論文2003-40SC-6-7

처리율을 개선시킨 분산연산 방식의 IDCT 프로세서 설계

(A Design of high throughput IDCT processor in Distrited Arithmetic Method)

金 炳 民 *, 裵 絃 悳 **, 趙 泰 元 **

(Byung Min Kim, Hyeon Deok Bae, and Tae Won Cho)

요 약

본 논문에서는 가산기 기반 분산연산방식(Adder-Based DA)과 bit-serial방식을 적용한 8x1 1D-IDCT프로세서를 제안하였다. 하드웨어 소모를 줄이기 위해 bit-serial 방식을 적용하고 동작 속도의 향상을 위해 분산연산 방식을 적용한다. 또한 계수식의 변환을 통해 하드웨어 구현의 규칙성과 크기를 줄일 수 있으며 동작 클럭수를 줄이기 위해 부호 확장 처리 방식을 제안한다. 합성결과 게이트 수는 총 17,504개가 사용되었고 이중에서 부호 확장처리단은 전체 구조에서 20.6%를 사용하게 된다. 짝수, 홀수 부분에서는 기존의 계수표현에서 non-zero 비트가 130개가되지만, 제안한 방식을 적용한 짝수와 홀수 부분에서의 non-zero 비트는 각각 28개와 32개로 54% 줄일 수 있었다. 또한 부호 확장 처리단의 제안함으로써 처리율은 2배가 향상 되었고 설계한 IDCT 프로세서는 100㎒에서 50Mpixels/s의 처리율을 나타내었다.

Abstract

In this paper, An 8x1 1D-IDCT processor with adder-based distributed arithmetic(DA) and bit-serial method is presented. To reduce hardware cost and to improve operating speed, the proposed 8x1 1D-IDCT used the bit-serial method and DA method. The transform of coefficient equation results in reduction in hardware cost and has a regularity in implementation. The sign extension computation method reduces operation clock. As a result of logic synthesis, The gate count of designed 8x1 1D-IDCT is 17,504. The sign extension processing block has gate count of 3,620. That is 20% of total 8x1 1D-IDCT architecture. But the sign extension processing block improves more than twice in throughput. The designed IDCT processes 50Mpixels per second and at a clock frequency of 100㎒.

Keyword : DCT-IDCT, 분산연산방식, Bit-Serial방식

* 正會員, 라이온텍,
(Lion Tech)
** 正會員, 忠北大學校 電子工學科
(Department of Electronic Engineering Chungbuk National University)
※ 본 논문은 IDEC센터의 부분적 지원에 의해서 이루어졌음.
接受日字:2002年7月31日, 수정완료일:2003年10月15日

Ⅰ. 서 론

디지털 영상압축 및 복원 기술은 정지영상의 경우 주로 공간상에서의 중복성을 제거하기 위한 DCT (Discrete Cosine Transform)/IDCT(Inverse DCT)등을 이용한 변환부호화를 기반으로 하고 있으며 동영상의 경우는 DCT/IDCT와 화면간 물체들의 움직임 보상 (Motion Compensation)을 이용한 예측부호화를 기반으

로 하고 있다[1,2]. 또한 영상정보의 압축 기술이 빠르게 발전되면서 영상정보도 컴퓨터 시스템에서 다룰 수 있게 됨으로써 컴퓨터와 통신의 융합. 더해 영상, 즉 방송의 기능이 추가로 통합되어 가고 있다. 이러한 압축기술들에 대한 표준으로 정지화상용 JPEG, 음성용 MPEG Audio, 동화상용 H.261, MPEG(1/2/4), 압축된 정보들을 다중화 시키거나 분리시키고 동기 시키는 MPEG 시스템 기술에서 DCT/IDCT는 데이터의 압축 복원에 필수적 요소로 채택되었으며 HDTV 등의 영상 압축 및 복원에 대한 국제 표준의 핵심 기술로 채택되고 있다.

DCT/IDCT 블록은 계산량이 많은 주요 연산 블록으로 승산기를 사용하여 구현하는 것이 일반적이다. 그러나 승산기의 사용은 하드웨어 구현시 많은 면적을 요구하는 단점이 있다. 또 하나의 방법으로는 분산연산방식(DA : Distributed Arithmetic)[4-6]이 있다. 분산연산방식에는 롬(ROM) 기반 분산연산방식(ROM-based DA)[5]과 가산기 기반 분산연산방식(Adder-Based DA)[6]으로 분류할 수 있는데 롬 기반 분산연산방식은 가장 많이 사용되고 있는 방식으로서 구현방법이 간단하나 클럭당 처리 비트수가 커질수록 롬의 크기가 증가하는 단점이 있다. 반면 가산기 기반 분산연산방식은 가산기와 공유성을 이용하여 계수의 non-zero 비트만을 하드웨어로 구현되기 때문에 크기 및 동작속도 측면에서 효율적인 구현이 가능하다[6].

본 논문에서는 기존의 가산기 기반 분산연산방식과 bit serial 방식을 사용하며 게이트수를 감소시켰다. 또한 계수의 변형을 통해 구현되는 하드웨어의 수와, 입력 화소값의 부호확장으로 인해 증가되는 동작 클럭수를 부호 확장처리부에서 처리함으로서 동작 클럭수를 감소시켰다.

본 논문의 구성은 II절에서 일반적인 IDCT 알고리듬 및 분산연산 구조에 대해 설명하고 III절에서는 설계한 IDCT 프로세서의 구조에 대해서 설명한다. IV절에는 실험 결과를 V절에서는 비교 및 분석한 뒤 VI절에서 결론을 맺는다.

II. IDCT 알고리듬 및 가산기 기반 분산연산구조

IDCT 알고리즘은 영상 복원을 위한 효과적인 디코딩기술인데 이를 이용하여 처리된 트랜스폼-영역의 데이터 블록들을 원 데이터로 변환시킨 것이다. IDCT 계수를 C, 입력벡터를 X, 그리고 변환 행렬을 Z라 하면, 2D-IDCT는 $Z=C^TXC$로 표현 할 수 있다. 3개의 행렬을 곱하는 것은 두 번의 행렬-벡터 곱과 한번의 행렬 전치를 수행함으로써 계산할 수 있다. 입력되는 벡터가 X^T일 때 첫 번째 1D 변환은 $x=C^TX^T$가 되고, 이를 $x^T=XC$로 전치한 다음에 다시 두 번째 1D 변환을 수행하면 $Z=C^TXC$가 된다.

N-point 분산연산의 기본식은 식 (1)와 같이 나타낼 수 있다.

$$Y_i=\sum_{j=1}^{N} C_{i,j}\times X_i \qquad (i=1,2,3,\cdots,N). \tag{1}$$

여기서 X_i는 입력 화소값이고 $C_{i,j}$는 계수 행렬값이다. i,j는 계수 행렬에서의 위치값을 나타내고 Y_i는 1D-IDCT의 결과값을 나타낸다. 식 (1)의 $Y_1 \sim Y_N$을 일반적인 내적 연산의 경우에 대해서 고려할 때 길이가 N인 내적 연산 Y=CX는 다음과 같이 나타낼 수 있다[6].

$$Y_i=\sum_{j=1}^{N} C_i\times X_i, \tag{2}$$

$$C_i=\sum_{j=1}^{M} C_{i,j}2^{-j}, \qquad X_i=\sum_{k=1}^{L} X_{i,k}2^{-k}.$$

분산연산이란 벡터와 행렬의 곱셈을 수행할 때 비트 단위로 순차적으로 처리하는 연산을 말한다. 비트 단위로 고려하기 위해 식 (2)에서 C_i와 X_i를 각각 2의 보수 형태로 나타내면 식 (3)과 같이 표현되고 X_i를 비트 단위로 고려하기 위해 식 (3)의 $X_i=\sum_{k=1}^{L} X_{i,k}2^{-k}$을 식 (2)에 대입하면 식 (4)와 같이 표현된다. 이와 같이 입력값을 분해하는 방식을 롬 기반 분산연산 방식이라 한다[6].

$$Y=\sum_{i=1}^{N} C_i\times\left(\sum_{k=1}^{L} X_{i,k}2^{-k}\right)=\sum_{k=1}^{L}\left(\sum_{i=1}^{N} C_i\times X_{i,k}\right)2^{-k} \tag{4}$$

C_i를 비트 단위로 고려할 경우 식 (3)의 $C_i=\sum_{j=1}^{M} C_{i,j}2^{-j}$ 을 식 (2)에 대입하면 식 (5)와 같이 표현되고 이와 같이 계수를 분해하는 방식을 가산기 기반 분산연산 방식이라 한다[6].

$$Y=\sum_{i=1}^{N} X_i\times\left(\sum_{j=1}^{M} C_{i,j}2^{-j}\right)=\sum_{j=1}^{M}\left(\sum_{i=1}^{N} C_{i,j}\times X_i\right)2^{-j}. \tag{5}$$

8-포인트 1D-IDCT의 일반식은 식 (6)과 같다. 식 (6)에서 Z는 1D-IDCT결과의 위치값을 나타낸다.

$$x(z)=1/2\sum_{u=0}^{7}c_uX_u\cos[\pi(2h+1)u/16], \quad (6)$$

$$where \quad c_u=1/\sqrt{2}, \quad for \quad u=0$$
$$c_u=1, \quad otherwise.$$

위의 식 (6)을 행렬형태로 나타낼 경우 8x8 행렬 형태로 표현되나 8x8 행렬계산은 계산량이 많아지기 때문에 식 (7), 식 (8)와 같이 4x4 행렬형태로 나누어서 계산하게 된다.

$$\begin{bmatrix}x_0\\x_1\\x_2\\x_3\end{bmatrix}=\frac{1}{2}\begin{bmatrix}C_4 & C_2 & C_4 & C_6\\ C_4 & C_6 & -C_4 & -C_2\\ C_4 & -C_6 & -C_4 & C_2\\ C_4 & -C_2 & C_4 & -C_6\end{bmatrix}\begin{bmatrix}X_0\\X_2\\X_4\\X_6\end{bmatrix}+\frac{1}{2}\begin{bmatrix}C_1 & C_3 & C_5 & C_7\\ C_3 & -C_7 & -C_1 & -C_5\\ C_5 & -C_1 & C_7 & C_3\\ C_7 & -C_5 & C_3 & -C_1\end{bmatrix}\begin{bmatrix}X_1\\X_3\\X_5\\X_7\end{bmatrix}, \quad (7)$$

$$\begin{bmatrix}x_7\\x_6\\x_5\\x_4\end{bmatrix}=\frac{1}{2}\begin{bmatrix}C_4 & C_2 & C_4 & C_6\\ C_4 & C_6 & -C_4 & -C_2\\ C_4 & -C_6 & -C_4 & C_2\\ C_4 & -C_2 & C_4 & -C_6\end{bmatrix}\begin{bmatrix}X_0\\X_2\\X_4\\X_6\end{bmatrix}-\frac{1}{2}\begin{bmatrix}C_1 & C_3 & C_5 & C_7\\ C_3 & -C_7 & -C_1 & -C_5\\ C_5 & -C_1 & C_7 & C_3\\ C_7 & -C_5 & C_3 & -C_1\end{bmatrix}\begin{bmatrix}X_1\\X_3\\X_5\\X_7\end{bmatrix}, \quad (8)$$

$$where \quad C_k=\cos[\pi k/16], \quad k=2z+1.$$

Ⅲ. 제안한 IDCT 프로세서 구조

제안한 IDCT프로세서는 가산기와 감산기의 개수를 줄임으로써 하드웨어의 면적을 최소화 할 수 있도록 IDCT 계수식의 변형을 통해 전가산기(FA)와 감산기의 갯수를 줄였다. 계수식의 변형은 양의 계수만으로 계수처리단을 구성함으로써 규칙성을 가지며 이로 인하여 하드웨어 구현을 쉽게 하였다. 또한 제안한 구조는 bit-serial 방식을 적용하여 하드웨어 소모를 줄였고 bit-serial 방식의 적용으로 인한 처리율의 감소를 향상시키기 위해 분산연산방식을 적용하여 처리율을 향상시켰다. 또한 입력 화소값의 부호 확장으로 인한 부가적인 클럭수 줄이는 구조를 제안하여 동작속도를 향상시켰으며 8x1 1D-IDCT에서는 행렬분해 방식을 적용하여 구현하였다.

식 (7), 식 (8)은 다음과 같이 짝수부분(식 (9))과 홀수부분(식 (10))으로 나누어 질 수 있다.

$$\begin{bmatrix}U_0\\U_1\\U_2\\U_3\end{bmatrix}=\begin{bmatrix}x_0+x_7\\x_1+x_6\\x_2+x_5\\x_3+x_4\end{bmatrix}=\begin{bmatrix}C_4 & C_2 & C_4 & C_6\\ C_4 & C_6 & -C_4 & -C_2\\ C_4 & -C_6 & -C_4 & C_2\\ C_4 & -C_2 & C_4 & -C_6\end{bmatrix}\begin{bmatrix}X_0\\X_2\\X_4\\X_6\end{bmatrix}, \quad (9)$$

$$\begin{bmatrix}V_0\\V_1\\V_2\\V_3\end{bmatrix}=\begin{bmatrix}x_0-x_7\\x_1-x_6\\x_2-x_5\\x_3-x_4\end{bmatrix}=\begin{bmatrix}C_1 & C_3 & C_5 & C_7\\ C_3 & -C_7 & -C_1 & -C_5\\ C_5 & -C_1 & C_7 & C_3\\ C_7 & -C_5 & C_3 & -C_1\end{bmatrix}\begin{bmatrix}X_1\\X_3\\X_5\\X_7\end{bmatrix}. \quad (10)$$

여기서 계수부분에서의 non-zero 비트와 비트별로 들어오는 입력값이 가산기와 감산기를 통해 계산이 되므로, 식의 변형을 통해 가산기와 감산기의 숫자를 줄일 수 있었다. 식 (9)의 행렬형태를 일반식으로 풀어서 표현한 식이 식 (11)이다. 제안한 구조는 식 (11)을 식 (12)와 같이 계수의 음(-)의 값을 입력 데이터에 넘겨줌으로써 식을 변형하였다. 각각의 식에서 계수에 포함된 음(-)의 값은 계수 처리단(Coefficient Processing Block)에서 감산기를 만들어야 하는 부담이 있으므로 감산기를 없애기 위해 계수의 음(-)의 값을 식 (12)와 같이 입력 데이터에 넘겨주어 전처리단(Pre Computation Block)에서 먼저 계산이 되도록 하고 계수 처리단에서는 양의 계수값에 대해 D-플립플롭(D-F/F)과 전가산기만을 사용하여 구현하였다.

$$\begin{aligned}U_0&=C_4X_0+C_2X_2+C_4X_4+C_6X_6\\ U_1&=C_4X_0+C_6X_2-C_4X_4-C_2X_6\\ U_2&=C_4X_0-C_6X_2-C_4X_4+C_2X_6\\ U_3&=C_4X_0-C_2X_2+C_4X_4-C_6X_6,\end{aligned} \quad (11)$$

$$\begin{aligned}U_0&=C_4X_0+C_2X_2+C_4X_4+C_6X_6\\ U_1&=C_4X_0+C_6X_2+C_4(-X_4)+C_2(-X_6)\\ U_2&=C_4X_0+C_6(-X_2)+C_4(-X_4)+C_2X_6\\ U_3&=C_4X_0+C_2(-X_2)+C_4X_4+C_6(-X_6).\end{aligned} \quad (12)$$

같은 방식으로 홀수 부분에 대해서도 식 (10)의 식을 풀어서 표현한 뒤 짝수 부분에서 행한 방식으로 적용하면 계수의 음(-)의 값은 입력값에서 처리가 되고 계수처리단에서는 양의 계수값으로만 구현된다.

<그림 1>은 8x1 1D-IDCT 구조의 처리과정을 나타내는 블록도 이고 <그림 2>는 8x1 1D-IDCT 구조의 상세도 이다. 8x1은 크게 입력부, 전처리부, 계수처리부, 부호확장처리부, 후처리부로 구성된다.

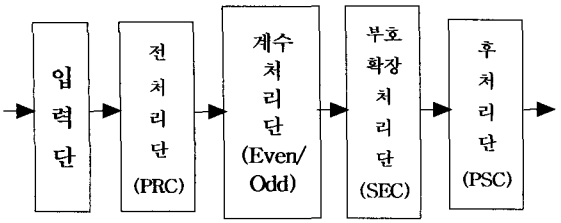


그림 1. 8x1 IDCT 구조의 블록도
Fig. 1. Block diagram of an 8x1 IDCT Architecture.

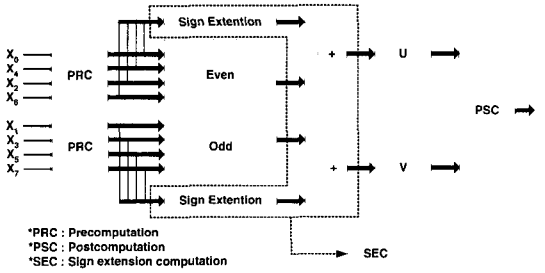


그림 2. 8x1 IDCT 구조의 상세도
Fig. 2. Detailed diagram of an 8x1 IDCT Architecture.

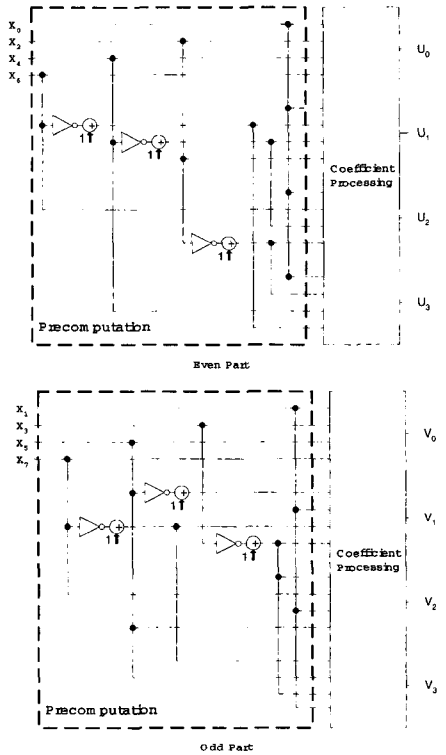


그림 3. 전처리단 구조
Fig. 3. Precomputation architecture(PRC).

입력부는 병렬로 입력되는 화소값들을 LSB부터 비트별로 출력하기 위해 단순한 parallel-to-serial 레지스터를 사용하였다. 클럭당 처리 비트수에 따라 입력부에서 출력되는 비트수는 다르게된다. 제안한 구조는 1bit/clock(bit per clock)의 구조로써 입력부에서는 클럭당 1비트씩 다음단(전처리단)의 입력으로 출력하게된다.

전처리단은 앞의 수식 (11), 식 (12)에서 설명한 것처럼 계수의 음(-)의 값을 입력 데이터에서 처리하기 위해 사용되는 구조이다. 이러한 방법으로 계수처리단에서 사용되는 계수값에 대해서 non-zero 비트를 줄임으로서 하드웨어의 감소효과를 볼 수 있다. <그림 3>은 짝수 부분과 홀수 부분의 전처리단 구조이다.

계수 처리단은 짝수 부분과 홀수 부분으로 구성되어 있다. 짝수 부분에서 사용되는 계수값은 식 (12)에서 보인 것처럼 IDCT 식의 변형을 통해 계수의 음(-)의 값을 전처리단을 이용해 입력 데이터에 부가시킴으로서 계수 처리단에서 감산기가 필요하지 않게 되었다. 또한 계수 처리단에서 사용되는 전가산기의 개수도 줄일 수 있었다. 가산기 기반 분산연산에서 non-zero 비트의 감소에 의한 전가산기의 감소는 면적의 감소 효과를 볼 수 있다.

<표 1>에서는 코사인 계수를 16비트 2진수로 나타낸 것이다. 짝수 부분에서는 기존의 계수표현에서 non-zero 비트가 62개가되지만, 제안한 방식을 적용한 짝수 부분에서의 non-zero 비트는 28개로 54% 줄일 수 있었다. 그리고 홀수 부분에서는 기존의 계수 표현에서는 non-zero 비트가 68개인데 반해 제안된 방식은 32개로 줄임으로서 53%의 감소 효과를 얻었다. 줄어든 non-

표 1. 기존구조와 제안한 구조에서 사용되는 계수의 non-zero 비트 비교
Table 1. Comparison of the number of non-zero bit in conventional and proposed architecture.

		Coefficients	Binary	# of non-zeros
일반적인 구조	짝수 부분	C_0,C_2,C_4,C_6, $C_0,-C_2,-C_4,-C_6$		62
	홀수 부분	C_1,C_3,C_5,C_7, $C_1,-C_3,-C_5,-C_7$		68
제안한 구조	짝수 부분	C_4	0.0101101010000010	28
		C_2	0.0111011001000001	
		C_4	0.0101101010000010	
		C_6	0.0011000011111011	
	홀수 부분	C_1	0.0111110110001010	32
		C_3	0.0001100011111000	
		C_5	0.0110101001101101	
		C_7	0.0100011100011100	

zero 비트의 수만큼 하드웨어 구현 시 면적이 줄어든다.

짝수 부분의 식은 식 (11)에서 식 (12)와 같은 식으로 변형 할 수 있다. 여기서 기존의 계수표현에서는 C_2, C_4, C_6, $-C_2$, $-C_4$, $-C_6$ 로 나타나는 것과는 달리 계수에서의 음(-)의 부분을 없애고 대신 음(-)의 처리를 입력 X_0, X_2, X_4, X_6에 적용함으로 추가적인 $-X_2$, $-X_4$, $-X_6$을 처리해야 하는 전처리단 블록이 발생한다. 그러나 계수처리에 있어서는 기존 C_2, C_4, C_6, $-C_2$, $-C_4$, $-C_6$의 6가지의 계수 값을 처리 해 주어야 하나 본 논문에서는 C_4, C_2, C_4, C_6의 4가지 계수 값으로 처리 해 줄 수 있다.

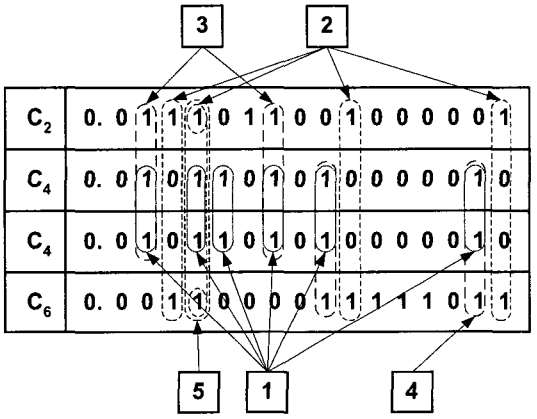


그림 4. 짝수 부분의 공유항
Fig. 4. Common sharing of an Even part.

또, C_4, C_2, C_4, C_6을 <그림 4>와 같이 공유항(Common Subexpression Sharing)기법을 사용함으로서 4가지 계수 C_4, C_2, C_4, C_6과 4가지 입력데이터 X_0, X_2, X_4, X_6의 곱에 대한 합의 연산을 하드웨어로 구현하는데 있어서 간단한 구조를 제시하였다. 그림은 각 계수 값 C_2, C_4 ,C_4, C_6을 나타낸 것으로 '1'로 나타난 비트에 대해서만 전가산기가 필요하게된다. 전가산기 [1]의 경우 전가산기 [1]이 사용되는 세 개의 계수값 C_2, C_4, C_4의 각 자리수의 계수값이 '1'이므로 결국 $X_0+X_2+X_4$의 계산결과 같게된다. 즉 전가산기 [1]의 출력이 사용되는 비트 위치는 2^2, 2^4, 2^5, 2^7, 2^9, 2^{15}이 된다. 같은 방식으로 2^2, 2^7비트의 위치에서는 전가산기 [3]의 출력으로 사용된다. 전가산기 [2], [4], [5]도 같은 방식으로 구성하면 짝수 부분의 summation network에서 사용되는 전가산기의 수는 5개가된다. <그림 5>에서는 전가산기와 D 플립플롭을 사용하여 짝수부분을 구성한 블록도를 나타낸 것이다. 이와 같은 블록이 계수처리단에서 4개가 사용된다. 4개의 짝수 부분 블록 입력은 짝수 부분의 식에서 나타난 X_0, X_2, X_4, X_6를 전처리단(pre-computation)에서 계산된 값 X_0, X_2, $-X_2$, X_4, $-X_4$, X_6, $-X_6$으로 입력을 받는다.

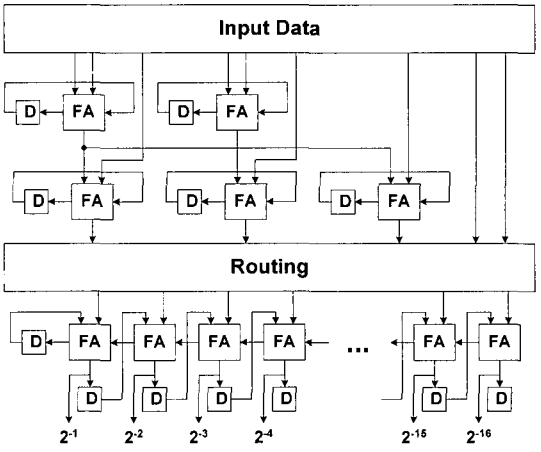


그림 5. 짝수 부분 블록도
Fig. 5. Block diagram of Even part.

<그림 5>에서 라우팅 부분 위에 5개의 전가산기는 <그림 4>에서 나타난 공유된 1비트 전가산기를 나타낸 것이며, 우측의 입력에서 내려오는 화살표들은 <그림 5>에서 공유되지 않은 non-zero 비트 처리를 나타낸 것이다. Routing 아래에 나타난 전가산기와 D 플립플롭은 16비트로 적용된 계수 값의 비트별 쉬프트된 것을 나타내기 위한 shift-adder단이다. 처음 클럭에서 입력되는 데이터의 입력 1비트 값은 LSB값부터 MSB값으로 입력되기 때문에 <그림 6>에서 출력되는 값은 처음 클럭에서 2^1~2^{16}비트 자리의 값이 출력되고 다음 클럭에서 출력되는 값은 2^0~2^{15}비트 자리값이 출력된다. 이러한 방법으로 입력데이터 12비트는 12클럭에 걸쳐 입력이 되고 출력값은 14클럭(입력클럭수+2클럭)후에 결과값을 볼 수 있다. 2클럭이 더 소모되는 이유는 <그림 5>의 블록도에서 나와있는 것처럼 내부 D 플립플롭에 저장되어있는 값들에 대한 계산 결과를 얻기 위해 필요한 것이다.

홀수 부분도 짝수 부분과 같이 식의 변형을 통해 구현 할 수 있다. 기존의 홀수 부분을 구현하기 위해 사용되는 계수는 C_1, C_3, C_5, C_7, $-C_1$, $-C_3$, $-C_5$, $-C_7$의 8

가지 계수 값들이 필요하였다. 그러나 본 논문에서 식의 변형으로 홀수 부분의 구현에 사용되는 계수가 C_1, C_3, C_5, C_7 의 4가지로 줄어든다.

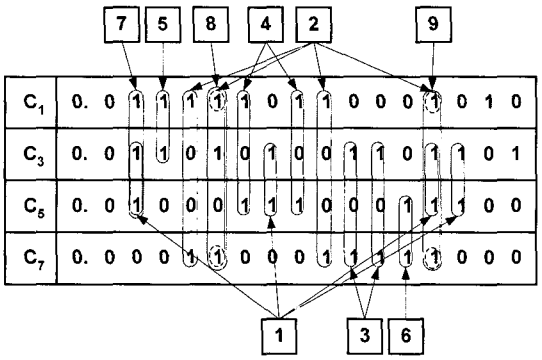


그림 6. 홀수 부분의 공유항
Fig. 6. Common sharing of an Odd part.

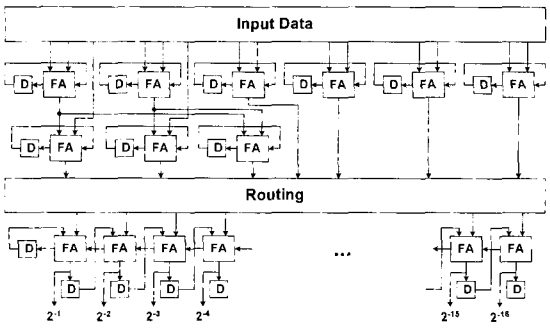


그림 7. 홀수 부분 블록도
Fig. 7. Block diagram of an Odd part.

홀수 부분도 짝수 부분과 같이 계수의 음(-)의 값을 입력 X_1, X_3, X_5, X_7의 전처리단에서 처리함으로써 계수처리단의 summation network의 하드웨어를 줄였다. 대신 입력값은 X_1, X_3, X_5, X_7에서 X_1, X_3, X_5, X_7, $-X_3$, $-X_5$, $-X_7$로 각각 바뀐다. 홀수 부분에서 사용되는 계수 C_1, C_3, C_5, C_7에 대한 공유항은 <그림 6>에서 나타내었다. 짝수 부분과 같은 방법으로 구현할 때 홀수 부분은 짝수 부분보다 공유될 수 있는 '1'의 값이 적기 때문에 짝수 부분보다 많은 전가산기가 필요하게 된다. 홀수 부분에서 사용되는 전가산기는 <그림 6>처럼 9개의 전가산기가 필요하게 된다. <그림 7>에서는 전가산기와 D 플립플롭을 사용하여 홀수부분을 구성한 블록도를 나타낸 것이다.

부호 확장처리단은 입력화소값이 음수일 경우에 고려될 사항으로 일반적인 add & shift 형태의 곱셈연산에서 양수(피승수) x 음수(승수) 형태의 곱셈을 처리하기 위해서는 음수의 MSB를 생성된 결과값의 비트만큼 부호 확장을 해서 연산을 해야만 정확한 결과를 얻을 수 있다. 부호확장 처리단은 이러한 두 수의 곱셈에서 음수를 부호확장하지 않고 정확한 값을 얻기 위해 처리되는 부분이다.

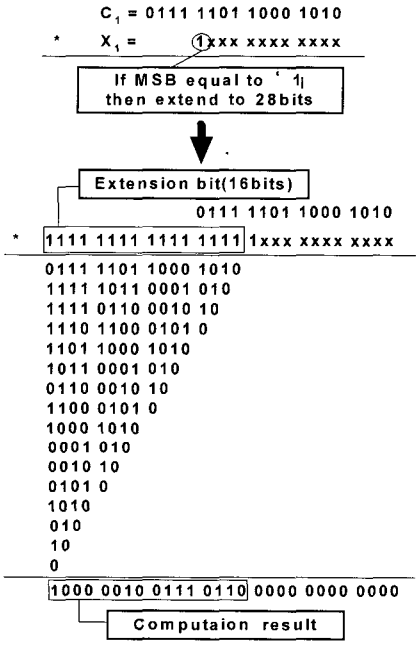


그림 8. 계수 C_1에 대한 부호 확장값 계산
Fig. 8. Computing the sign extension value of the coefficient, C_1.

본 논문에서 사용되고있는 입력데이터 12비트, 고정계수값 16비트의 곱셈을 처리할 경우(12 x 16) 28비트의 곱셈 결과가 생성되는데 입력데이터 12비트가 처리되기 위해서는 12클럭이 필요하다. 그러나 입력데이터가 음수일 경우 입력데이터는 28비트로 부호확장을 하게된다. 즉 입력데이터가 1000 1100 0000(12비트)인 경우 올바른 곱셈결과를 얻기 위해서는 1111 1111 1111 1111 1000 1100 0000(28비트)로 부호를 확장해야한다. 그러나 이러한 방식으로 입력데이터의 부호확장 방법을 본 논문에서 사용하는 분산연산 방식에 적용시킬 경우 입력데이터가 처리되기 위해서 28클럭(12클럭:원 입력데이터 클럭 + 16클럭:부호확장 클럭)이 필요하게 되므로 연산속도에서의 감소효과를 줄 수 있다. 이렇게 확장된 부호를 처리하기 위한 방법으로는 클럭당 처리 비트수를 증가시키는 방법이 있는데 이러한 방식은 하

드웨어에서의 증가되는 단점을 가지고 있다. 또한 동작 클럭수는 줄일 수 있으나 클럭을 결정하는 임계경로가 증가하기 때문에 큰 효과를 볼 수는 없다. 따라서 본 논문에서 제안한 방식은 입력데이터가 음수일 경우 부호 확장된 값에 대한 곱의 결과를 미리 계산하여 저장한 뒤 입력데이터의 MSB를 조사하여 '1'인 경우 미리 계산된 결과를 계수 처리단에서 생성된 결과와 더함으로서 정확한 결과를 얻을 수 있다.

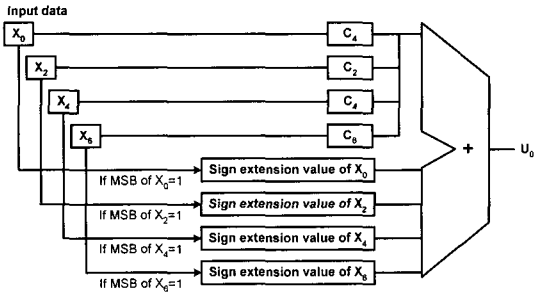


그림 9. 부호 확장처리단 블록도
Fig. 9. Block diagram of a sign extension processing part.

계수 C_1과 입력데이터 X_1이 곱해질 경우 X1의 MSB가 '1'일 때 <그림 8>과 같이 입력데이터 X_1을 13비트에서 28비트까지 '1'로 확장하여 계산된 결과를 롬에 저장하였다가 입력데이터의 MSB가 '1'일 경우 저장된 값과 계수처리단에서 출력된 값을 더한다. 이와 같은 방식으로 C_2~C_7까지 적용하게되면 12비트 입력데이터의 MSB를 28비트까지 확장할 필요가 없어지게 되고 따라서 입력데이터를 처리하는데 부호확장을 할 경우 28클럭에서 14클럭만이 필요하게된다. 이것은 처리속도에 향상을 가져온다. 계수 C_2~C_7에 대해서도 C_1과 같은 방식으로 계산할 경우 각각의 부화 확장 값은 <표 2>와 같이 된다.

표 2. 계수들의 부호 확장값
Table 2. Sign extension value of coefficients.

계수	부호 확장값
C_1	1000 0010 0111 0110
C_2	1000 1001 1011 1111
C_3	1001 0101 1001 0011
C_4	1010 0101 0111 1110
C_5	1011 1000 1110 0100
C_6	1100 1111 0000 0101
C_7	1110 0111 0000 1000

짝수 부분의 경우 식 (9)와 <그림 3>에서처럼 중간 결과값 U_0~U_3가 발생하게되는데 하나의 U#값을 얻기 위해 4개의 계수값과 4개의 입력값이 필요하게된다. 이 경우 4개의 입력값이 모두 음수인 경우 발생되는 부호 확장값이 각각의 입력에 대해서 발생하기 때문에 계수 처리단에서 출력된 결과값에 4개의 부호 확장값을 더해주어야 한다. 즉 <그림 9>와 같이 U_0값은 $C_4X_0+C_2X_2+C_4X_4+C_6X_6$의 계산결과로 얻는다. 앞의 계산식에서 X_0, X_2, X_4, X_6의 값이 음수인 경우 본 논문에서 제시한 부호 확장처리방식을 적용할 경우 최종적인 U_0값을 얻기 위해서는 <그림 9>에서처럼 결과값에 4개의 부호 확장값을 더해야한다.

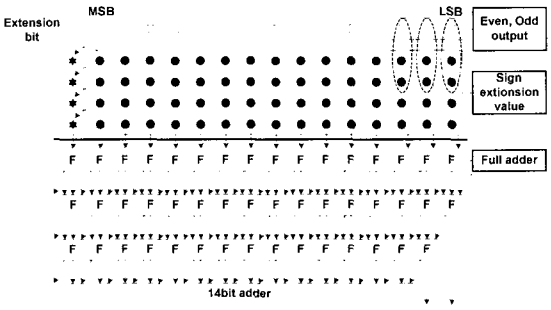


그림 10. 부호 확장처리부 상세도
Fig. 10. Detailed diagram of sign extension processing part.

<그림 10>은 부호 확장처리단의 상세도를 나타낸 것이다. 그림은 5개의 16비트의 수를 더하는 과정을 나타낸 것인데 일반적인 16비트 가산기(Δt:16비트 가산기 지연시간)를 사용할 경우 $3\Delta t$의 시간이 걸리지만 <그림 10>과 같은 구조에서는 1비트 전가산기 지연시간(Δf)+ 14비트 가산기지연시간(Δy)이 걸리게 된다.

비교를 위해 가산기를 RCA(Ripple Carry Adder)로 구성하였을 때 5개의 16비트의 수를 더하기 위해 소모되는 시간은 $16 \times 3 \times \Delta f = 48\Delta f$가 된다. 반면 본 논문에서 사용한 구조에서는 $3 \times \Delta f + 14 \times \Delta f = 17\Delta f$의 시간이 걸린다. 따라서 일반적인 방법보다는 2배 이상 빠른 성능을 보인다.

Ⅳ. 실험 결과

제안한 IDCT 프로세서의 검증방법으로는, VHDL로 코딩하여 Altera사의 Max+plus Ⅱ로 논리 검증을 하였

으며 Compass를 사용하여 0.6㎛공정 라이브러리로 합성 및 모의실험을 하였다. 또한 Synopsys의 design_analyzer를 통해 합성된 전체 블록을 확인하였다.

<표 3>은 제안한 8x1 1D-IDCT를 모의실험한 결과로써, 합성결과 최대 동작 속도는 100㎒는 계수 처리단에서 결정되는데 1비트 화소값이 입력되어 계수처리단의 출력으로 나오기까지 소요되는 시간에 의해서 결정된다. 따라서 계수처리단이 임계경로를 가지는 블록이 되고 최대 동작 주파수가 100㎒가 된다.

표 3. 제안한 8x1 1D-IDCT의 모의실험 결과.
Table 3. Simulation results of the proposed 8x1 1D-IDCT.

Technology	0.6㎛, 3-metal CMOS
Function	IDCT
Data format	IN : 12bit
	OUT : 9bit
Gate count	17,400
Transistor count	69,400
Clock rate	100㎒
Throughput	50Mpixels/s

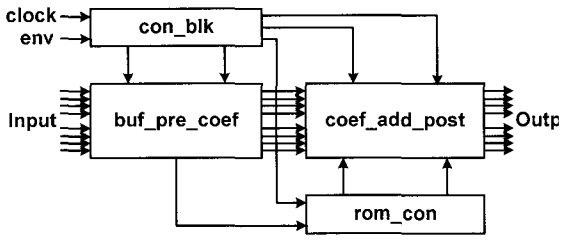


그림 11. 합성 블록도
Fig. 11. Synthesis block diagram.

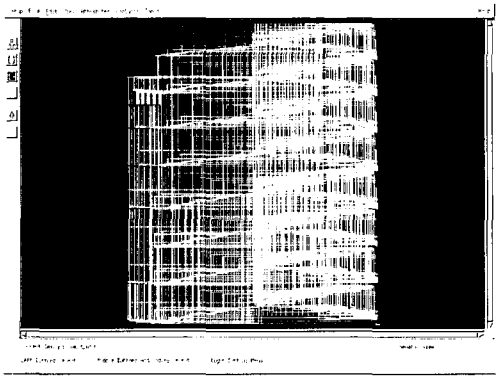


그림 12. 합성된 8x1 1D-IDCT
Fig. 12. Synthesized 8x1 1D-IDCT.

합성된 결과는 네부분으로 구성되는데 첫 번째는 buf_pre_coef로 입력단에서 계수 처리단까지 두 번째는 coef_add_post로 부호 확장 덧셈부에서 후처리단까지 세 번째는 rom_con으로 부호 확장 계산값이 저장되어 있는 롬에서부터 부호 확장 덧셈부 전까지 네 번째는 con_blk인 제어단이다. 위의 내용의 블록도는 <그림 11>과 같다.

<그림 12>는 합성된 8x1 1D-IDCT에 대한 논리합성도이다. 합성결과 설계한 IDCT 프로세서는 100㎒에서 동작하며 블록별 합성된 게이트수는 <표 3>에 나타내었다. <표 4>에서 부호 확장처리단은 rom_con 블록으로 전체 하드웨어의 20.6%를 차지한다. 따라서 하드웨어면에서는 전체 구조의 20.6%가 증가를 보였으나, 연산 속도에 있어서 2배의 속도 향상을 볼 수 있다.

표 4. 합성된 8x1 1D-IDCT의 게이트 수
Table 4. Gate count of synthesized 8x1 1D-IDCT block.

	buf_pre_coef	coef_add_post	rom_con	con_blk	Total
Gate count	7644	6136	3620.8	104	17504.8

<표 5>는 기존의 발표된 논문과 본 논문에서 구현한 구조의 비교를 나타낸 것이다. [1][2]는 병렬처리방식으로써 입력 데이터 비트를 병렬로 처리한다. [3]은 digit-serial 방식으로 클럭당 다중 비트 처리하는 방식이다. 제안한 구조는 bit-serial 방식에 DA 구조를 적용하였다. bit-serial 방식을 사용함으로써 하드웨어의 크기를 줄이고 DA 구조 적용함으로써 연산속도를 빠르게 하기 위함이다. 그러나 설계한 IDCT 프로세서의 합성결과가 기존 구조와의 하드웨어를 비교할 때 크게 차이가 나지 않는 것은 bit-serial 방식을 적용으로 처리 속도를 향상시키기 위해서 제안한 부호 확장 처리단에이 전체 구조의 20.6%를 차지하였기 때문이다. 부호 확장 처리단을 배제할 경우 전체 트랜지스터 수는 50K로 타구조에 비해 감소되는 것을 볼 수 있다.

부호 확장처리단은 제안한 구조가 bit-serial 방식으로 클럭당 1비트씩을 처리하기 때문에 부호 확장을 고려할때 입력데이터가 처리되는 시간은 28클럭으로 느린 처리율을 향상시키기 위해 제안한 구조이다. 입력 데이터가 처리되는 시간을 28클럭으로 계산할 경우 처리율

은 24Mpixels/s가 된다(전처리단과 후처리단에서 소요되는 클럭을 포함할 경우 33클럭이됨). 따라서 낮은 처리속도를 보상해주기 위한 방법으로 부호 확장계산 방식을 적용하여 28클럭을 14클럭으로 줄임으로써 처리율을 50Mpixels/s로 증가시킬 수 있었다.

표 5. 제안한 방법과 기존 방법의 성능 비교.
Table 5. Performance comparison between conventional and proposed method.

	Molloy [1]	Katayama [2]	Rambaldi [3]	Chang [11]	Guo [12]	Proposed
Function	DCT/ IDCT	DCT/ IDCT	IDCT	DCT/ IDCT	IDCT	IDCT
Technology	1.2㎛ CMOS	0.35㎛ CMOS	0.5㎛ CMOS	0.35㎛ IP4M	0.35㎛ CMOS	0.6㎛ CMOS
# of TR	110K	70.7K	65K	119K	36K	70K
Throughput (Mpixels/s)	25	27	27		66	50
Clock rate	50㎒	54㎒	27㎒	100㎒	66㎒	100㎒

V. 결 론

본 논문에서는 bit-serial 방식에 DA 방식을 적용하여 계수식의 변형을 통해 하드웨어의 면적을 줄일 수 있는 IDCT 프로세서 구조를 제안하였다.

또한 기존의 부호 확장비트 처리에서는 12비트(입력 데이터 비트:12클럭)에 16비트(부호 확장처리 비트:16클럭)가 더해진 28비트(28클럭)가 처리되어야 하나, 제안한 부호 확장계산 방법을 사용하여 12비트(입력 데이터 비트:12클럭)에 부호 확장처리단에서 미리 계산된 값과 계수 처리단에서 계산된 값을 더함으로써 IDCT 연산의 처리율을 향상 시켰다. 성능비교에서 제안한 구조가 클럭당 처리되는 비트수가 적음에도 불구하고 다른 논문과 비슷한 성능을 보인 것은 부호 확장 계산 방식을 적용하였기 때문이다. 합성결과에서 8x1 1D-IDCT는 7만개의 트랜지스터가 소요되었으며 이중 1만 4천개의 트랜지스터가 계산 속도 향상을 위한 부호 확장 처리단에서 사용되었는데, 이것은 전체 시스템의 20.6%를 차지한다. 따라서 하드웨어 소모면에서 볼 때 부호 확장 처리단을 제외한다면 타 구조와 비교해서 우수하다는 것을 알 수 있다.

참 고 문 헌

[1] Stephen, Molloy; Rajeev, Jain. "A 110-K Transistor 25-MPixels/s Configurable Image Transform Processor Unit", IEEE J. of Solid-State Circuits, Vol. 33, No. 1, Jan, 1997.

[2] Y, Katayama; T, Kitsuki; Y, Ooi. "A block processing unit in a single-chip MPEG-2 video encoder LSI", in Proc. IEEE Workshop Signal Processing Systems, pp. 459-468, 1997.

[3] R, Rambaldi; A, Ugazzoni; R, Guerrieri, "A 35uW 1.1V gate array 8x8 IDCT processor for video-telephony", Proc. IEEE ICASSP, Vol. 5, pp. 2993-2996, 1998..

[4] T. S. Chang, C.S. Kung, C. W. Jen, "A Simple Processor Core Design for DCT/IDCT", IEEE Transactions on Circuits and Systems for Video Technology, Vol. 10, No. 3, pp. 439-447, April, 2000.

[5] 이철동, 정순기 "ROM 방식의 곱셈기를 이용한 8X8 2차원 DCT의 구현", 대한전자공학회논문지, Vol. 33-A, No. 11, pp. 152-161, 11, 1996

[6] T. S. Chang, C.S. Kung, C. W. Jen, "New distributed arithmetic algorithm and its application to IDCT", Circuits and Systems for Video Technology, IEEE Transactions on circuit device Syst, Vol. 146, No. 4, Aug, 1999.

[7] Wendl Pan; Shams, A.; Bayoumi, M.A. "NEDA: a new distributed arithmetic architecture and its application to one dimensional discrete cosine transform", Signal Processing Systems, 1999. SiPS 99. 1999 IEEE Workshop on, pp. 159-168, 1999.

[8] Nam Ik Cho; San Uk Lee, "Fast algorithm and implementation of 2-D discrete cosine transform", IEEE Transactions on Circuits and Systems, Vol. 38, No. 3, pp. 297-305, March, 1991.

[9] Kyeounsoo Kim; Jong-Seog Koh, "An area efficient DCT architecture for MPEG-2 video

encoder", IEEE Transactions on Consumer Electronics, Vol. 45, No. 1, pp. 62-67, Feb, 1999.

[10] Tian-Sheuan Chang; Jiun-In Guo; Chein-Wei Jen, "A compact IDCT processor for HDTV applications", Signal Processing Systems, 1999. SiPS 99. 1999 IEEE Workshop on, pp. 151-158, 1999.

[11] Jen-Shiun Chiang; Yi-Fang Chiu; Teng-Hung Chang, "A high throughput 2-dimensional DCT /IDCT architecture for real-time image and video system", Electronics, Circuits and Systems, 2001. ICECS 2001. The 8th IEEE International Conference on , Volume: 2, 2-5 Sept. 2001 Page(s): 867 -870 vol.2.

[12] Jiun-In Guo, "A low cost 2-D inverse discrete cosine transform design for image compression", Circuits and Systems, 2001. ISCAS 2001. The 2001 IEEE International Symposium on, Volume:4, 6-9 May 2001 Page(s): 658-661 vol. 4

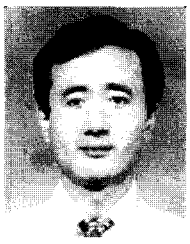
저 자 소 개

金 炳 民(正會員)

2000년 2월 : 충북대학교 전자공학과 학사. 2002년 2월 : 충북대학교 전자공학과 석사. 2002년 1월~현재 : (주) 라이온텍 기술연구소 전임 연구원. <주관심분야 : 저전력 회로설계>

裴 絃 憙(正會員)

1977년 2월 : 한양대학교 전자공학과 공학사. 1980년 2월 : 서울대학교 전자공학과 공학석사. 1992년 2월 : 서울대학교 전자공학과 공학박사. 1983년~1987년 : 관동대학교 전자공학과 조교수. 1994년~1995년 : Syracuse Univ., 방문교수. 1987년~현재 : 충북대학교 전자공학과 교수. <주관심분야 : 적응신호처리, 다차원 신호처리, 웨이브렛 변환의 신호처리 응용>



趙 泰 元(正會員)

1973년 2월 : 서울대학교 전기공학과 학사. 1973년~1984년 : 금성전선(주). 1986년 5월 : 미국 루이빌대학교 전자공학과 석사. 1992년 5월 : 미국 켄터키주립대 전자공학과 박사. 1992년 3월~현재 : 충북대학교 전기전자공학부 교수. <주관심분야 : 마이크로프로세서 설계, 집적회로 설계, Routing, 저전력 회로 설계>