論文2001-38SD-7-7

내적연산을 위한 가산기 공유항의 최적 추출기법 제안 및 이를 이용한 DCT 설계

(The Optimal Extraction Method of Adder Sharing Component for Inner Product and its Application to DCT Design)

林 國 贊 *, 張 榮 眞 *, 李 顯 洙 *

(Guk-Chan Lim, Young-Jin Jang, and Hyon-Soo Lee)

요 약

직교변환이나 필터처리를 위한 대부분의 DSP알고리즘에서는 내적을 효율적으로 처리할 수 있는 하드웨어 구조가 필수적이다. 내적을 계산하기 위한, 전통적인 MAC구조는 실리콘 면적의 비용이 높기 때문에 승산기가 없는 분산연산구조가 널리 사용된다. 본 논문은 분산연산구조에서 가산기 공유항을 최대로 추출하여 구현에 필요한 하드웨어의 요소를 최소화하기 위한 방법으로 신경망의 최적화 알고리즘을 이용하는 방법을 제안한다. 제안한 방법은 내적의 깊이에 따라 복잡해지는 가산기 공유항 추출 과정을 최적화함으로써 단시간에 최소의 FA와 FF를 이용한 최적의 가산-네트워크 구성이 가능하다. 또한, 제안한 방법을 적용한 DCT 설계에서는 기존의 ROM-기반 분산연산 보다도 효율적인 구성이 가능하다.

Abstract

The general DSP algorithm, like orthogonal transform or filter processing, needs efficient hardware architecture to compute inner product. The typical MAC architecture has high cost of silicon. Because of this reason, the distributed arithmetic without multiplier is widely used for implementing inner product. This paper presents the optimization to reduce required hardware in distributed arithmetic by using extraction method of adder sharing component. The optimization process uses Boltzmann-machine which is one of the neural network. This proposed method can solve problem that is increasing complexity depending on depth of inner product and compose optimal summation-network with the minimum FA and FF in a few time. The designed DCT by using proposed method is more efficient than a ROM-based distributed arithmetic.

I. 서 론

휴대용 멀티미디어 디바이스에 사용되는 DSP (Digital Signal Processing) 알고리즘들은 계산상의 복잡성 때문에 효율적인 하드웨어 구조가 필수적이다. 특히, DCT(Discrete Cosine Transform)나 DFT (Discrete Fourier Transform)와 같은 직교변환이나 FIR(Finite-length Impulse Response), 필터(filter) 처리에서는 내적연산(inner product)을 어떠한 구조로 구현하느냐에 따라서 전체 시스템의 성능 및 하드웨어 크기가 좌우된다. 분산연산(distributed arithmetic)은 이러한 내적을 계산하는데 승산기나 MAC(Multiply Accumulator)를 사용하는 것보다 소비전력 및 크기를 효율적으로 줄일 수 있고, 고속동작이 가능한 회로 구현이 쉽기 때문에 소규모 신호처리 시스템 설계에 많이 이용되고 있다.[1]

* 正會員, 慶熙大學校 電子計算工學科
(Dept. of Computer Engineering Kyunghee Univ)
接受日字:2000年12月26日, 수정완료일:2001年6月13日

본 논문에서는 기존의 ROM-기반 분산연산구조가 내적의 깊이에 따라 지수적으로 증가하는 고정된 LUT (Look-Up Table) 사용으로 인한 하드웨어의 비효율성을 해결하기 위해서, 크기 및 동작속도 측면에서 우수한 성능을 가지는 가산기-기반 분산연산[2]의 최적화 설계방법에 대해서 다룬다. 이를 위해 기존의 ROM을 가산-네트워크(Summation-Network)로 대체하고, 고정된 계수데이터를 비트단위로 확장하여 비트 패턴에 따른 가산기 공유항을 최대한 추출, 가산-네트워크 구성에 필요한 FA(Full Adder) 및 FF(Flip-Flop)수를 최소화하였다. 이러한 최소화를 위한 최적화 과정은 신경망 알고리즘의 하나인 볼츠만-머신(Boltzmann-Machine)[3]을 사용했으며, 최종 가산-네트워크를 구성하기 위한 일련의 설계방법론에 대해서도 기술한다. 또한, 기존의 ROM-기반 분산연산 구조와 MCMT(Multiple Constant Multiplier Tree) 구조[4]를 비교하여 문제점해결을 위한 방향을 제시하였고, 다양한 DSP 알고리즘에 제안한 설계방법이 적용될 수 있도록 일반화하였다. DCT 모듈설계에서는 제안한 방법을 이용한 경우가 소요되는 하드웨어 및 동작속도 측면에서 기존의 방법보다 효율적인 결과를 얻었다.

Ⅱ. 내적연산을 위한 기존의 구조

신호처리 분야에서 가장 일반적인 알고리즘은 내적연산으로 식 (1)과 같이 표현된다.

$$z=\sum_{n=0}^{N-1} C_n \cdot x_n \tag{1}$$

여기서 C_n은 고정된 계수이고, x_n은 입력, z는 출력, N은 내적의 깊이를 나타낸다.

이러한 내적을 계산하는데 MAC을 이용하는 경우에는 내적의 깊이(N)만큼의 MAC 유닛이 필요하다. 식 (1)의 입력데이터를 비트단위로 확장하기 위하여 $x_n=\sum_{l=0}^{L-1} x_{n,l}2^{-l}$을 대입하면 식 (2)와 같다.

$$z=\sum_{n=0}^{N-1}\left(\sum_{l=0}^{L-1}(C_n \cdot x_{n,l})2^{-l}\right) \tag{2}$$

식 (2)에 대한 하드웨어 구조는 그림 1과 같다.

그림 1의 MAC 구조는 다중 MAC 유닛을 사용하기 때문에 하드웨어 오버헤드가 크다. 분산연산은 식 (2)의 연산순서를 식 (3)처럼 재정렬한 것으로, 그림 2처럼 MAC 유닛의 Shift-Adder를 하나만 사용하는 구조 설계가 가능하다.

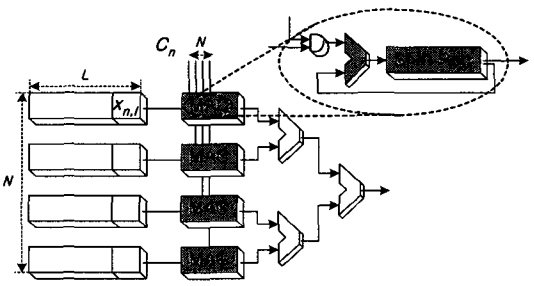


그림 1. 내적연산을 위한 MAC 구조 (N=4).
Fig. 1. MAC architecture for computing inner product (N=4).

$$z=\sum_{l=0}^{L-1}\left(\sum_{n=0}^{N-1}(C_n \cdot x_{n,l})\right)2^{-l} \tag{3}$$

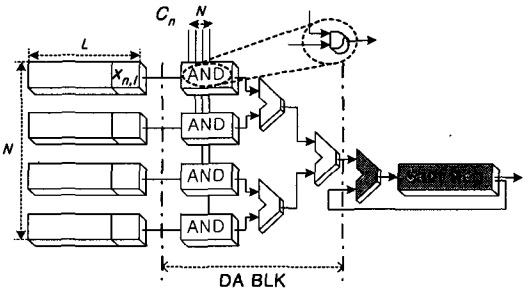


그림 2. 내적연산을 위한 분산연산 구조 (N=4).
Fig. 2. Distributed arithmetic architecture for computing inner product (N=4).

대부분의 신호처리분야에서 사용하는 알고리즘들은 계수데이터가 상수값이다. 따라서, 출력데이터는 입력데이터에만 의존한다. ROM-기반 분산연산은 'DA BLK(Distributed Arithmetic Block)'의 연산과정을 미리 계산하여 ROM에 저장하는 방법으로, ROM을 사용하기 때문에 계수데이터의 유연성 및 입력 비트단위의 파이프라인 구조 설계가 가능하다. 따라서, FPGA를 이용한 DSP 칩 구현에 효율적이다.[5] 그러나, 구현에 필요한 ROM의 크기는 내적의 깊이에 따라 지수적으로 증가한다는 문제점을 가진다.

내적을 계산하기 위한 또 다른 접근 방법으로 MCMT(Mutiple Constant Multiplier Trees)를 이용한

방법이 있다. 이 방법은 계수가 0인 부분의 연산결과는 결과값에 영향을 주지 않기 때문에 필요한 가산 연산을 줄일 수 있다. 또한 계수 비트의 패턴이 같은 곳의 가산연산을 공유하도록 설계하기 때문에 소요되는 하드웨어 측면에서 보다 효율적인 구현이 가능하다. 아래 식 (4)는, 식 (2)의 계수데이터를 비트단위로 확장하기 위해 $C_n=\sum_{m=0}^{M-1}C_{n,m}2^{-m}$을 대입한 수식이다.

$$z=\sum_{m=0}^{M-1}(\sum_{n=0}^{N-1}(C_{n,m}\cdot x_n))2^{-m} \tag{4}$$

식 (4)에서, 계수데이터가 각각 10, 12, 15, 7인 경우를 고려할 경우, 식 (5)와 같은 행렬식으로 표현할 수 있다.

$$z=\begin{pmatrix}z_0\\z_1\\z_2\\z_3\end{pmatrix}=\begin{pmatrix}0&0&1&1\\1&0&1&1\\0&1&1&1\\1&1&1&0\end{pmatrix}\begin{pmatrix}x_0\\x_1\\x_2\\x_3\end{pmatrix} \tag{5}$$

여기서, $z=z_0+(z_1+(z_2+(z_3)\cdot 2)\cdot 2)\cdot 2$이다.

예를 들어, 위의 식 (5)에서, z_1에 대한 계산식을 서브 트리로 표현하면 그림 3과 같다.

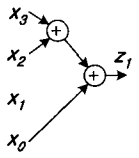


그림 3. z_1을 계산하기 위한 서브 트리
Fig. 3. Sub tree for computing z_1.

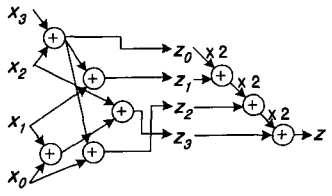


그림 4. 내적연산을 위한 MCMT 구조의 예 (N=4).
Fig. 4. The example of MCMT architecture for computing inner product (N=4)

z_0, z_1, z_2, z_3의 모든 서브 트리를 각각 구성하여 하나로 합하게 되면, 그림 4처럼 계수의 비트 패턴에 따라 가산항을 공유할 수 있어, 계산에 필요한 가산기를 줄일 수 있다. 그러나, MCMT는 입력데이터가 워드 단위이므로 가산기의 처리시간 지연문제와 가산기 공유항을 추출하기가 복잡하다는 문제점을 가지고 있다.

Ⅲ. 가산기-기반 분산연산의 구조

가산기-기반 분산연산은 입력데이터와 계수데이터 모두를 비트단위로 확장함으로써, ROM-기반 분산연산의 특징인 비트단위 파이프라인 구조를 적용하여 고속동작이 가능하고, MCMT의 특징인 계수에 따른 가산항 소거 및 가산기 공유특성을 최대한 이용하여 필요한 하드웨어를 줄일 수 있는 설계가 가능하다. 식 (2)에서 계수와 입력데이터를 각각 비트단위로 확장하면 식 (6)과 같이 표현할 수 있다.

$$z=\sum_{l=0}^{L-1}(\sum_{m=0}^{M-1}(\sum_{n=0}^{N-1}(C_{n,m}\cdot x_{n,l}))2^{-m})2^{-l} \tag{6}$$

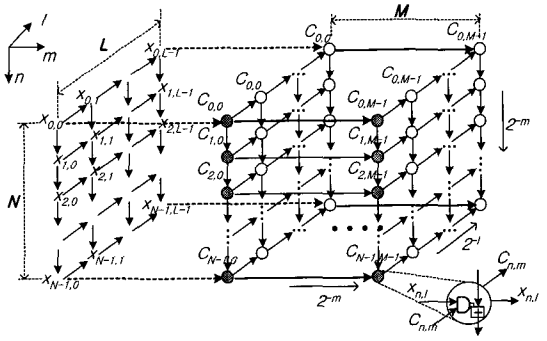


그림 5. 가산기-기반 분산연산의 DG
Fig. 5. DG of Adder-based Distributed arithmetic.

식 (6)의 연산관계를 비트단위 DG (Dependency Graph)로 표현하면 그림 5와 같다. 식 (6)에서 3개의 인덱스(l, m, n)에 의해 DG가 육면체형태로 확장된다. 각 노드는 비트단위 승산이므로 하나의 AND와, 결과값을 더하기 위한 가산기로 구성된다. DG를 l-방향으로 프로젝션(projection)시키면 그림 6과 같이 표현할 수 있다. 프로젝션 결과 소거된 l-방향의 인덱스 정보는, 그림 2와 같이 입력데이터를 비트단위로 입력하고 하나의 shift-adder를 이용하여 계산할 수 있다. 또한, 그림 2에서 'DA BLK'을 구현하는데, MCMT의 특징 즉, 계수비트가 0인 부분의 가산 항을 줄인 서브 트리와 이 서브 트리들을 하나의 가산-네트워크에 포함시켜, 계수 비트의 패턴이 같은 곳의 가산을 공유하게 함으로써 구현에

필요한 가산기의 수를 줄일 수 있다.

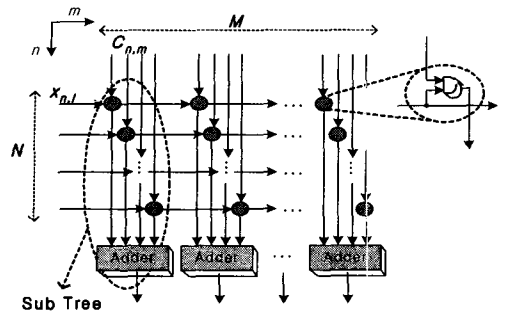


그림 6. 가산기-기반 분산연산 DG의 l-방향 프로젝션.
Fig. 6. Projection in *l-direction* of Adder-based distributed arithmetic DG.

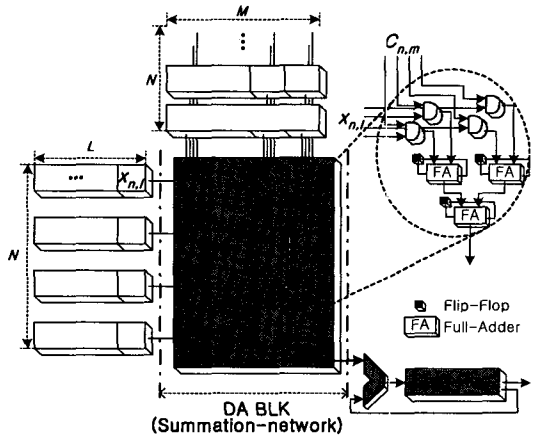


그림 7. 내적연산을 위한 가산기-기반 분산연산 구조 (N=4).
Fig. 7. Adder-based Distributed arithmetic architecture for computing inner product (N=4).

그림 7은 내적연산을 위한 가산기-기반 분산연산의 구조이다. 여기서 계수 비트 패턴에 따라 최대로 가산기 공유항을 추출하여 최소의 FA와 FF로 가산-네트워크를 구성해야 한다.

Ⅳ. 신경망을 이용한 최적화 설계

최적의 가산-네트워크를 설계함에 있어서 기존에는 설계자가 직접 반복적인 작업을 수행하여 최적의 공유항 정보를 추출하였다. 본 장에서는 신경망의 볼츠만-머신을 이용한 방법에 대해서 기술한다.

1. 최적의 가산-네트워크를 위한 설계조건

그림 8의 예를 통하여, 계수 비트의 패턴에 따른 가산기 공유항의 설정관계를 살펴보자.

그림 8은 N=4, M=8일 때의 계수 비트 예와 이때, 공유할 수 있는 공유묶음 관계를 그래프로 나타내었다. 여기서 그래프는 이론 1의 조건을 모두 만족해야만, 가산-네트워크 구성에 필요한 FF와 FA의 수를 최소화 할 수 있다.

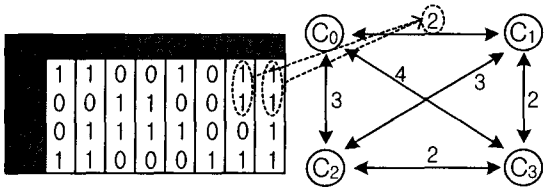
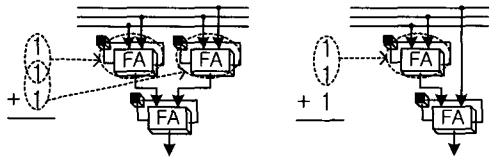


그림 8. 계수 비트 예와 계수의 가산기 공유항 관계를 나타내는 그래프
Fig. 8. The Example of coefficient data bit and graph to express relationship of adder sharing component in coefficient data.

이론 1. 최적의 가산-네트워크를 위한 '설계조건'
Theory 1. The 'design constraint' to compose optimal summation-network.

1. 각 단계에서 가산기 공유항에 속한 노드는 같은 단계의 다른 공유항에 포함될 수 없다.
2. 각 단계에서 가산기 공유항에 속한 노드의 엣지 합은 최대가 되어야한다.
3. 조건 1, 2를 만족하는 경우가 둘 이상일 때, 엣지가 0인 두 노드가 선택되어야한다.



(a) '설계조건 1'을 만족하지 못한 경우 (b) '설계조건 1'을 만족한 경우

그림 9. '설계조건 1'에 따른 가산-네트워크 구조
Fig. 9. Summation-network depending on 'design constraint 1'.

그림 9의 (a)와 (b)는 각각 이론 1의 '설계조건 1'을 만족하지 못한 경우와 만족한 경우의 가산-네트워크이

다. '설계조건 1'을 만족하지 못하는 경우에는 (a)와 같이 계산에 필요 없는 FA와 FF가 생성될 수 있으므로 '설계조건 1'을 반드시 만족해야 한다.

그림 10은 그림 8의 예를 대상으로 '설계조건 2'의 만족여부에 따라 구성한 최종 가산-네트워크이다. (a)는 '설계조건 2'를 만족하지 않은 경우로 C_0-C_1와 C_2-C_3, C_0-C_3와 C_1-C_2, C_0-C_2 순서로 가산기 공유항을 선택하여 구성한 그림이고 (b)는 '설계조건 2'를 만족하도록, C_0-C_3와 C_1-C_2, C_0-C_2 순서로 노드를 선택하여 구성한 그림이다. 그림 10에서 알 수 있듯이 최종 구현되는 가산-네트워크의 FA와 FF수가 달라지기 때문에 '설계조건 2'를 만족해야한다.

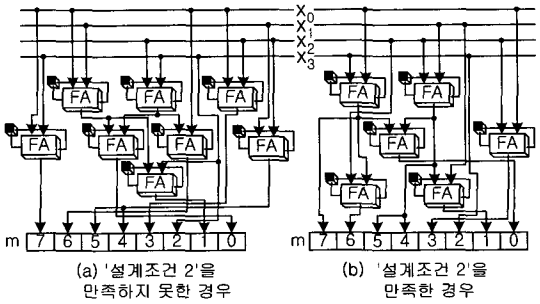


그림 10. '설계조건 2'에 따른 가산-네트워크 구조
Fig. 10. Summation-network depending on 'design constraint 2'.

'설계조건 3'은 '설계조건 1'과 '설계조건 2'가 모두 만족한 경우의 후행 조건으로 엣지가 0인 곳은 실제 가산기가 필요하지 않기 때문에, 이를 만족하면 보다 효율적인 가산-네트워크 구성이 가능하게 된다.

2. 문제의 복잡도

그래프로 표현된 정보를 기반으로 발생할 수 있는 경우들 중에서, 이론 1의 '설계조건'을 모두 만족한 경우를 찾기 위한 복잡도를 고려해 보자. 노드 수(N)에 대하여 생성 가능한 엣지의 수(E)는 식 (7)과 같다.

$$E=\frac{N(N-1)}{2} \tag{7}$$

또한, 각 단계별로 '설계조건 1'을 만족하는 최대 가산기 공유묶음 수(S)는 $\lfloor \log_2 N \rfloor$ 이다. 결국, E개의 엣지 중에서 S개를 순서에 상관없이 선택할 수 있는 경우의 수를 SB라 하면, 이는 조합문제이므로 식 (8)과 같이 정의할 수 있다.

$$SB={}_E C_S=\frac{E!}{S!(E-S)!} \tag{8}$$

그림 11은 N, E, SB의 관계를 나타내었다. N이 커짐에 따라 SB는 지수적으로 증가하므로 이들 중 '설계조건'을 모두 만족하는 경우를 찾는 것은 매우 어렵다. 결국, 이는 NP-완전문제(Non-deterministic polynomial time complete problem)에 해당하며,[2] 효율적인 설계를 위해서 본 논문에서는 신경망의 볼츠만-머신을 이용하였다.

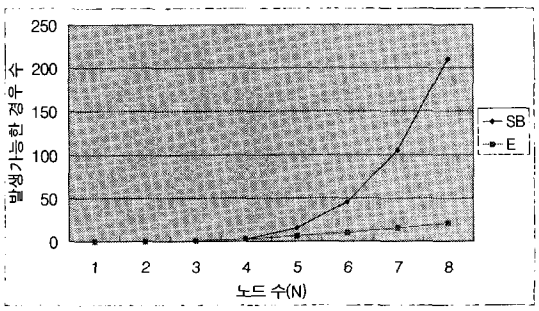


그림 11. N에 따른 E와 SB의 복잡도.
Fig. 11. Complexity of E and SB depending on N.

3. 볼츠만 머신을 이용한 가산기 공유항 추출

본 논문에서는 가산기 공유항 관계를 나타내기 위하여, 그림 12처럼 2차원 행렬식으로 표현하였다. 양방향 그래프이므로 행렬식은 대각선을 기준으로 대칭된다. 그림 12는 C_0와 C_2가 가산기 공유항으로 추출되었을 때의 행렬식 표현과 신경망의 출력상태이다.

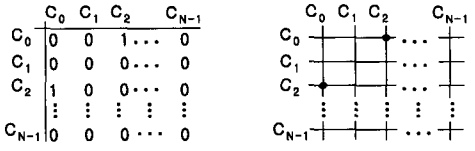


그림 12. 행렬식 및 신경망 상태의 예 (C_0-C_2).
Fig. 12. The example of matrix form and neural network status (C_0-C_2).

최적의 상태를 출력하기 위해서는 그림 12의 행렬 표현방식이 이론 2와 같은 5가지의 '동작조건'에 맞도록 동작해야 한다.

이론 2. 최적의 가산-네트워크를 위한 '동작조건'
Theory 2. The 'processing constraint' to compose optimal summation-network.

1. 한 행과 한 열에는 반드시 하나의 점만 활성화 되어야 한다.
2. 전체 활성화되는 점은 $2\lfloor \log_2 N \rfloor$ 개이다.
3. 대각선 점은 반드시 활성화 될 수 없다.
4. 활성되는 점에 해당하는 두 노드가 연결되는 엣지의 웨이트 합이 최대가 되어야 한다.
5. 위의 모든 '동작조건'이 만족하는 경우가 둘 이상일 때, 결합 엣지의 웨이트가 0을 포함한 부분이 활성화되어야 한다.

'동작조건 1'과 '동작조건 2'는 '설계조건 1'을 만족하기 위한 조건이고 '동작조건 3'은 자신끼리의 가산기 공유항이 설정될 수 없기 때문에 설정된다. 그리고, '동작조건 4'와 '동작조건 5'는 각각 '설계조건 2'와 '설계조건 3'을 만족하기 위해 설정된다.

각 '동작조건'에 대한 신경망의 에너지 수식은 식 (9)~(13)와 같다.

$$E_1=\frac{A}{2}\sum_x\sum_i\sum_y\sum_j V_{xi}V_{yj}\delta_{xy}(1-\delta_{ij}) + \frac{B}{2}\sum_x\sum_y\sum_i\sum_j V_{xi}V_{yj}\delta_{ij}(1-\delta_{xy}) \quad (9)$$

$$E_2=\frac{C}{2}\sum_x\sum_i(V_{xi}-2\log_2 N)^2 \quad (10)$$

$$E_3=\frac{D}{2}\sum_x\sum_i V_{xi}\delta_{xi} \quad (11)$$

$$E_4=-\frac{E}{2}\sum_x\sum_i\sum_y\sum_j(S(x,i)+S(y,j))\,\delta_{xj}\delta_{yi}(1-\delta_{xi}) \quad (12)$$

$$E_5=\frac{F}{2}\sum_x\sum_i\sum_y\sum_j S(x,i)\delta_{xy}\delta_{ij}(1-\delta_{xi}) \quad (13)$$

여기서 A, B, C, D, E, F는 에너지 상수이고, x, y는 행렬식의 행 노드, i, j는 열 노드를 나타내기 위한 인덱스이다. δ_{xy}는 $x=y$이면 1, $x\neq y$이면 0을 출력하고, $S(x,y)$는 x와 y를 연결하는 엣지의 웨이트를 나타낸다. 그리고 V_{xy}는 행렬 (x,y)의 출력값을 나타낸다.

행렬 표현방식에 대한 전체 에너지 수식은 식 (14)와 같이 주어진다.

$$E=-\frac{1}{2}\sum_{x,y,i,j}T_{xi,yj}V_{xi}V_{yj}-\sum_{x,i}I_{xi}V_{xi} \quad (14)$$

여기서 T와 I는 각각 신경망의 결합계수(connection weight)와 임계값(offset value)이다.

최적의 가산기 공유항 묶음관계를 설정하기 위한 총 에너지는 $E_1+E_2+E_3+E_4+E_5$이고, 이 값이 전체 에너지에 해당하므로, 식 (14)의 T와 I는 식 (15)와 식 (16)과 같이 나타낼 수 있다.

$$\begin{aligned}T_{xi,yj}=&-A\delta_{xy}(1-\delta_{ij})\\&-B\delta_{ij}(1-\delta_{xy})\\&-C\\&+E(S(x,i)+S(y,j))\delta_{xj}\delta_{yi}(1-\delta_{xi})\\&-FS(x,i)\delta_{xy}\delta_{ij}(1-\delta_{xi})\end{aligned} \quad (15)$$

$$I_{xi}=2C\log_2 N-\frac{D}{2}\delta_{xi} \quad (16)$$

식 (15)와 식 (16)에 의해 설정된 T와 I을 정리 1의 볼츠만-머신 동작 규칙에 적용하면, 볼츠만-머신은 최소 에너지 상태로 수렴하게 되고, 특정한 상태를 출력하게 된다. 출력된 정보는 가산-네트워크를 최적으로 구성할 수 있는 가산기 공유항 정보를 나타낸다.

정리 1에서 P는 확률, $u_i(t)$는 현재시간 (t)의 입력 총합, T는 볼츠만-머신의 온도를 나타낸다. T가 아주 작으면 P가 1또는 0의 극한값을 갖게 되어 특정 상태로 빠르게 수렴하지만, 지역 최소점에 빠질 확률이 높기 때문에 최적의 상태를 얻기가 어렵다. 반면에 T가 너무 크면 최소 에너지 상태에 도달하는데 시간이 너무 많이 걸리기 때문에 적절한 T의 변화가 필요하다. 이러한 과정을 Simulated Annealing이라 하는데, 본 논문에서는 S.Geman & D. Geman에 의해 제안된, 식 (17)과 같이 $T(t)$를 낮추어 가는 방법을 사용했다. 여기서 G는 임의의 정수이다.

$$T(t)\geq\frac{G}{\log(1+t)} \quad (17)$$

볼츠만-머신은 항상 최적의 상태를 찾는 것은 아니다. 그러나, 최소에너지 수렴률이 매우 높기 때문에 몇 번의 반복과정을 통해 쉽게 최소 에너지 상태를 찾을 수 있다. 또한, 에너지 값으로 현재상태와 이전상태의 최적화 정도를 분별할 수 있기 때문에 설계자가 쉽게 가산기 공유항 정보의 최적화 정도를 판별할 수 있다.

정리 1. 볼츠만-머신의 동작 규칙
Property 1. Processing rule of Boltzmann- machine.

Step 1. 그림 12의 (b)에서 랜덤으로 한 개의 유닛, i을 선택하여 $u_i(t)$을 구한다.

$$u_i(t)=\sum_{i\neq j}T_{i,j}V_j(t)+I_i$$

Step 2. 확률 P로 $V(i+1)$을 1로 설정한다.

$$P[V_i(t+1)=1]=f(\frac{u_i(t)}{T})$$

$$f(x)=\frac{1}{1+\exp(-x)}$$

Step 3. i이외의 유닛 출력은 변화시키지 않는다.

$$V_j(t+1)=V_j(t)$$

Step 4. 특정 상태에 수렴할 때까지 T을 변화시키며 Step 1로 돌아간다.

4. 가산-네트워크 구성

그림 13은 가산-네트워크를 구성하기 위한 제안한 방법의 흐름도이다.

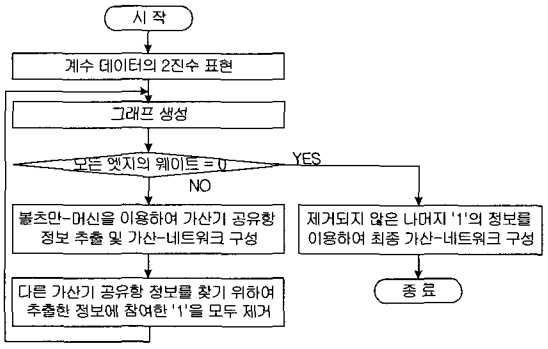


그림 13. 가산-네트워크 구성 흐름도
Fig. 13. Flow to compose summation-network.

제안한 최적화 설계 방법을 토대로 그림 8의 예를 가산-네트워크로 구성하는 과정이 그림 14에 나타나 있다.

그림 14에서의 (a)는 계수데이터의 비트 정보를 기반으로 구성한 그래프로부터 최적의 정보를 추출하여 가산-네트워크를 구성하는 과정이다. (b)에서는 (a)의 가산기 공유항에 참여한 1을 소거한 후 다시 생성한 그래프 정보를 기반으로 가산-네트워크를 구성하는 과정이다. (c)는 더 이상 엣지의 웨이트가 없어, 가산기 공유항에 참여하지 않은 나머지 1의 정보를 기반으로 가산-네트워크를 구성하는 과정이다. 최종 가산-네트워크 구성은 그림 10의 (b)와 같다.

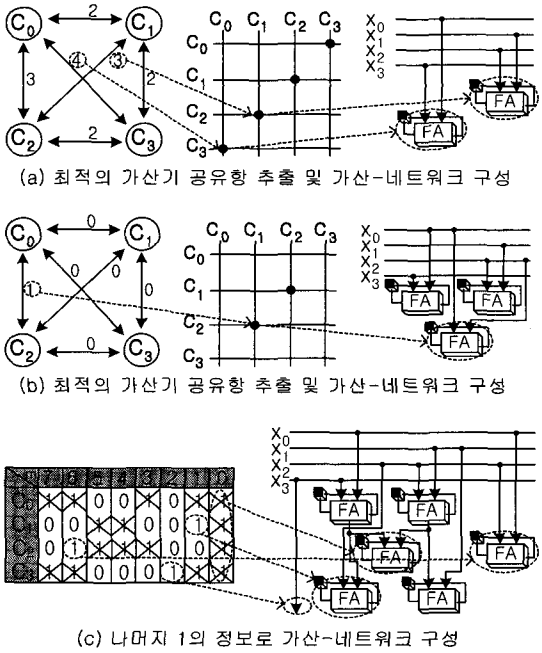


그림 14. 볼츠만-머신을 이용한 가산기 공유항 추출 및 가산-네트워크 구성
Fig. 14. The extraction of adder sharing component by using Boltzmann-machine and the organization of summation-network.

정확한 동작을 수행하기 위해서는 계수 비트에 포함된 모든 1의 정보가 가산-네트워크 구성에 참여해야한다. 본 논문에서 제안한 볼츠만-머신의 효율성은 가산-네트워크 구성에 필요한 FA와 FF의 효율성에만 관계된다. 따라서 신경망이 어떠한 상태로 동작하더라도 추출한 정보를 기반으로 구성한 가산-네트워크는 항상 정확한 동작을 보장받을 수 있다.

V. DCT설계 및 결과 비교

1. DCT 설계

입력데이터를 $\{x_n;n=0,1,2,\ldots,N-1\}$, DCT 변환 후의 데이터를 $\{z_k;k=0,1,2,\ldots,N-1\}$라 하면, DCT는 식 (18)과 같이 정의할 수 있다.

$$z_k=\frac{2\varepsilon_k}{N}\sum_{n=0}^{N-1}x_n\cos[\frac{\pi(2n+1)k}{2N}] \quad (18)$$

샘플링 포인트 8점에 대한 DCT의 일반 행렬식은 식 (19)와 식 (20)처럼 두 개의 4×4 계수 행렬과 입력벡터의 승산으로 표현할 수 있다.

$$\begin{bmatrix} z_0 \\ z_2 \\ z_4 \\ z_6 \end{bmatrix} = \begin{bmatrix} C_4 & C_4 & C_4 & C_4 \\ C_2 & C_6 & -C_6 & -C_2 \\ C_4 & -C_4 & -C_4 & C_4 \\ C_6 & -C_2 & C_2 & -C_6 \end{bmatrix} \begin{bmatrix} x_0+x_7 \\ x_1+x_6 \\ x_2+x_5 \\ x_3+x_4 \end{bmatrix} \quad (19)$$

$$\begin{bmatrix} z_1 \\ z_3 \\ z_5 \\ z_7 \end{bmatrix} = \begin{bmatrix} C_1 & C_3 & C_5 & C_7 \\ C_3 & -C_7 & -C_1 & -C_5 \\ C_5 & -C_1 & C_7 & C_3 \\ C_7 & -C_5 & C_3 & -C_1 \end{bmatrix} \begin{bmatrix} x_0-x_7 \\ x_1-x_6 \\ x_2-x_5 \\ x_3-x_4 \end{bmatrix} \quad (20)$$

여기서 $C_k = \cos(k\pi/16)$이다.

그림 15는 8점 DCT를 분산연산 구조로 구현하기 위한 블록도이다. 여기서 '입력데이터 재구성 부'는 8개의 입력데이터에 대하여 연산횟수를 줄이기 위해 계수행렬을 4×4로 분해하기 위한 부분이고, '행렬연산 부'는 식 (19)와 식 (20)을 계산하기 위한 부분이다.

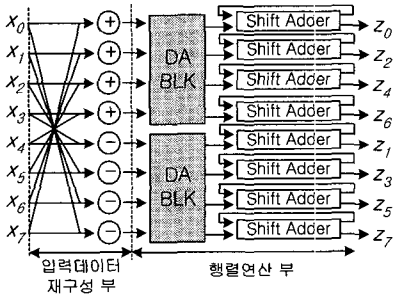


그림 15. 8점 DCT의 분산연산 구조 블럭도
Fig. 15. Block diagram of distributed arithmetic architecture for 8-point DCT.

본 논문에서는 식 (19)을 계산하기 위하여 그림 15의 'DA BLK'를 가산-네트워크로 설계하였다. 이를 위해 본 논문에서는 제안한 가산 공유항 추출 방법을 이용한 시뮬레이터를 설계하였다. 표 1은 식 (19)에 해당하는 계수 데이터에 대하여, 가산-네트워크를 구성하기 위해 추출한 가산 공유항 정보이다.

본 논문에서는 추출된 표 1의 공유항 정보를 기반으로 가산-네트워크를 구성하였다. 입력데이터와 계수데이터 크기(L, M)는 16비트로 하였고, 결과값 크기는 32비트로 설계하였다. 그림 16은 임의의 입력데이터 9654, 76, 2405, 5654일 때의 시뮬레이션 파형이다. 그림 17은 합성 결과를 나타내었다.

표 1. 식 (19)를 위한 최적의 가산기 공유항 추출
Table 1. The extraction of optimal adder sharing component for eqn. 19.

계수데이터	추출한 공유항 정보
$C_4\ C_4\ C_4\ C_4$	① $C_0-C_3,\ C_1-C_3$
$C_2\ C_6\ -C_6\ -C_2$	① $C_0-C_3,\ C_1-C_3$ ② $C_0-C_1,\ C_2-C_3$
$C_4\ -C_4\ -C_4\ C_4$	① $C_0-C_3,\ C_1-C_2$
$C_6\ -C_2\ C_2\ -C_6$	① $C_0-C_3,\ C_1-C_2$

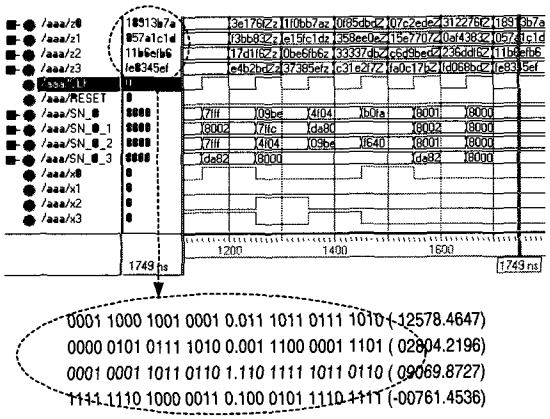


그림 16. 표 1의 추출정보를 기반으로 설계한 'DA BLK'의 시뮬레이션 파형.
Fig. 16. Simulation wave of 'DA BLK' based on extraction information of table 1.

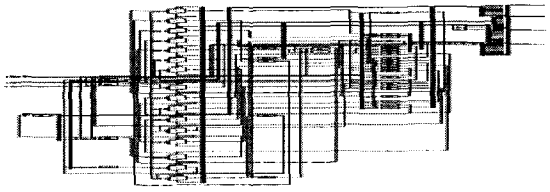


그림 17. 표 1의 추출정보를 기반으로 설계한 'DA BLK'의 합성 그림
Fig. 17. Synthesis figure of 'DA BLK' based on extraction information of table 1.

2. 결과 비교

표 2는 8점 DCT를 설계하는데 필요한 기존의 방법들을 비교하였다. 표 2에 나타난 바와 같이 FDCT (Fast DCT)[6~8]나 시스톨릭 구조[9]는 승산기 사용으로 인한 하드웨어 오버헤드가 매우 크다. 반면에 분산연산은 승산기 없이 DCT를 설계할 수 있고, 비트단위 파이

프라인 설계가 용이하기 때문에 훨씬 효율적인 설계가 가능하다.

표 2. 8점 DCT를 구현하는데 필요한 하드웨어 비교

Table 2. The comparison of required hareware to implement of 8-point DCT.

	Multi-plier	Adder/ Subtracter	Shift Adder	Additional Hardware
Chen[6]	16	26	-	-
Lee[7]	12	29	-	-
Loeffler[8]	11	29	-	-
Systolic[9]	8	16	-	-
분산연산	-	*8*	*8*	*ROM or Summation - network*

분산연산은 그림 15와 같은 블록도를 갖는다. 여기에서, 식 (19)를 계산하는데 필요한 '행렬연산 부'를 MAC, ROM-기반 분산연산, 제안한 방법을 이용한 분산연산으로 각각 설계하는 경우의 하드웨어를 비교하면 표 3과 같다.

표 3. 식 (19)의 행렬연산을 구현하는데 필요한 하드웨어 비교

Table 3. The comparison of required hareware to implement of eqn. 19.

	Adder	Shift Adder	Additional Hardware
MAC	12	16	-
ROM-기반 분산연산	-	4	ROM 18*16*4bit
Proposed	-	*4*	*가산-네트워크 (FA · FF 16개)*

표 3에서 MAC구조는 구현에 필요한 하드웨어가 매우 크다. 또한, 표 3에서 ROM-기반 분산연산과 제안한 방법의 차이점은 ROM 또는 가산-네트워크 사용여부이므로, 이 두 경우에 소비되는 게이트 및 동작 속도를 직접 설계하여 합성한 비교 결과가 표 4에 나타나있다. 합성은 LeonardoSpectrum Level 3의 v1999.1f 버전을 사용하였으며 'XCL05u' 라이브러리로 합성하였다.

표 4에서 'DA BLK'을 구현하는데, 제안한 방법을 이용하여 가산-네트워크로 설계하면 기존의 ROM을 사용하는 것보다도 훨씬 효율적인 결과를 얻을 수 있다. 또한, 제안한 방법을 이용하는 경우에는 가산 결과 발생하는 캐리가 FF에 저장되기 때문에, 가산된 결과를 출력하는 ROM-기반보다 출력포트가 $\log_2 M$만큼 작게 설계된다. 이는 'Shift Adder'가 보다 작은 크기로 설계할 수 있음을 의미한다. 그러나, 제안한 방법을 이용한 분산연산에서는 완전한 출력값을 얻는데 필요한 전체 클럭 수는 캐리에 저장된 데이터를 출력하도록 여분의 클럭($\log_2 M$)이 더 소비된다. 또한, 가산기 공유률이 높을수록 팬 아웃(Fan out)이 커진다는 단점도 가지지만, 전체적으로 소요되는 하드웨어 자원 및 동작속도 측면에서 기존의 방법들 보다 매우 효율적임을 알 수 있다.

표 4. 'DA BLK'의 합성결과

Table 4. Synthesis result of 'DA BLK'.

	Total gates	Frequency (MHz)	Total ports
ROM	718	127.1	77
가산-네트워크	*377*	*294.2*	*69*
Reduction	47.5%	231.4%	10.4%

Ⅵ. 결 론

신호처리 분야에서 사용되는 많은 알고리즘들은 대부분 내적연산을 갖고 있다. 본 논문에서는 DCT를 대상으로 내적연산을 효율적으로 처리할 수 있도록 가산기-기반 분산연산의 핵심 블록인 가산-네트워크를 최적으로 설계하였다. 또한, 가산-네트워크를 설계하는데 있어서 신경망의 볼츠만-머신을 이용하여 가산기 공유항 정보를 최적으로 추출하도록 일반화하였다. 제안한 설계 방법은 계수데이터에 의존하여 변화하는 가산-네트워크 구조의 설계를 최적화 할 수 있기 때문에, 내적연산이 필요한 다른 신호처리 분야에도 광범위하게 적용 할 수 있다.

참 고 문 헌

[1] Bernie New, "A distributed arithmetic approach to designing scalable DSP chips", EDN Design Feautre, Vol. Aug-17, pp. 107~114, Aug 1995.

[2] T.-S.Chang, C.Chen, C.-W.Jen, "New distributed arithmetic algorithm and its application to IDCT", IEE Proc. Circuits Devices Syst., vol.

146, No. 4, Aug 1999.

[3] James P. Coughlih, Robert H. Baran, "Neural Computation in Hopfield Networks and Boltzmann Machines", Univ of Delaware pr, February 1995.

[4] Benyamin D., Luk W., Villasenor J., "Optimizing FPGA-based vector product designs", FCCM '99 Proc., pp 188~197, 1999.

[5] Kumar D., Parhi K.K., "Performance trade-off of DCT architectures in Xilinx FPGAs", Signals Systems and Computers, Thirty-Third Asilomar Conference, Vol. 1, pp. 579~583, 1999.

[6] W.H.Chen, C.H.Smith, S.C.Fralick, "A fast computational algorithm for the discrete transform", IEEE Trans. Commun., Vol. COM-25, pp. 1004~1008. Sept. 1977.

[7] 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.

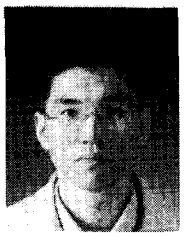
[8] C. Loeffler, A. Ligtenberg, G.S.Moschytz, "Practical fast AD DCT algorithm with 11 multiplications", In Proc. IEEE ECASSP, vol. 2, pp. 988~991, Feb. 1989.

[9] Sung Bum Pan and Rae-Hong Park, "Unified Systolic Arrays for Computation of the DCT/DST/DHT", IEEE Trans. Circuits and Systems for video Tech., vol 7, no 2, April 1997.

저 자 소 개

林 國 贊(正會員)

1977년 11월 20일생. 1999년 경희대학교 전자계산공학과 학사. 2001년 경희대학교 대학원 전자계산공학과 석사. 현재 LG전자 네트워크 연구소 연구원. 주요관심분야는 병렬처리 및 ASIC 설계



張 榮 眞(正會員)

1996년 경희대학교 전자계산공학과 학사. 1998년 경희대학교 대학원 전자계산공학과 석사. 1998년~현재 동 대학원 박사과정 재학 중. 주요관심분야는 병렬처리, 디지털 신호처리, ASIC 설계

李 顯 洙(正會員) 第33卷 B編 第10號 參照

1985~현재 경희대학교 전자정보학부 교수. 1999.9~2000.2 Oregon State Univ. ECE 객원교수. 2000.2~2000.8 Univ. of California. Irvine ECE 객원교수