연산공유 승산 알고리즘을 이용한 내적의 최적화 및 이를 이용한 1차원 DCT 프로세서 설계

論 文
53D-2-3

Optimization Design Method for Inner Product Using CSHM Algorithm and its Application to 1-D DCT Processor

李 泰 旭* · 趙 相 福**
(Tae-Wook Lee · Sang-Bock Cho)

Abstract - The DCT algorithm needs an efficient hardware architecture to compute inner product. The conventional design method, like ROM-based DA(Distributed Arithmetic), has large hardware complexity. Because of this reason, a CSHM(Computation SHaring Multiplication) was proposed for implementing inner product by Park. However, the Park's CSHM has inefficient hardware architecture in the precomputer and select units. Therefore it degrades the performance of the multiplier. In this paper, we presents the optimization design method for inner product using CSHM algorithm and applied it to implementation of 1-D DCT processor. The experimental results show that the proposed multiplier is more efficient than Park's when hardware architectures and logic synthesis results were compared. The designed 1-D DCT processor by using proposed design method is more high performance than typical methods.

Key Words : 승산기(Multiplier), 연산공유 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, 영상처리 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]. 내적연산의 하드웨어 구현을 위해 최근까지 연구된 방법은 크게 두 가지로 분류될 수 있다. 첫째, 내적연산에 사용되는 승산을 최소화시키는 알고리즘에 관한 연구와 둘째, 내적연산의 하드웨어 구조 최적화에 관한 연구이다. 하드웨어 최적화 방법 중 ROM-기반 분산산술(ROM-based DA, Distributed Arithmetic)은 입력변수를 비트단위로 확장하여 입력의 각 비트에 따른 부분 승산값을 ROM에 저장하는 방식이다. 이 방식은 기존의 여타 구조들에 비해 내적의 하드웨어 구현에 가장 효과적이라고 인정되고 있다[3]-[5]. 하지만 ROM-기반 분산산술은 입력의 개수 및 연산의 정확도 증가에 따라 ROM의 크기가 지수적으로 증가하는 단점을 가지고 있다[1]. 최근 들어 내적연산에 ROM-기반 분산산술 대신 연산공유 승산기를 사용한 연구들이 Park등에 의해 발표되었다[1,2],[6,7]. 연산공유 승산 알고리즘은 하나의 변수와 다수의 계수간 승산을 나타내는 벡터 스케일링 연산(vector scaling operation)을 이용하여 내적연산을 벡터 스케일링 연산의 가산형태로 바꾸어 계산한다. 연산공유 승산 알고리즘을 이용한 승산기는 정해진 계수에 대해서 뿐 아니라 모든 계수에 대한 승산이 가능하고 회로의 복잡도 및 소비전력을 효과적으로 줄일 수 있어 기존에 널리 사용되던 ROM-기반 분산산술보다 내적의 하드웨어 구현에 있어 더 효과적이다[1,2]. 하지만 Park등의 연산공유 승산 알고리즘은 하드웨어 구현시 전처리기에 다수의 가감산이 쓰이고 선택기에 필요이상의 부가적인 회로를 필요로 한다. 따라서 본 논문에서는 기존의 승산 알고리즘의 장점을 살리면서 하드웨어 복잡도 및 동작속도 측면에서 우수한 성능을 가지는 새로운 알고리즘 및 하드웨어 구조를 제안하고 이를 1차원 DCT 프로세서에 적용하였다. 본 논문의 구성은 다음과 같다. 2장에서는 효율적인 내적연산을 위한 연산공유 승산 알고리즘을 제시한다. 3장에서는 2장에서 제시된 알고리즘을 이용한 내적 연산기의 구조에 대해 살펴본다. 그리고 4장에서는 연산공유 승산기를 이용한 DCT 프로세서의 구조에 대해 살펴보고 5장에서 효용성 검증을 위한 성능비교 및 시뮬레이션을 수행한다. 마지막으로 6장에서는 연구 결과에 대한 결론을 맺는다.

2. 연산공유 승산 알고리즘

연산공유 승산 알고리즘은 내적연산을 좀더 효율적인 하드웨어 구조로 설계 하기위해 제안된 것으로 행렬승산 형태인 내적연산을 벡터 스케일링 연산으로 바꾸어 가산과 시프트 연산만으로 내적을 구한다. 일반적인 내적연산은 식(1)과 같이 $L \times L$ 행렬의 승산형태로 표현할 수 있다.

* 準 會 員 : 蔚山 大學校 博士課程
** 正 會 員 : 蔚山 大學校 敎授
接受日字 : 2003年 8月 4日
最終完了 : 2003年 12月 6日

$$Y = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{L-1} \end{bmatrix} = \begin{bmatrix} a_{0,0} & a_{0,1} & \cdots & a_{0,L-1} \\ a_{1,0} & a_{1,1} & \cdots & a_{1,L-1} \\ \vdots & \vdots & & \vdots \\ a_{L-1,0} & a_{L-1,1} & \cdots & a_{L-1,L-1} \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{L-1} \end{bmatrix} = \begin{bmatrix} a_{0,0}x_0 + a_{0,1}x_1 + \cdots + a_{0,L-1}x_{L-1} \\ a_{1,0}x_0 + a_{1,1}x_1 + \cdots + a_{1,L-1}x_{L-1} \\ \vdots \\ a_{L-1,0}x_0 + a_{L-1,1}x_1 + \cdots + a_{L-1,L-1}x_{L-1} \end{bmatrix} \quad (1)$$

식(1)은 다수의 변수 및 계수가 승산되는 형태를 가지고 있어 계산이 복잡하다. 계산의 복잡성을 줄이기 위한 방법으로 하나의 변수에 여러 개의 계수를 승산하는 벡터 스케일링 연산을 식(2)와 같이 정의한다.

$$Y_j = x_j \begin{bmatrix} a_{0,j} \\ a_{1,j} \\ \vdots \\ a_{L-1,j} \end{bmatrix} \quad (2)$$

단, $j = 0, 1, \ldots, L-1$이다.)

식(2)를 이용하여 식(1)을 다시 나타내면 벡터 스케일링 연산의 가산 형태인 식(3)을 얻을 수 있다.

$$Y = Y_0 + Y_1 + \cdots + Y_{L-1} = x_0 \begin{bmatrix} a_{0,0} \\ a_{1,0} \\ \vdots \\ a_{L-1,0} \end{bmatrix} + x_1 \begin{bmatrix} a_{0,1} \\ a_{1,1} \\ \vdots \\ a_{L-1,1} \end{bmatrix} + \cdots + x_{L-1} \begin{bmatrix} a_{0,L-1} \\ a_{1,L-1} \\ \vdots \\ a_{L-1,L-1} \end{bmatrix} \quad (3)$$

이때, 변수에 승산되는 각 계수 a_{ij}를 n비트의 크기를 가진 이진수 형태로 식(4)와 같이 정의하여 식(2)에 대입하면, 벡터 스케일링 연산은 식(5)와 같이 표현된다.

$$a_{i,j} = \sum_{k=0}^{\frac{n}{2}-1} A_{i,j}^{2k} 2^{2k} \quad (4)$$

단, n은 짝수이고 $i, j = 0, 1, \ldots, L-1$이다.
계수 $A^{2k}_{i,j}$는 00, 01, 10, 11중 하나의 값을 가진다.)

$$Y_j = x_j \begin{bmatrix} a_{0,j} \\ a_{1,j} \\ \vdots \\ a_{L-1,j} \end{bmatrix} = \begin{bmatrix} \sum_{k=0}^{\frac{n}{2}-1} A_{0,j}^{2k} 2^{2k} \\ \sum_{k=0}^{\frac{n}{2}-1} A_{1,j}^{2k} 2^{2k} \\ \vdots \\ \sum_{k=0}^{\frac{n}{2}-1} A_{L-1,j}^{2k} 2^{2k} \end{bmatrix} \quad (5)$$

식(5)는 계수 a_{ij}를 2비트의 계수 A^{2k}_{ij}로 분해하여 변수 x_j와 승산한 것으로 이를 행렬의 승산형태로 나타내면 식(6)과 같다.

$$Y_j = x_j \left\{ \begin{bmatrix} A_{0,j}^{n-2} & A_{0,j}^{n-4} & \cdots & A_{0,j}^{0} \\ A_{1,j}^{n-2} & A_{1,j}^{n-4} & \cdots & A_{1,j}^{0} \\ \vdots & \vdots & & \vdots \\ A_{L-1,j}^{n-2} & A_{L-1,j}^{n-4} & \cdots & A_{L-1,j}^{0} \end{bmatrix} \cdot \begin{bmatrix} 2^{n-2} \\ 2^{n-4} \\ \vdots \\ 2^0 \end{bmatrix} \right\} \quad (6)$$

식(6)을 행렬 승산하여 그 결과를 다시 나타내면 식(7)과 같다.

$$Y_j = \begin{bmatrix} 2^{n-2}(A_{0,j}^{n-2}x_j) + 2^{n-4}(A_{0,j}^{n-4}x_j) + \cdots + 2^0(A_{0,j}^{0}x_j) \\ 2^{n-2}(A_{1,j}^{n-2}x_j) + 2^{n-4}(A_{1,j}^{n-4}x_j) + \cdots + 2^0(A_{1,j}^{0}x_j) \\ \vdots \\ 2^{n-2}(A_{L-1,j}^{n-2}x_j) + 2^{n-4}(A_{L-1,j}^{n-4}x_j) + \cdots + 2^0(A_{L-1,j}^{0}x_j) \end{bmatrix} \quad (7)$$

식(7)을 살펴보면 다음과 같은 사실을 알 수 있다. 벡터 스케일링 연산의 계수 $A^{2k}_{i,j}$는 00, 01, 10, 11중 하나의 값을 가지므로 $(00)x_j$, $(01)x_j$, $(10)x_j$, $(11)x_j$의 값을 전처리 단계에서 미리 계산하여 공유한다면 벡터 스케일링 연산은 단순히 가산과 시프트만으로 수행될 수 있고 전처리 단계의 승산결과 공유로 인해 하드웨어 크기를 줄일 수 있다. 또한, $(00)x_j$는 0의 값을 가지고 $(10)x_j$는 $(01)x_j$를 왼쪽으로 1비트 시프트하여 구현할 수 있으므로 실제 전처리 단계에서 필요한 연산은 $(01)x_j$, $(11)x_j$으로 줄어들게 된다.

최종 내적연산을 나타내는 식(3)은 식(6) 및 식(7)을 사용하여 식(8) 또는 식(9)의 형태로 나타낼 수 있다. 내적연산은 벡터 스케일링 연산의 가산형태로 구성되므로 하나의 벡터 스케일러와 누산기만으로 전체 내적이 가능하여 행렬의 승산형태를 이용하는 것보다 하드웨어 구조가 간단하다는 장점이 있다.

$$\begin{aligned} Y = {} & x_0 \left\{ \begin{bmatrix} A_{0,0}^{n-2} & A_{0,0}^{n-4} & \cdots & A_{0,0}^{0} \\ A_{1,0}^{n-2} & A_{1,0}^{n-4} & \cdots & A_{1,0}^{0} \\ \vdots & \vdots & & \vdots \\ A_{L-1,0}^{n-2} & A_{L-1,j}^{n-4} & \cdots & A_{L-1,0}^{0} \end{bmatrix} \cdot \begin{bmatrix} 2^{n-2} \\ 2^{n-4} \\ \vdots \\ 2^0 \end{bmatrix} \right\} \\ & + x_1 \left\{ \begin{bmatrix} A_{0,1}^{n-2} & A_{0,1}^{n-4} & \cdots & A_{0,1}^{0} \\ A_{1,1}^{n-2} & A_{1,1}^{n-4} & \cdots & A_{1,1}^{0} \\ \vdots & \vdots & & \vdots \\ A_{L-1,1}^{n-2} & A_{L-1,1}^{n-4} & \cdots & A_{L-1,1}^{0} \end{bmatrix} \cdot \begin{bmatrix} 2^{n-2} \\ 2^{n-4} \\ \vdots \\ 2^0 \end{bmatrix} \right\} + \cdots \\ & + x_{L-1} \left\{ \begin{bmatrix} A_{0,L-1}^{n-2} & A_{0,L-1}^{n-4} & \cdots & A_{0,L-1}^{0} \\ A_{1,L-1}^{n-2} & A_{1,L-1}^{n-4} & \cdots & A_{1,L-1}^{0} \\ \vdots & \vdots & & \vdots \\ A_{L-1,L-1}^{n-2} & A_{L-1,L-1}^{n-4} & \cdots & A_{L-1,L-1}^{0} \end{bmatrix} \cdot \begin{bmatrix} 2^{n-2} \\ 2^{n-4} \\ \vdots \\ 2^0 \end{bmatrix} \right\} \end{aligned} \quad (8)$$

$$\begin{aligned} Y = {} & \begin{bmatrix} 2^{n-2}(A_{0,0}^{n-2}x_0) + 2^{n-4}(A_{0,0}^{n-4}x_0) + \cdots + 2^0(A_{0,0}^{0}x_0) \\ 2^{n-2}(A_{1,0}^{n-2}x_0) + 2^{n-4}(A_{1,0}^{n-4}x_0) + \cdots + 2^0(A_{1,0}^{0}x_0) \\ \vdots \\ 2^{n-2}(A_{L-1,0}^{n-2}x_0) + 2^{n-4}(A_{L-1,0}^{n-4}x_0) + \cdots + 2^0(A_{L-1,0}^{0}x_0) \end{bmatrix} \\ & + \begin{bmatrix} 2^{n-2}(A_{0,1}^{n-2}x_1) + 2^{n-4}(A_{0,1}^{n-4}x_1) + \cdots + 2^0(A_{0,1}^{0}x_1) \\ 2^{n-2}(A_{1,1}^{n-2}x_1) + 2^{n-4}(A_{1,1}^{n-4}x_1) + \cdots + 2^0(A_{1,1}^{0}x_1) \\ \vdots \\ 2^{n-2}(A_{L-1,1}^{n-2}x_1) + 2^{n-4}(A_{L-1,1}^{n-4}x_1) + \cdots + 2^0(A_{L-1,1}^{0}x_1) \end{bmatrix} + \cdots \end{aligned} \quad (9)$$

$$+\begin{bmatrix} 2^{n-2}(A^{n-2}_{0,L-1}x_{L-1})+2^{n-4}(A^{n-4}_{0,L-1}x_{L-1})+\cdots+2^{0}(A^{0}_{0,L-1}x_{L-1}) \\ 2^{n-2}(A^{n-2}_{1,L-1}x_{L-1})+2^{n-4}(A^{n-4}_{1,L-1}x_{L-1})+\cdots+2^{0}(A^{0}_{1,L-1}x_{L-1}) \\ \vdots \\ 2^{n-2}(A^{n-2}_{L-1,L-1}x_{L-1})+2^{n-4}(A^{n-4}_{L-1,L-1}x_{L-1})+\cdots+2^{0}(A^{0}_{L-1,L-1}x_{L-1}) \end{bmatrix}$$

예를 들어, 계수의 값이 각각 $a_{0,0}$=1010, $a_{0,1}$=1011, $a_{0,2}$=1101, $a_{1,0}$=1110, $a_{1,1}$=0111, $a_{1,2}$=1011, $a_{2,0}$=1111, $a_{2,1}$=0101, $a_{2,2}$=1011이라면 내적연산은 식(10)과 같이 표현할 수 있다.

$$Y=\begin{bmatrix} y_0 \\ y_1 \\ y_2 \end{bmatrix}=\begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} \\ a_{1,0} & a_{1,1} & a_{1,2} \\ a_{2,0} & a_{2,1} & a_{2,2} \end{bmatrix}\cdot\begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = x_0\begin{bmatrix} a_{0,0} \\ a_{1,0} \\ a_{2,0} \end{bmatrix}+x_1\begin{bmatrix} a_{0,1} \\ a_{1,1} \\ a_{2,1} \end{bmatrix}+x_2\begin{bmatrix} a_{0,2} \\ a_{1,2} \\ a_{2,2} \end{bmatrix} \quad (10)$$

식(10)의 벡터 스케일링 연산 Y_j를 구하면 식(11)과 같다.

$$Y_0=x_0\begin{bmatrix} a_{0,0} \\ a_{1,0} \\ a_{2,0} \end{bmatrix}=x_0\begin{bmatrix} 10 & 10 \\ 11 & 10 \\ 11 & 11 \end{bmatrix}\begin{bmatrix} 2^2 \\ 2^0 \end{bmatrix}=\begin{bmatrix} 2^2(10)x_0+2^0(10)x_0 \\ 2^2(11)x_0+2^0(10)x_0 \\ 2^2(11)x_0+2^0(11)x_0 \end{bmatrix} \quad (11a)$$
$$=\begin{bmatrix} 2^2\{(01)x_0\ll 1\}+2^0\{(01)x_0\ll 1\} \\ 2^2(11)x_0+2^0\{(01)x_0\ll 1\} \\ 2^2(11)x_0+2^0(11)x_0 \end{bmatrix}$$

$$Y_1=x_1\begin{bmatrix} a_{0,1} \\ a_{1,1} \\ a_{2,1} \end{bmatrix}=x_1\begin{bmatrix} 10 & 11 \\ 01 & 11 \\ 01 & 01 \end{bmatrix}\begin{bmatrix} 2^2 \\ 2^0 \end{bmatrix}=\begin{bmatrix} 2^2(10)x_1+2^0(11)x_1 \\ 2^2(01)x_1+2^0(11)x_1 \\ 2^2(01)x_1+2^0(01)x_1 \end{bmatrix} \quad (11b)$$
$$=\begin{bmatrix} 2^2\{(01)x_1<<1\}+2^0(11)x_1 \\ 2^2(01)x_1+2^0(11)x_1 \\ 2^2(01)x_1+2^0(01)x_1 \end{bmatrix}$$

$$Y_2=x_2\begin{bmatrix} a_{0,2} \\ a_{1,2} \\ a_{2,2} \end{bmatrix}=x_2\begin{bmatrix} 11 & 01 \\ 10 & 11 \\ 10 & 11 \end{bmatrix}\begin{bmatrix} 2^2 \\ 2^0 \end{bmatrix}=\begin{bmatrix} 2^2(11)x_2+2^0(01)x_2 \\ 2^2(10)x_2+2^0(11)x_2 \\ 2^2(10)x_2+2^0(11)x_2 \end{bmatrix} \quad (11c)$$
$$=\begin{bmatrix} 2^2(11)x_2+2^0(01)x_2 \\ 2^2\{(01)x_2\ll 1\}+2^0(11)x_2 \\ 2^2\{(01)x_2\ll 1\}+2^0(11)x_2 \end{bmatrix}$$

그리고 벡터 스케일링 연산을 이용한 내적연산은 식 (12)와 같다.

$$Y=\begin{bmatrix} 2^2\{(01)x_0\ll 1\}+2^0\{(01)x_0\ll 1\} \\ 2^2(11)x_0+2^0\{(01)x_0\ll 1\} \\ 2^2(11)x_0+2^0(11)x_0 \end{bmatrix}+\begin{bmatrix} 2^2\{(01)x_1<<1\}+2^0(11)x_1 \\ 2^2(01)x_1+2^0(11)x_0 \\ 2^2(01)x_0+2^0(01)x_0 \end{bmatrix}+\begin{bmatrix} 2^2(11)x_2+2^0(01)x_2 \\ 2^2\{(01)x_2\ll 1\}+2^0(11)x_2 \\ 2^2\{(01)x_2\ll 1\}+2^0(11)x_2 \end{bmatrix} \quad (12)$$

3. 연산공유 승산기를 이용한 내적 연산기 구조

3.1 연산공유 승산기 구조

연산공유 승산기는 크게 전처리기와 선택 및 가산부(Select& Adder unit)의 2블록으로 구성된다. 그림 1은 변수 x_j와 8비트의 계수 $a_{i,j}$를 승산하는 연산공유 승산기의 구조를 나타낸 것으로 각 블록의 동작은 다음과 같다.

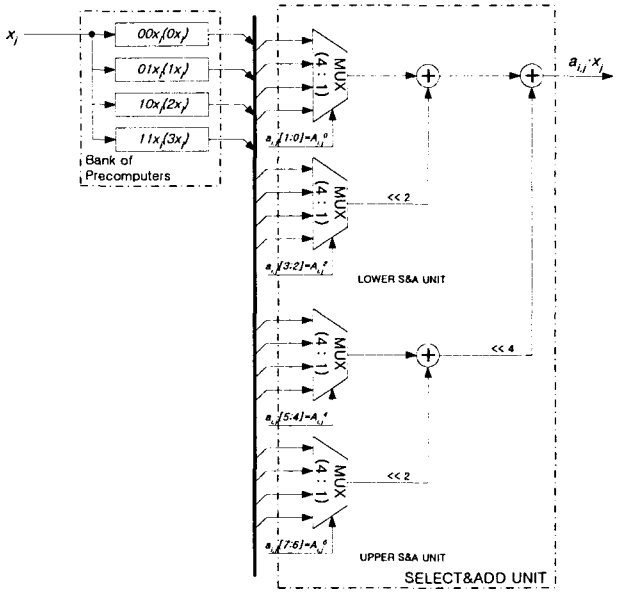

그림 1 변수와 8비트 계수의 승산을 위한 연산공유 승산기
Fig. 1 A CSHM for the multiplication of a variable and an 8-bit coefficient

① 전처리기 : 변수 x_j와 2비트 계수 $A^{2k}_{i,j}$에 대한 승산을 수행하여 그 값을 선택 및 가산부와 공유한다. 그림 2는 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.

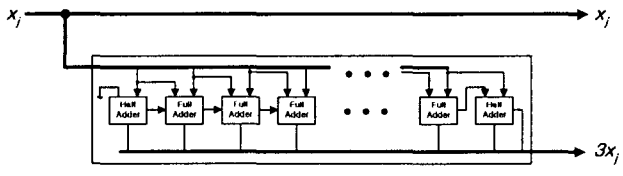

그림 2 전처리기 구조
Fig. 2 Architecture of the precomputer

② 선택 및 가산부 : 전처리기의 승산결과를 2비트씩 나누어 선택한 후 시프트 및 가산을 통해 최종 승산결과를 출력한다. 선택 및 가산부는 2개의 4×1 멀티플렉서와 1개의 가산기로 구성된다. 표 1은 4×1 멀티플렉서의 동작을 나타낸 것으로 8비트의 계수는 2비트씩 나뉘어져 멀티플렉서의 선택신호로 사용된다.

표 1 선택신호에 따른 4x1 멀티플렉서 출력
Table 1 4x1 multiplexer output for the select signal

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

3.2 벡터 스케일러 및 내적 연산기 구조

벡터 스케일러(vector scaler)는 하나의 변수 x_j와 L개의 계수 ($a=[a_{0,j}, a_{1,j}, ..., a_{L-1,j}]$)에 대한 승산을 위한 회로이다. 벡터 스케일러는 하나의 전처리기를 L개의 선택 및 가산부가 공유하는 구조를 갖는다. 그림 3은 식(2)를 수행하기위한 벡터 스케일러 구조를 나타낸다.

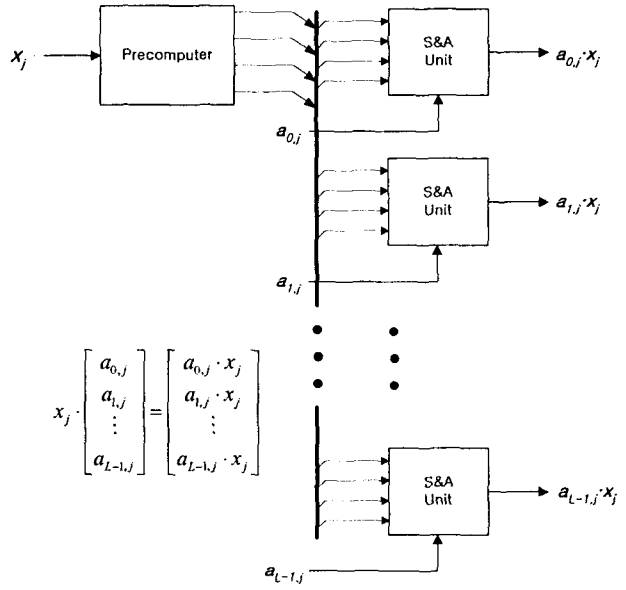

그림 3 벡터 스케일러 구조
Fig. 3 Architecture of a vector scaler

내적연산은 식(3)에 나타낸 바와 같이 벡터 스케일링 연산 값을 가산하는 형태로 이루어져 있으므로 벡터 스케일러와 누산기를 이용하여 구현할 수 있다. 매 클럭마다 Y_j 벡터 스케일링 연산 값이 누산기에 누적되므로 일반적인 $L\times L$ 행렬의 내적에 걸리는 클럭 수는 L이 된다. 그림 4는 식(3)을 수행하기위한 내적 연산기 구조를 나타낸다.

4. 연산공유 승산기를 이용한 DCT 프로세서 구조

4.1 DCT 알고리즘

$N\times N$ 화소 블록 $x(i, j)$에 대한 2차원 DCT변환 계수 $z(u, v)$는 다음과 같다.

$$z(u,v)=\frac{2}{N}C(u)C(v)\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}x(i,j)\cos\left[\frac{(2i+1)u\pi}{2N}\right]\cos\left[\frac{(2j+1)v\pi}{2N}\right] \quad (13)$$

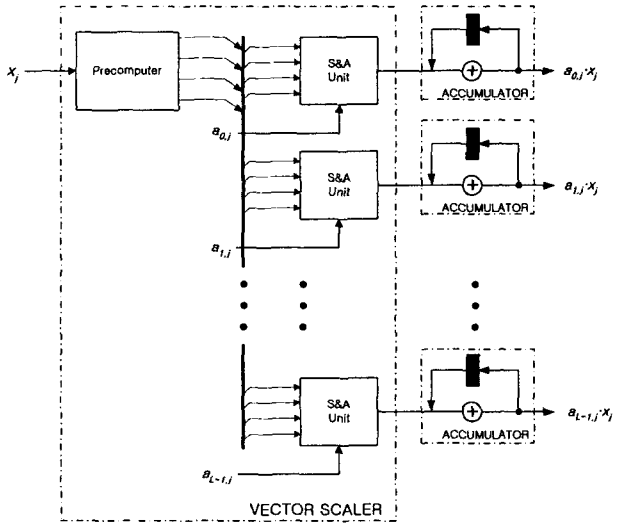

그림 4 내적 연산기 구조
Fig. 4 Architecture of an inner product

(단, $C(k)=\begin{cases}\frac{1}{\sqrt{2}} & k=0\\ 1 & otherwise\end{cases}$이다.)

식 (13)을 행-렬 분해(row-column decomposition)에 의한 행렬형태로 표현하면 식(14)와 같다.

$$z=CxC^T=C[Cx^T]^T \quad (14)$$

여기서 $y=Cx^T$라 하면 $z=C[Cx^T]^T=Cy^T$가 된다. 즉, 입력 x에 대하여 1차원 DCT 수행 후, 행렬의 전치 그리고 1차원 DCT의 과정을 거침으로써 2차원 DCT의 결과를 얻을 수 있다. 식(14)에서 행렬 C의 기저벡터 원소 C_{ij}는 다음과 같이 표현된다.

$$C_{ij}=\sqrt{\frac{2}{N}}A(i)\cos\frac{(2j+1)\pi i}{2N}$$
$$A(w)=\begin{cases}\frac{1}{\sqrt{2}} & w=0\\ 1 & otherwise\end{cases} \quad (15)$$

N=8에 대한 1차원 DCT를 행렬식으로 나타내면 식(16)과 같다.

$$\begin{bmatrix}y_0\\y_1\\y_2\\y_3\\y_4\\y_5\\y_6\\y_7\end{bmatrix}=\frac{1}{2}\begin{bmatrix}C_4 & C_4 & C_4 & C_4 & C_4 & C_4 & C_4 & C_4\\ C_1 & C_3 & C_5 & C_7 & -C_7 & -C_5 & -C_3 & -C_1\\ C_2 & C_6 & -C_6 & -C_2 & -C_2 & -C_6 & C_6 & C_2\\ C_3 & -C_7 & -C_1 & -C_5 & C_5 & C_1 & C_7 & -C_3\\ C_4 & -C_4 & -C_4 & C_4 & C_4 & -C_4 & -C_4 & C_4\\ C_5 & -C_1 & C_7 & C_3 & -C_3 & -C_7 & C_1 & -C_5\\ C_6 & -C_2 & C_2 & -C_6 & -C_6 & C_2 & -C_2 & C_6\\ C_7 & -C_5 & C_3 & -C_1 & C_1 & -C_3 & C_5 & -C_7\end{bmatrix}\begin{bmatrix}x_0\\x_1\\x_2\\x_3\\x_4\\x_5\\x_6\\x_7\end{bmatrix} \quad (16)$$

여기서, x_i는 입력화소 값, y_i는 DCT 계수 값, $C_k = \cos \pi k/16$은 코사인 계수 값을 나타낸다.

위 식을 두 개의 4×4 행렬 분해하면 식(17)과 같다.

$$\begin{bmatrix} y_0 \\ y_2 \\ y_4 \\ y_6 \end{bmatrix} = \begin{bmatrix} d & d & d & d \\ b & f & -f & -b \\ d & -d & -d & d \\ f & -b & b & -f \end{bmatrix} \cdot \begin{bmatrix} x_0+x_7 \\ x_1+x_6 \\ x_2+x_5 \\ x_3+x_4 \end{bmatrix}$$
$$\begin{bmatrix} y_1 \\ y_3 \\ y_5 \\ y_7 \end{bmatrix} = \begin{bmatrix} a & c & e & g \\ c & -g & -a & -e \\ e & -a & g & c \\ g & -e & c & -a \end{bmatrix} \cdot \begin{bmatrix} x_0-x_7 \\ x_1-x_6 \\ x_2-x_5 \\ x_3-x_4 \end{bmatrix} \quad (17)$$

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

식(17)에 나타낸 바와 같이 DCT는 대부분 내적연산으로 이루어지며 이를 하드웨어로 어떻게 설계하느냐가 DCT 프로세서의 성능을 좌우한다. 식(17)을 연산공유 승산기에 적용하기 위해 벡터 스케일링 연산의 가산 형태를 가진 식(18)로 변형한다.

$$\begin{bmatrix} y_0 \\ y_2 \\ y_4 \\ y_6 \end{bmatrix} = (x_0+x_7)\begin{bmatrix} d \\ b \\ d \\ f \end{bmatrix} + (x_1+x_6)\begin{bmatrix} d \\ f \\ -d \\ -b \end{bmatrix} + (x_2+x_5)\begin{bmatrix} d \\ -f \\ -d \\ b \end{bmatrix} + (x_3+x_4)\begin{bmatrix} d \\ -b \\ d \\ -f \end{bmatrix} \quad (18a)$$

$$\begin{bmatrix} y_1 \\ y_3 \\ y_5 \\ y_7 \end{bmatrix} = (x_0-x_7)\begin{bmatrix} a \\ c \\ e \\ g \end{bmatrix} + (x_1-x_6)\begin{bmatrix} c \\ -g \\ -a \\ -e \end{bmatrix} + (x_2-x_5)\begin{bmatrix} e \\ -a \\ g \\ c \end{bmatrix} + (x_3-x_4)\begin{bmatrix} g \\ -e \\ c \\ -a \end{bmatrix} \quad (18b)$$

변형된 식의 $x_0 \pm x_7$, $x_1 \pm x_6$, $x_2 \pm x_5$, $x_3 \pm x_4$는 변수에 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 계수벡터에 해당된다. 1차원 DCT의 내적연산에 사용되는 계수는 $a \sim f$의 7개이다. 표 2는 12비트의 1차원 DCT 계수와 알파벳을 나타낸 것이다.

표 2 12비트 1차원 DCT 계수 및 알파벳

Table 2 12-bit DCT coefficients and alphabets

Coeff	Value	Binary numbers	*alphabet · x*
a	0.4904	00 11 11 10 11 00	*x, 3x*
b	0.4619	00 11 10 11 00 10	*x, 3x*
c	0.4157	00 11 01 01 00 11	*x, 3x*
d	0.3536	00 10 11 01 01 00	*x, 3x*
e	0.2778	00 10 00 11 10 00	*x, 3x*
f	0.1913	00 01 10 00 01 11	*x, 3x*
g	0.0975	00 00 11 00 01 11	*x, 3x*

4.2 1차원 DCT 프로세서 구조

그림 5는 벡터 스케일러를 이용한 1차원 DCT 프로세서 구조를 나타낸다. 1차원 DCT 프로세서는 가감산기, 멀티플렉서, 벡터 스케일러, 누산기, 그리고 제어기로 구성된다. 가감산기는 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 계수는 벡터 스케일러에 의해 승산된다. 벡터 스케일링 연산 값은 누적기에 저장되며 가산승산 또는 감산승산에 대해 각각 4번의 누산을 수행하면 8입력에 대한 1차원 DCT 결과가 완성된다.

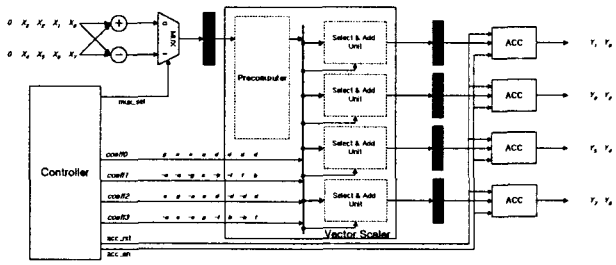

그림 5 벡터 스케일러를 이용한 1차원 DCT 프로세서 구조

Fig. 5 Architecture of the 1-D DCT processor using a vector scaler

그림 6은 1차원 DCT 프로세서 타이밍 다이어그램을 나타낸다.

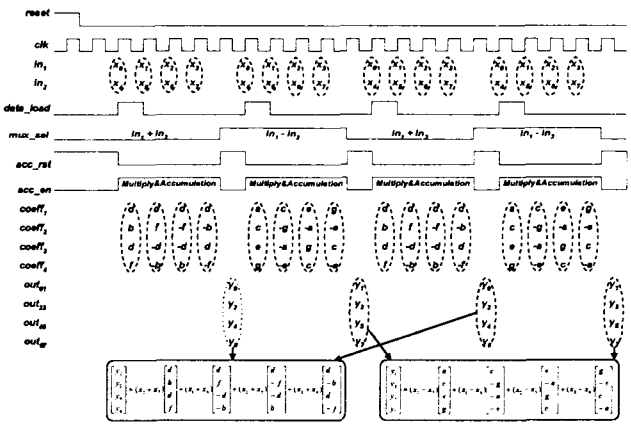

그림 6 1차원 DCT 프로세서 타이밍 다이어그램

Fig. 6 Timing diagram of the 1-D DCT processor

5. 성능비교 및 시뮬레이션

연산공유 승산기를 이용한 1차원 DCT 프로세서는 VHDL의 기능적인 기술(behavioral modeling) 기법과 구조적인 기술(structural modeling)기법을 사용하여 설계하였으며, 하드웨어 합성 및 시뮬레이션은 Altera의 Max+plusII를 사용하였다. 표 3은 1차원 DCT 프로세서를 설계하는데 필요한 하드웨어를 기존의 방식들과 비교한 것이다. 표 3에 나타난 바와 같이 고속 DCT 연산 알고리즘[6]-[8]은 다수의 승산기를 사

용하여 하드웨어 복잡도가 매우 크다. 반면, ROM-기반 분산 산술과 연산공유 승산기는 DCT연산을 승산기 없이 수행할 수 있고, 파이프라인 설계가 용이하기 때문에 훨씬 효율적인 하드웨어 설계가 가능하다.

표 3 1차원 DCT 프로세서 구현에 필요한 하드웨어 비교 (CSHM : Computation SHaring Multiplier)

Table 3 Comparison of required hardware to implement the 1-D DCT processor (CSHM : Computation SHaring Multiplier)

	Multiplier	Adder& Subtracter	ROM	Accum -ulator	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 4는 Park등과 본 논문의 연산공유 승산기를 사용하여 벡터 스케일러를 구현할 때 사용되는 하드웨어를 비교한 것이다. 표 4에 나타낸 바와 같이 Park등의 승산기는 전처리기 및 선택기의 하드웨어 복잡도가 크다. 반면, 본 논문의 연산공유 승산기는 전처기의 가감산기 수를 줄이고, 선택기의 Shifter와 Ishifter를 제거함으로써 기존에 비해 간단한 하드웨어 구조를 가진다.

표 4 벡터 스케일러 구현시 필요한 하드웨어 비교

Table 4 Comparison of required hardware to implement the vector scaler

		Park's CSHM	Proposed CSHM
Precomputer	Adder	4	1
	Subtracter	1	-
Select unit	8x1 MUX	12	-
	4x1 MUX	-	24
	Shifter/Ishifter	12/12	-
	Adder	-	12
Adder		8	8

표 5는 표 4에 나타낸 우수성을 증명하기 위한 것으로, Park등과 본 논문의 연산공유 승산기를 계수 개수 및 비트 수에 따라 벡터 스케일러로 구현한 후 Altera의 Max+plusII를 통한 논리합성을 수행하여 로직 셀 사용도를 비교하였다. 표 5에 나타낸 바와 같이 본 논문의 연산공유 승산기는 Park등의 승산기 보다 평균 55.6%의 로직 셀을 절약함을 확인할 수 있다. 따라서 본 논문의 연산공유 승산기가 기존의 것보다 내적연산의 하드웨어 구현에 훨씬 유리함을 알 수 있다.

표 6은 ROM-기반 분산산술, Park등의 연산공유 승산기, 그리고 본 논문의 연산공유 승산기를 이용하여 1차원 DCT 프로세서를 설계한 후 논리합성을 통해 나타난 로직 셀과 플립플롭 사용도 그리고 동작주파수를 비교한 것이다. 표 3에서 설명한 바와 같이 ROM-기반 분산산술이 기존의 방식 보다 우수하다고 가정한다면 연산공유 승산기를 이용한 DCT 프로세서는 ROM-기반 분산산술보다 로직 셀과 플립플롭 사용도 그리고 동작주파수 모두 우수함을 나타내었다. 연산공유 승산기 비교시, 제안된 승산기는 Park등의 승산기에 비해 적은 수의 로직 셀로 DCT 구현이 가능하고 동작주파수 또한 향상되었다. 따라서 제안된 승산기는 내적연산이 많은 DCT구현에 가장 적합하고 우수한 성능을 나타냄을 알 수 있다.

표 5 벡터 스케일러 구현에 사용되는 로직 셀 비교

Table 5 Comparison of logic cells to implement the vector scalers

Coefficient number	Bit	Park's vector scaler	Proposed vector scaler	Logic cells Reduction(%)
2	4	414	140	66.18
	8	808	324	59.90
	12	1164	504	56.70
	16	1513	695	54.06
	20	1895	882	53.46
	24	2238	1074	52.01
	28	2753	1274	53.72
	32	3197	1474	53.89
4	4	743	272	63.39
	8	1489	638	57.15
	12	2161	998	53.82
	16	3121	1381	55.75
	20	3781	1720	54.51
	24	4547	2092	53.99
	28	5352	2480	53.66
	32	6182	2876	53.48
6	4	1057	404	61.78
	8	2123	952	55.16
	12	3400	1458	57.12
	16	4464	2010	54.97
	20	5615	2574	54.16
	24	6758	3132	53.65
	28	7929	3714	53.16
	32	9213	4308	53.24
8	4	1371	536	60.90
	8	2976	1266	57.46
	12	4496	1940	56.85
	16	5937	2676	54.93
	20	7425	3428	53.83
	24	8974	4172	53.51
	28	10543	4948	53.07
	32	12167	5740	52.82
10	4	1685	668	60.36
	8	3692	1552	57.96
	12	5672	2422	57.30
	16	7363	3342	54.61
	20	9253	4282	53.72
	24	11200	5212	53.46
	28	13151	6182	52.99
	32	15181	7172	52.76

표 6 3가지 1차원 DCT 프로세서 비교

Table 6 Comparison with three types of 1-D DCT processors

	ROM-based DA	Park's CSHM	Proposed CSHM
Input bit	9	9	9
Output bit	16	16	16
Logic cells required	1876	1594	1074
Flip-flops required	461	112	112
Clock period	47.9ns	47.7ns	41.1ns
Operating Freq.	20.87MHz	20.96MHz	24.33MHz

그림 7은 제안된 연산공유 승산기를 이용한 1차원 DCT 프로세서의 시뮬레이션 파형을 나타낸 것이다. 그림 7의 타이밍 다이어그램과 같이 초기입력이 들어간 후 4클럭 후 식(12a)가 완료되고 누산기 초기화 후 다시 4클럭 후 식(12b)가 완료된다. 따라서 초기입력 후 9클럭이 지나면 8개 입력에 대한 1차원 DCT 연산이 끝나게 된다.

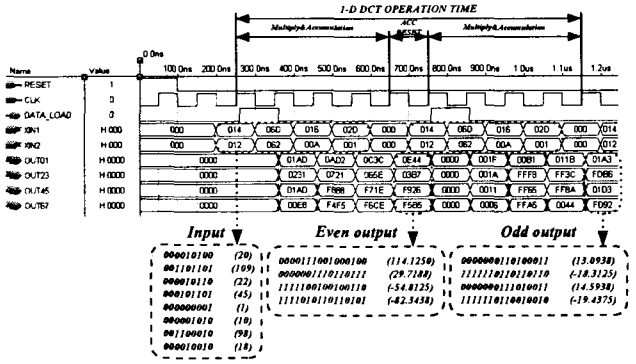

그림 7 제안된 연산공유 승산기를 이용한 1차원 DCT 프로세서 시뮬레이션 파형

Fig. 7 Simulation waveform of the 1-D DCT processor using the proposed CSHM

6. 결 론

본 논문에서는 내적연산을 효율적으로 처리할 수 있는 연산공유 승산 알고리즘과 하드웨어 구조를 제안하고 이를 1차원 DCT 프로세서에 적용하였다. 일반적인 내적연산은 행렬의 승산 형태를 가져 계산이 복잡하며 하드웨어 구현이 어렵다. Park등에 의해 제안된 연산공유 승산 알고리즘은 하드웨어 구현시 전처리기 및 선택부의 하드웨어 복잡도가 커 승산기의 성능이 저하되는 문제점을 가진다. 제안된 승산 알고리즘은 계수벡터를 2비트씩 처리함으로써 이러한 문제점을 해결하였다. 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 분산산술과 기존 방식인 Park등의 연산공유 승산기에 비해 제안된 승산기가 하드웨어 복잡도 및 동작주파수 모두 뛰어난 성능을 나타내었다. 제안된 알고리즘과 승산기는 DCT뿐 아니라 DSP 알고리즘에 많이 사용되는 내적 및 행렬의 승산 등에 응용 가능하다.

감사의 글

본 연구는 한국과학재단 지정 울산대학교 네트워크 기반 자동화연구센터의 지원(NARC)과 산업자원부 반도체 설계 교육센터(IDEC)의 지원으로 수행되었음.

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 architecture 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] J.S, Park, K. Muhammad and K. Roy, "High-Performance FIR Filter Design Based on Sharing Multiplication", IEEE Trans. on VLSI, Vol. 11, pp. 244-253300, Apr. 2003.

[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.

저 자 소 개

이 태 욱 (李 泰 旭)

1973년 5월 28일생. 1998년 울산대학교 전자공학과 졸업. 2000년 동 대학원 전자공학과 졸업(공학석사). 2000~현재 동 대학원 전기전자정보시스템 공학부 박사과정

Tel : 052-259-1629, Fax : 052-259-1686
E-mail : twlee00@korea.com

조 상 복 (趙 相 福)

1955년 6월 10일생. 1979년 한양대학교 전자공학과 졸업. 1981년 동 대학원 전자공학과 졸업(공학석사). 1985년 동 대학원 전자공학과 졸업(공학박사). 1986년~현재 울산대학교 전기전자정보시스템 공학부 교수

Tel : 052-259-2202, Fax : 052-259-1686
E-mail : sbcho@mail.ulsan.ac.kr