2000年度 大韓電子工學會 秋季綜合學術大會 論文集 第23卷 第2號 2000/11

가산기-기반 분산 연산의 최적화 설계 및 이를 이용한 DCT 프로세서 설계

임 국 찬, 장 영 진, 이 현 수
경희대학교 전자계산공학과
전화 : 031-201-2947 / 핸드폰 : 016-707-2553

The Optimization Design of Adder-based Distributed Arithmetic and DCT Processor design

Guk-Chan Lim, Young-Jin Jang, Hyon-Soo Lee
Dept. of Computer Engineering, KyungHee University
E-mail : gclim@cann.kyunghee.ac.kr

Abstract

The Process of Inner Product has been widely used in a DSP. But it is difficult to implement by a dedicated hardware because it needs many computation steps for multiplication and addition. To reduce these steps, it is essential to design efficient hardware architecture.

This paper proposes the design method of adder-based distributed arithmetic for implementation of DCT module and the automatic design of summation-network which is a core block in the proposed design method.

Finally, it shows that the proposed design method is more efficient than a ROM-based distributed arithmetic which is the typical design method.

1. 서론

DSP(Digital Signal Processing) 분야에서 사용하는 알고리즘들은 계산상의 복잡성 때문에 효율적인 하드웨어 구조가 필수적이다. 특히, DCT(Discrete Cosine Transform)[1]와 같은 처리에서는 내적(Inner Product)을 어떠한 구조로 구현하느냐에 따라서 전체 시스템의 성능이 좌우되는 경우가 많다.

DA(Distributed Arithmetic) 알고리즘은 이러한 내적을 계산하는데 곱셈기를 사용하는 것보다 소비전력 및 크기를 효율적으로 줄일 수 있고, 고속동작이 가능한 회로 구현이 쉽기 때문에 소규모 신호처리 시스템 설계에 많이 이용되고 있다[2].

본 논문에서는 가산기-기반 DA의 핵심 블록인, 가산-네트워크(summation network)를 최적으로 설계함에 있어서 신경망의 홉필드(Hopfield) 네트워크[3]를 이용한 방법을 제안한다. 성능 평가를 위하여, 제안한 방법을 이용한 가산기-기반 DA와 기존의 ROM-기반 DA 방법을 이용하여 각각 DCT 모듈을 설계하였다.

시뮬레이션 결과, 제안한 방법을 이용한 가산기-기반 DA가 동작속도 및 소요되는 게이트 측면에서, 기존의 ROM-기반 DA보다 효율적임을 확인하였다. 또한, 제안한 설계 방법은 다양한 형태의 내적 계산에 대응되는 가산-네트워크를 단시간에 최적으로 설계할 수 있기 때문에, 응용 목적에 따른 재사용 모듈 설계가 어려운 가산기-기반 DA를 효율적으로 설계할 수 있다.

2. DA를 이용한 DCT

DCT와 같은 내적 연산을 구현하기 위해서 곱셈기를 사용하는 경우에는 하드웨어 오버헤드가 크기 때문에 DA 구조가 주로 사용된다[4].

DCT연산의 행렬식 표현은 다음과 같다.

$$\begin{bmatrix} z_0 \\ z_1 \\ z_2 \\ z_3 \\ z_4 \\ z_5 \\ z_6 \\ z_7 \end{bmatrix} = \begin{bmatrix} C_4 & C_4 & C_4 & C_4 & C_4 & C_4 & C_4 & C_4 \\ C_1 & C_3 & C_5 & C_7 & -C_7 & -C_5 & -C_3 & -C_1 \\ C_2 & C_6 & -C_6 & -C_2 & -C_2 & -C_6 & C_6 & C_2 \\ C_3 & -C_7 & -C_1 & -C_5 & C_5 & C_1 & C_7 & -C_3 \\ C_4 & -C_4 & -C_4 & C_4 & C_4 & -C_4 & -C_4 & C_4 \\ C_5 & -C_1 & C_7 & C_3 & -C_3 & -C_7 & C_1 & -C_5 \\ C_6 & -C_2 & C_2 & -C_6 & -C_6 & C_2 & -C_2 & C_6 \\ C_7 & -C_5 & C_3 & -C_1 & C_1 & -C_3 & C_5 & -C_7 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} \quad \text{(식 1)}$$

여기에서, $C_k = \cos(k\pi/16)$이다. k가 행, n이 열 인덱스라고 하면 (식 1)은 아래 수식으로 나타낼 수 있다.

$$z_k = \sum_{n=0}^{N-1} C_{k,n} x_n \quad \text{(식 2)}$$

(식 2)를 처리하기 위한 DA의 기본 구조는 그림 1과 같다.

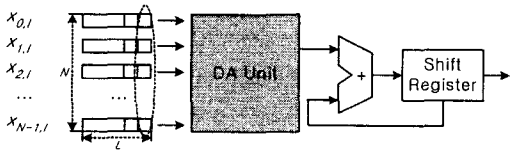

그림 1. DA의 기본 구조

2.1 기존의 ROM-기반 DA를 이용한 DCT

입력을 비트 단위로 확장하기 위해 (식 2)에 $x_n = \sum_{l=0}^{L-1} x_{n,l} 2^{-l}$을 대입하면 아래와 같이 변형된다.

$$z_k = \sum_{n=0}^{N-1} C_{k,n} \left(\sum_{l=0}^{L-1} x_{n,l} 2^{-l} \right) = \sum_{l=0}^{L-1} \left(\sum_{n=0}^{N-1} C_{k,n} x_{n,l} \right) 2^{-l} \quad \text{(식 3)}$$

ROM-기반 DA는, 그림 1의 DA Unit을 ROM으로 구성하여 계산된 결과를 미리 저장해 두는 방식이다. 그림 2는 ROM에 저장되는 값들의 예이다. 여기에서 M은 계수 데이터의 비트 크기(width)이다.

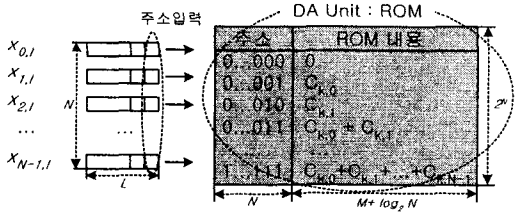

그림 2. ROM-기반 DA의 구조

2.2 가산기-기반 DA을 이용한 DCT

(식 3)의 계수 데이터를 비트 영역으로 확장하기 위해, $C_{k,n} = \sum_{m=0}^{M-1} C_{k,n,m}$을 대입시키면 아래와 같은 결과를 얻을 수 있다.

$$z_n = \sum_{l=0}^{L-1} \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} (C_{k,n,m} x_{n,l}) 2^{-l} \quad \text{(식 4)}$$

(식 4)을 비트 단위 DG(Dependency Graph)로 표현하면 그림 3과 같다.

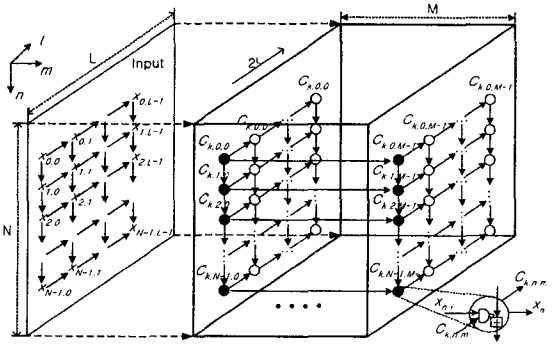

그림 3. 가산기-기반 DA의 비트 단위 DG

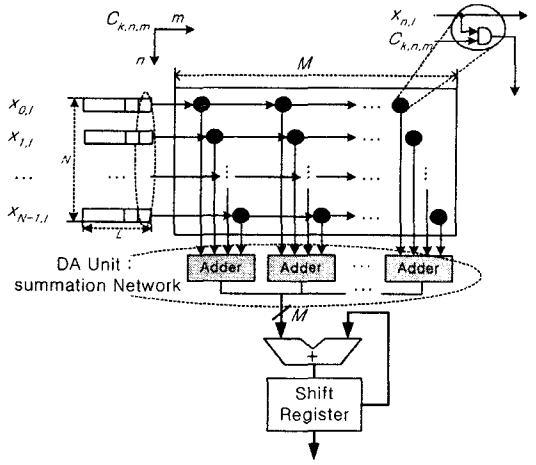

그림 4. 가산기-기반 DA의 구조

그림 3에서 l방향으로 프로젝션시키면, 그림 4처럼 계수의 비트 행렬만이 남게되고, 입력데이터는 직렬로 입력된다. 또한, 각 노드에 존재하는 덧셈 블록을 최종단에서 처리하도록 하면, 위 그림 4와 같이 나타낼 수 있다. 그림 4에서 $C_{k,n,m}$은 고정된 계수로 '1' 또는 '0'의 값을 갖는다. '0'일 때는 입력에 관계없이 최종결과가 영향을 받지 않으므로 생략할 수 있고, 계수 비트 패턴이 같은 곳은 덧셈 항을 공유할 수 있다. 따라서, 가산기-기반 DA는 이러한 정보를 얼마나 효율적으로 추출하여 가산-네트워크를 구성하느냐에 따라 성능이 좌우된다[5].

3. 가산기-기반 DA의 최적화 설계

3.1 홉필드 네트워크 설계

가산기-기반 DA에서 핵심블록인 가산-네트워크의 효율성은 다음 두 가지의 특성으로 결정된다.

1) Sparse non-zero bit property
2) Adder sharing property

1)은 계수 비트를 2진수로 표현할 때, 허용 에러 범위 내에서 '1'로 표현되는 부분을 적게 만드는 것이고, 2)는 공유 할 수 있는 가산기를 최적으로 추출함을 의미한다. 가산기의 공유 항을 추출하는 방법은 다음의 두 경우를 모두 고려해야 한다.

1) 계수 비트별 공유묶음은 공유되는 '1'의 수가 가장 많도록 $C_{k,i}-C_{k,j}$를 묶어야 한다.
   $\{i,j=0,1,2,\ldots,N-1,\ (\text{단},\ i\neq j)\}$
2) 공유묶음 결정순서가 다음 작업에 영향을 주기 때문에, 이를 고려한 순서가 결정되어야 한다.

이러한 최적화 문제를 해결하기 위해 홉필드 네트워크 알고리즘을 도입하여 가산-네트워크를 효율 좋게 설계할 수 있다.

그림 5는 공유묶음의 행렬식 표현방법의 예로, $C_{k,0}$ - $C_{k,2}$가 공유묶음으로 선택된 경우의 그림이다.

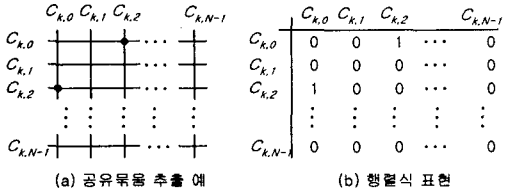

그림 5. 공유묶음 추출의 행렬식 표현

위 표현 방법이 최적의 상태를 만족하기 위해서 홉필드 네트워크는 다음의 동작조건들을 만족해야한다.

1. 한 행과 한 열에는 하나의 점만 활성화되어야 한다.
2. 대각선 점은 반드시 활성화 될 수 없다.
3. 활성화되는 점의 수는 $2\lfloor \log_2 N \rfloor$ 이다.
4. 공유묶음에 속하는 '1'의 수가 최대가 되어야 한다.
5. 위의 조건을 모두 만족하는 경우가 둘 이상일 때, 결합계수가 '0'을 포함한 공유묶음이 선택되어야한다.

각 조건에 대한 에너지 수식은 다음과 같다

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

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

$$E_3=\frac{D}{2}\sum_x\sum_i V_{xi}\delta_{xi}$$

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

$$E_5=\frac{F}{2}\sum_x\sum_i\sum_y\sum_j S(x,i)\delta_{xy}\delta_{ij}(1-\delta_{xi}) \qquad \text{(식 5)}$$

여기서 V_{xi}는 (x,i)의 출력으로 '0' 또는 '1'의 값을 갖고, δ_{ij}는 $i=j$일 때 '0'을 출력하고 그 외의 경우에는 '1'을 출력하는 함수이다. $S(x,i)$는 x와 i를 공유묶음으로 선택했을 때, 공유되는 '1'의 개수를 나타낸다.

홉필드 네트워크 전체 에너지 수식은 아래와 같다.

$$E=-\frac{1}{2}\sum_{x,y,i,j}T_{xi,yj}V_{xi}V_{yj}-\sum_{x,i}I_{xi}V_{xi} \qquad \text{(식 6)}$$

여기에서 T_{xi}는 결합계수이고 I_{xi}는 V_{xi}의 역치이다.

E_1,E_2,E_3,E_4,E_5의 합이 전체 네트워크 에너지이므로 $E=E_1+E_2+E_3+E_4+E_5$이다. 따라서, (식 5)와 (식 6)를 이용하여 결합계수 및 역치를 구하면 다음과 같다.

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

$$I_{xi}=2C\log_2 N-\frac{D}{2}\delta_{xi} \qquad \text{(식 7)}$$

(식 7)을 이용하여 홉필드 네트워크를 동작시키면 전체 에너지는 감소하며, 주어진 조건을 모두 만족했을 때 최소 에너지 즉, 최적화 상태를 갖게 된다.

3.2 가산-네트워크 최적화 설계 과정

제안한 가산-네트워크의 최적화 설계 과정은 그림 6과 같다.

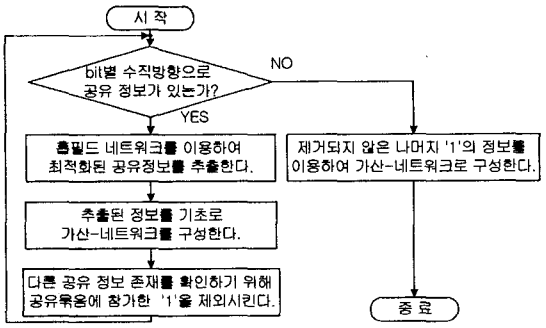

그림 6. 가산기-네트워크의 최적화 설계과정

그림 6의 최적화 설계과정에 대한, 동작 예가 그림 7에 나타나 있다.

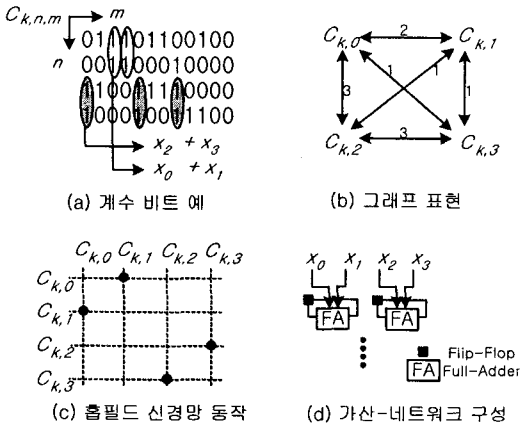

그림 7. 가산-네트워크 설계 예

그림 7의 (a)에서 $C_{k,n,m}$는 계수 비트이고 추출할 수 있는 공유 묶음의 예가 나타나 있다. (b)에는 계수가 비트 단위로 공유할 수 있는 '1'의 묶음 수를 그래프로 나타낸 그림이다. (c)는 홉필드 네트워크를 동작시켜 최적 값을 찾는 예이고, (d)는 추출된 정보를 바탕으로 가산-네트워크를 구성하는 그림이다. 이러한 과정을 반복함으로서, 최적 화된 가산-네트워크를 구성할 수 있다.

4. DCT 모듈 설계 및 평가

(식 1)의 8점 DCT를 처리하기 위해서, 8*8 계수 행렬의 특징을 이용하면 2개의 4*4행렬로 분해할 수 있다. 이에 대한 전체 블록도는 그림 8과 같다.

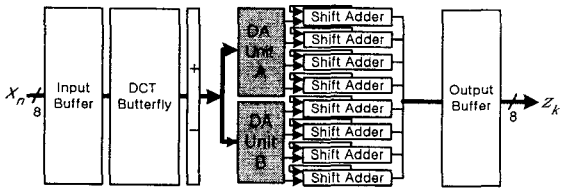

그림 8. DCT 전체 블록도

기존의 ROM-기반과 가산기-기반의 차이점은 DA Unit의 구현 방법으로, DA Unit이 ROM 또는 가산-네트워크로 구현된다. 그림 8에서 Shift Adder, 가/감산기, Butterfly 등은 공통적으로 사용하는 부분이므로, DA unit A만을 제안한 방법으로 가산-네트워크를 구성한 경우와 ROM을 이용한 경우로 설계하여 성능을 비교하였다. 입력과 계수는 16비트, 출력은 32비트로 가정하였다.

표 1. DA Unit A의 합성 결과 비교

	게이트 수	처리시간	동작주파수
ROM-기반	718	8.29	127.1MHz
제안한 방법이용	*377*	*3.96*	*294.2MHz*

표 1에서 알 수 있듯이 제안한 방법으로 구성한 가산-네트워크가 실제 ROM을 사용하는 것보다 크기와 동작속도 측면에서 좋은 성능을 보였다. 또한, 가산-네트워크의 구성 방법에 신경망을 도입함으로써, 최적화 설계에 대한 적절성과 유효함을 나타내었다.

설계한 가산-네트워크는 전가산기와 플립플롭을 각각 16개 사용하였고 290MHz 이상에서 동작이 가능하였다.

5. 결론

본 논문에서는 가산기-기반 DA의 핵심 블록인 가산-네트워크를 최적으로 설계할 수 있도록, 홉필드 네트워크를 이용한 설계방법을 제안하였다. 제안한 설계방법을 이용하여 DCT 모듈을 설계하는 경우가 기존 방식 보다 동작속도 및 소요되는 게이트 수가 효율적임을 시뮬레이션을 통해 확인하였다. 또한, 본 논문에서는 가산기-기반 DA의 핵심 블록인, 가산-네트워크의 설계를 자동화함으로서, 다양한 신호처리 분야에서 널리 이용될 수 있을 것으로 기대 된다.

References

[1] N.Ahmed, T.Natarajan, K.R.Rao, "Discrete cosine transform", IEEE Trans. Comput., Vol. C-23, pp. 90-94, Jan. 1974.

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

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

[4] D. Kumar and K.K. Parhi, "Performance Trade-off of DCT Architectures in Xilinx FPGAs", Proc. of 1999 Asilomar Conf. on Signals, Syst. and Comput. , Pacific Grove, CA, Oct. 24-27, 1999.

[5] 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, August 1999.