac{1}{2} C_2$, $c = \frac{1}{2} C_3$, $d = \frac{1}{2} C_4$, $e = \frac{1}{2} C_5$, $f = \frac{1}{2} C_6$, $g = \frac{1}{2} C_7$ 이다.)

식 (11)에 나타낸 바와 같이 DCT는 대부분 내적 연산으로 이루어지며 이를 하드웨어로 어떻게 설계하느냐가 DCT 프로세서의 성능을 좌우한다. 따라서 식 (11)을 연산 공유 승산기에 적용하기 위해 다음과 같이 벡터 스케일링 연산의 형태를 가진 식 (12)로 변형한다.

$$\begin{pmatrix} y_0 \\ y_2 \\ y_4 \\ y_6 \end{pmatrix} = (x_0 + x_7) \begin{pmatrix} d \\ b \\ d \\ f \end{pmatrix} + (x_1 + x_6) \begin{pmatrix} d \\ f \\ -d \\ -b \end{pmatrix} + (x_2 + x_5) \begin{pmatrix} d \\ -f \\ -d \\ b \end{pmatrix} + (x_3 + x_4) \begin{pmatrix} d \\ -b \\ d \\ -f \end{pmatrix} \quad (12a)$$

$$\begin{pmatrix} y_1 \\ y_3 \\ y_5 \\ y_7 \end{pmatrix} = (x_0 - x_7) \begin{pmatrix} a \\ c \\ e \\ g \end{pmatrix} + (x_1 - x_6) \begin{pmatrix} c \\ -g \\ -a \\ -e \end{pmatrix} + (x_2 - x_5) \begin{pmatrix} e \\ -a \\ g \\ c \end{pmatrix} + (x_3 - x_4) \begin{pmatrix} g \\ -e \\ c \\ -a \end{pmatrix} \quad (12b)$$



(그림 6) 기존의 연산 공유 승산기를 이용한 벡터 스케일러 구조-1

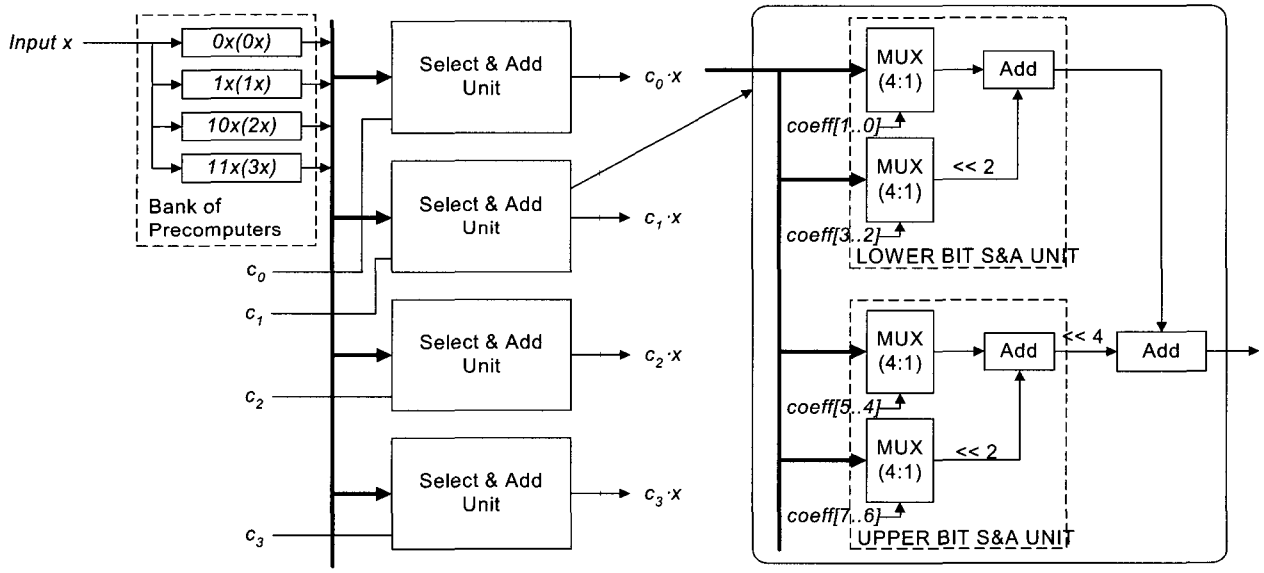
변형된 식의 $x_0 \pm x_7$, $x_1 \pm x_6$, $x_2 \pm x_5$, $x_3 \pm x_4$는 스케일러에 해당되고 $[d\ b\ d\ f]$, $[d\ f - d - b]$, $[d - f - d\ b]$, $[d - b\ d - f]$ 및 $[a\ c\ e\ g]$, $[c - g - a - e]$, $[e - a\ g\ c]$, $[g - e\ c - a]$는 상수 벡터에 해당된다. 다음 절에서는 DCT에 사용되는 벡터 스케일링 연산을 기존의 연산 공유 승산기와 제안된 연산 공유 승산기의 2가지 방식으로 설계하여 비교한다.

4.2 기존의 연산 공유 승산기를 이용한 벡터 스케일러 구조

1차원 DCT의 내적 연산을 수행하기 위한 상수는 <표 6>과 같이 7개이다. 7개의 상수를 4비트씩 나누면 연산 공유 승산기에 필요한 알파벳 수는 6개가 된다. 따라서 기존의 연산 공유 승산 알고리즘은 6개의 알파벳을 사용하여 1차원 DCT 연산을 수행한다. (그림 6)은 기존 승산기를 이용한 벡터 스케일러 구조를 나타낸 것이다. 전처리기는 6개의 알파벳을 이용하여 구성되고 4개의 상수에 대해 벡터 스케일러 형태로 구성된다.

〈표 6〉 1차원 DCT 내적 연산 수행을 위한 상수 및 알파벳 셋

Coeff	Value	Binary Numbers	Alphabet · x
a	0.49	0011 1111	$3x$, $15x$
b	0.46	0011 1011	$3x$, $11x$
c	0.42	0011 0101	$3x$, $5x$
d	0.35	0010 1101	x, $13x$
e	0.28	0010 0100	x
f	0.19	0001 1000	x
g	0.10	0000 1100	$3x$



(그림 7) 제안된 연산 공유 승산기를 이용한 벡터 스케일러 구조

4.3 제안된 연산 공유 승산기를 이용한 벡터 스케일러 구조

1차원 DCT 내적 연산을 위한 상수는 기존과 동일한 7개이다. 그러나 상수 벡터를 2비트씩 처리함으로써 전처리기에 필요한 알파벳 수를 2개로 줄여 좀 더 간단한 하드웨어 구현이 가능하다. 또한 기존 선택기에 사용된 Shifter와 Ishifter 블록이 필요 없는 장점을 가지고 있다. <표 7>은 제안된 전처리기에 사용되는 상수 및 알파벳 셋을 나타낸다.

〈표 7〉 제안된 전처리기에 사용되는 상수 및 알파벳 셋

Coeff	Value	Binary Numbers	Alphabet · x
a	0.49	00 11 11 11	0, 3x, 3x, 3x
b	0.46	00 11 10 11	0, 3x, x, 3x
c	0.42	00 11 01 01	0, 3x, x, x
d	0.35	00 10 11 01	0, x, 3x, x
e	0.28	00 10 01 00	0, x, x, 0
f	0.19	00 01 10 00	0, x, x, 0
g	0.10	00 00 11 00	0, 0, 3x, 0

제안된 연산 공유 승산기를 이용하여 4개의 상수에 대한 벡터 스케일러를 구현하면 (그림 7)과 같다.

4.4 1차원 DCT 프로세서 구조

(그림 8)은 벡터 스케일러를 이용한 1차원 DCT 프로세서 구조를 나타낸다. 1차원 DCT 프로세서는 가/감산기, 멀티플렉서, 벡터 스케일러, 누적기, 그리고 제어기로 구성된다. 가/감산기는 2개의 입력에 대해 가산과 감산을 수행하며 멀티플렉서를 통해 가산 또는 감산 값이 선택되어진다. 모든 입력은 2번씩 반복해서 들어오는데, 처음 입력은 가산 값을 선택하고 반복된 입력은 감산 값을 선택한다. 멀티플렉서의 출력과 제어기에서 출력되는 상수는 벡터 스케일러에 의해 승산 된다. 벡터 스케일링 연산 값은 누적기에 저장되며 가산 승산 또는 감산 승산에 대해 각각 4번의 가산을 수행하면 8 입력에 대한 1차원 DCT 결과가 완성된다.

(그림 9)는 1차원 DCT 프로세서 계산 흐름도를 나타낸다.

5. 성능비교 및 시뮬레이션

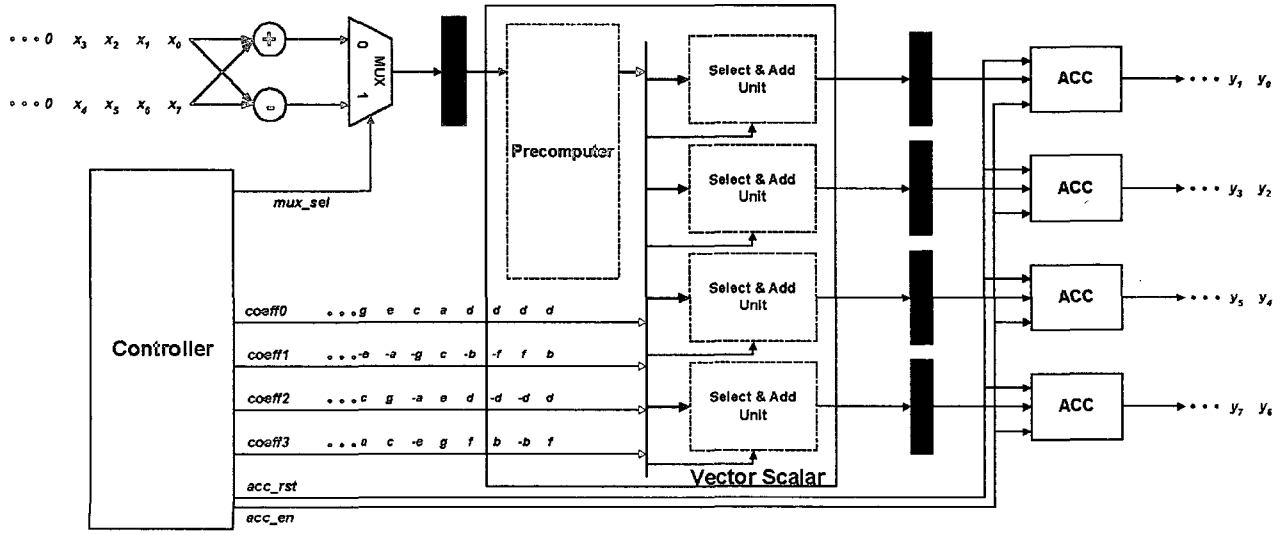
연산 공유 승산기를 이용한 1차원 DCT 프로세서는 VHDL의 기능적인 기술(behavioral modeling) 기법과 구조적인 기술(structural modeling) 기법을 사용하여 설계하였으며, 하드웨어 합성 및 시뮬레이션은 Altera의 Max + PLUS II를 사용하였다. <표 8>은 1차원 DCT 프로세서를 설계하는데 필요한 하드웨어를 기존의 방식들과 비교한 것이다. <표 8>에 나타낸 바와 같이 고속 DCT 연산 알고리즘[6-8]은 다수의 승산기를 사용하여 하드웨어 복잡도가 매우 크다. 반면, ROM 기반 분산 산술 연산과 연산 공유 승산기는 DCT 연산을 승산기 없이 수행할 수 있고, 파이프라인 설계가 용이하기 때문에 효율적인 하드웨어 설계가 가능하다.

〈표 8〉 1차원 DCT 프로세서 구현에 필요한 하드웨어 비교

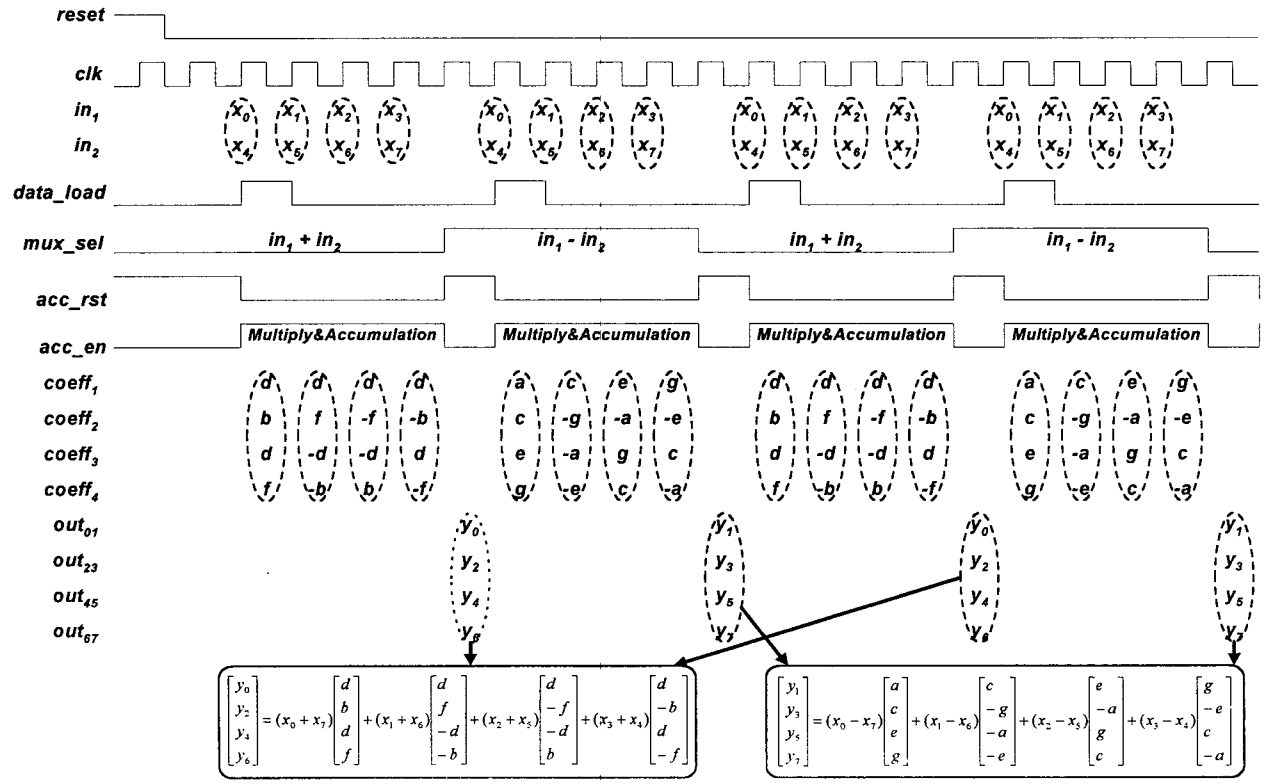
(CSHM : Computation Sharing Multiplier)

	Multiplier	Adder & Subtracter	ROM	Accumulator	Precomputer/ Select & Adder
Chen[8]	16	26	-	-	-
Lee[9]	12	29	-	-	-
Loeffler [10]	11	29	-	-	-
ROM based DA[4]	-	8	8	8	-
CSHM	-	2	-	4	1/4

<표 9>는 기존의 연산 공유 승산기와 제안된 연산 공유



(그림 8) 벡터 스케일러를 이용한 1차원 DCT 프로세서 구조



(그림 9) 1차원 DCT 프로세서 계산 흐름도

승산기를 사용하여 벡터 스케일러를 구현할 때 사용되는 하드웨어를 비교하여 나타낸 것이다. <표 9>에 나타낸 바와 같이 기존의 연산 공유 승산기는 전처리기 및 선택기의 하드웨어 복잡도가 크다. 반면, 제안된 연산 공유 승산기는 전처기의 가/감산기 수를 줄이고, 선택기의 Shifter와 Ishifter를 제거함으로써 기존에 비해 간단한 하드웨어 구조를 가진다.

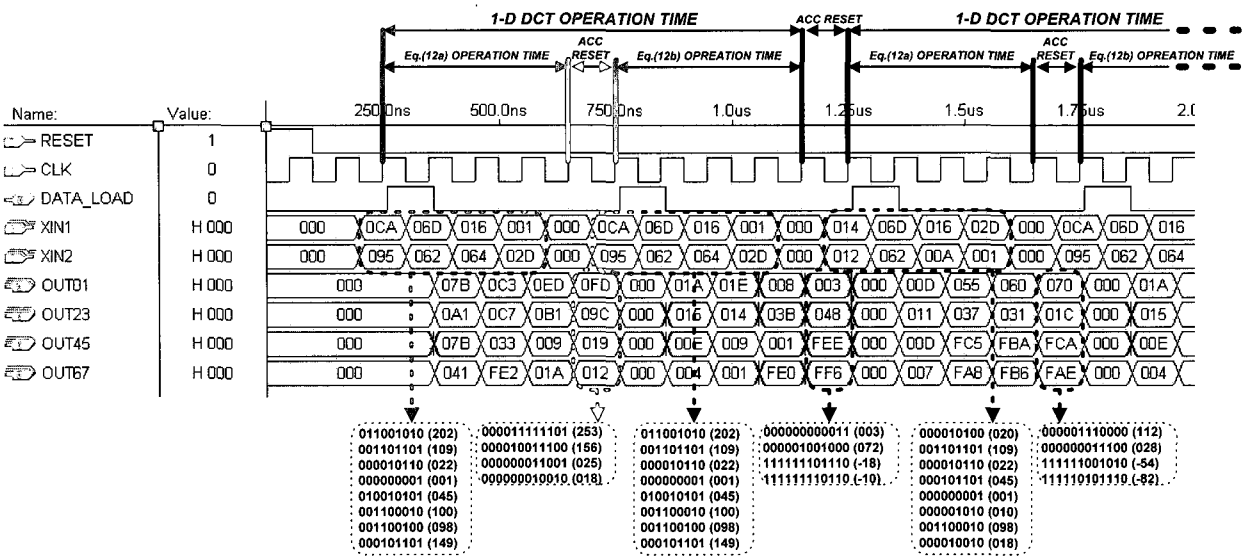
〈표 9〉 벡터 스케일러 구현시 필요한 하드웨어 비교

		기존의 연산 공유 승산기	제안된 연산 공유 승산기
Precomputer	Adder	4	2
	Subtracter	1	-
Select unit	8×1 MUX	8	-
	4×1 MUX	-	16
	Shifter/ Ishifter	8/8	-
	Adder	-	8

<표 10>은 <표 9>에 나타낸 우수성을 증명하기 위한 것으로, 기존의 연산 공유 승산기와 제안된 연산 공유 승산기를 상수 개수 및 비트 수에 따라 벡터 스케일러를 구현한 후, Altera의 Max + PLUSII를 통한 논리 합성을 수행하여 사용되는 로직 셀을 비교한 것이다.

〈표 10〉 상수 개수 및 비트 수에 따른 벡터 스케일러 로직 셀 비교

상수 개수	비 트	기존의 벡터 스케일러	제안된 벡터 스케일러	로직 셀 감소(%)
2	4	257	114	55.64
	8	491	285	41.96
	12	725	415	42.76
	16	969	575	40.66
4	4	459	222	51.63
	8	921	563	38.87
	12	1389	823	40.75
	16	1877	1117	40.49
6	4	661	330	50.08
	8	1351	841	37.75
	12	2053	1209	41.11
	16	2763	1671	39.52
8	4	863	438	49.25
	8	1781	1073	39.75
	12	2695	1609	40.30
	16	3663	2225	39.26
10	4	1065	546	48.73
	8	2193	1339	38.94
	12	3353	2009	40.08
	16	4563	2779	39.10



(그림 10) 제안된 연산 공유 승산기를 이용한 1차원 DCT 프로세서 시뮬레이션 파형

<표 10>에 의하면 제안된 연산 공유 승산기는 기존의 연산 공유 승산기 보다 평균 42%의 로직 셀을 절약함을 확인할 수 있다. 따라서 제안된 연산 공유 승산기가 기존의 것 보다 내적 연산의 하드웨어 구현에 훨씬 유리함을 알 수 있다.

<표 11>은 ROM 기반 분산 산술 연산, 기존의 연산 공유 승산기, 그리고 제안된 연산 공유 승산기를 이용하여 1차원 DCT 프로세서를 설계한 후 사용되는 로직 셀과 클럭 주기 및 동작 주파수를 비교한 것이다. <표 8>에서 설명한 바와 같이 ROM 기반 분산 산술 연산이 기존의 방식 보다 우수하다고 가정한다면, 기존의 연산 공유 승산기를 이용한 DCT 프로세서는 ROM 기반 분산 산술 연산 보다 로직 셀 사용도는 우수하나 동작 주파수는 약간 감소하는 단점이 있다. 반면, 제안된 연산 공유 승산기는 ROM 기반 분산 산술 연산과 기존 승산기에 비해 로직 셀 사용도와 동작 주파수 모두 우수함을 확인 할 수 있다.

〈표 11〉 3가지 1차원 DCT 프로세서 비교

	1-D DCT by using ROM based DA	1-D DCT by using conventional CSHM	1-D DCT by using proposed CSHM
Device	EPF10K40RC240-3	EPF10K20TC144-3	EPF10K20TC144-3
Input bit	9	9	9
Output bit	12	12	12
Logic cells required	1492	1065	721
Clock period	81.3ns	99.8ns	77.4ns
Flip flops required	309	93	93
Operating Freq.	12.30MHz	10.02MHz	12.91MHz

(그림 10)은 제안된 연산 공유 승산기를 이용한 1차원 DCT 프로세서의 시뮬레이션 파형을 나타낸 것이다. (그림 9)의 계산 흐름도와 같이 초기 입력이 들어간 후 4클럭 후 식 (12a)가 완료되고 누적기 초기화 후 다시 4클럭 후 식 (12b)가 완료된다. 1차원 DCT 연산은 초기 입력 후 9클럭이 지나면 완료된다.

6. 결 론

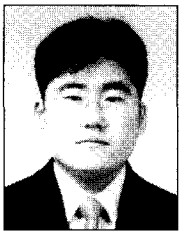
본 논문에서는 새로운 구조의 연산 공유 승산기를 제안하고 이를 1차원 DCT 프로세서에 적용하였다. 기존의 연산 공유 승산기는 전처리기 및 선택기의 하드웨어 복잡도가 커 승산기의 성능이 저하되는 문제점을 가진다. 새로운 연산 공유 승산기는 상수 벡터를 2비트씩 처리함으로써 전처리기에 사용되는 가/감산기 개수 및 지연 시간을 줄이고 선택기의 구조를 간단화 하였다. 승산기의 구조 및 로직 셀 사용도 비교시 제안된 승산기의 하드웨어 구조가 효율적이며, 로직 셀 사용도 또한 우수함을 확인하였다. 1차원 DCT 프로세서의 논리합성 결과 ROM 기반 분산 산술 연산과 기존의 연산 공유 승산기에 비해 제안된 승산기가 하드웨어 복잡도 및 동작 주파수 모두 뛰어난 성능을 나타내었다. 제안된 연산 공유 승산기는 DCT뿐 아니라 DSP 알고리즘에 많이 사용되는 내적 및 행렬의 승산 등에 응용 가능하다.

참 고 문 헌

[1] J. S. Park, S. K Kwon and K. Roy, "Low power reconfigurable DCT design based on sharing multiplication," In Proc. IEEE ICASS, Vol.3, pp.3116-3119, 2002.

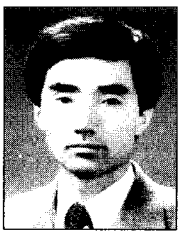
[2] S. K. Kwon, J. S. Park and K. Roy, "DCT processor ar-

chitecture based on computation sharing," In Proc. IEEE ICCSC, pp.162-165, 2002.
[3] S. A. White, "Applications of distributed arithmetic to digital signal processing : A tutorial review," IEEE ASSP Magazine, pp.1-19, Jul., 1989.
[4] G. M. Blair and G. S. Taylor, "Design for the discrete cosine transform in VLSI," IEE Proc. Comput. Digit. Tech., Vol. 145, No.2, pp.127-133, Mar., 1998.
[5] Bernie New, "A distributed arithmetic approach to designing scalable DSP chips," EDN Design Feature, Vol.Aug.-17, pp.107-114, Aug., 1995.
[6] K. Muhammad and K. Roy, "Reduced computational redundancy implementation of DSP algorithms using computation sharing vector scaling," IEEE Trans. on VLSI, Vol.10, pp.292-300, Jun., 2002.
[7] K. Muhammad and K. Roy, "Minimally redundant parallel implementation of digital filters and vector scaling," IEEE International Conference on Acoustics, Speech, and Signal Processing, Vol.6, pp.3295-3298, 2000.
[8] W. H. Chen, C. H. Smithm and C. C. Fralick, "A fast computational algorithm for the discrete transform," IEEE Trans. Commun., Vol.COM-25, pp.1004-1008, Sep., 1977.
[9] 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.
[10] C. Loeffler, A. Ligtenberg and G. S. Moschytz, "Practical fast DCT algorithm with 11 multiplications," In Proc. IEEE ECASSP, Vol.2, pp.988-991, Feb., 1989.



이 태 욱

e-mail : twlee00@korea.com
1998년 울산대학교 전자공학과(공학사)
2000년 울산대학교 대학원 전자공학과 (공학석사)
2000년~현재 울산대학교 대학원 전자공학과 박사과정
관심분야 : ASIC 설계, 디지털 신호처리회로 설계, 영상 처리 프로세서 설계 등



조 상 복

e-mail : sbcho@mail.ulsan.ac.kr
1979년 한양대학교 전자공학과(공학사)
1981년 한양대학교 대학원 전자공학과 (공학석사)
1985년 한양대학교 대학원 전자공학과 (공학박사)
1994년~1995년 Univ. of Texas, Austin 초빙학자
1986년~현재 울산대학교 전기전자정보시스템공학부 교수
관심분야 : ASIC 설계, 자동차 전자회로 설계, 비전 시스템 개발, 테스트 및 테스트 용이한 설계, 메모리테스트 등