論文 98-23-1-25

정규직교 이산웨이브렛을 위한 효율적인 VLSI 구조

正會員 장 시 중*, 김 대 용**, 김 순 영***, 이 문 호*

A Efficient VLSI Architecture for Orthonormal Discrete Wavelet Transform

Sijung Chang*, Daeyoung Kim**, Soonyung Kim***, Moon Ho Lee* *Regular Members*

※본 연구는 서울대학교 반도체공동연구소의 교육부 반도체 분야 학술연구 조성비(과제번호 : ISRC 96-E-2007)에 의해 수행되었습니다.

요 약

정규직교 웨이브렛에 사용되는 고대역 필터와 저대역 필터의 계수는 $g_{(n)}=(-1)^n h_{(L-n)}$ 같은 관계를 가지고 있다. 여기서 $h_{(n)}$, $g_{(n)}$은 각각 저대역 필터와 고대역 필터의 계수를, L은 필터의 차수를 의미한다. 일반적으로 이산 웨이브렛 변환과 역 이산 웨이브렛 변환을 직접적으로 구현할 시 필요한 곱셈기와 덧셈기의 수는 각각 $4(L+1)$, $4L$이 된다. 본 논문에서는 필터의 계수간의 연관성을 이용하여 필요한 곱셈의 양을 줄이는 방법을 연구하였고 이를 적용하면 필요한 곱셈기의 수를 $2(L+2)$로 줄일 수 있고, 덧셈기수는 $2(((L+1)\times 2)-1)$가 필요하다. 새로 제안한 분석 및 합성의 구조는 gate level 설계 툴인 QuickLogic을 이용해 시뮬레이션 하여 검증하였다. 기존 구조에 비해 새로 제안한 구조는 L이 3차인 경우 곱셈기의 수를 6개정도 감소시킬 수 있고 수행속도의 저하 없이 하드웨어 구조를 단순하게 구성할 수 있다.

ABSTRACT

In the orthonormal Discrete Wavelet Transform(DWT), the coefficients of highpass filter and lowpass filter are related to $g_{(n)}=(-1)^n h_{(L-n)}$, where $h_{(n)}$ is a coefficient of lowpass filter, $g_{(n)}$ is a coefficient of highpass filter and L is a degree of filter. Conventional architectures for DWT and Inverse DWT(IDWT) is composed of $4(L+1)$ multipliers and $4L$ adders [folded architecture]. In this paper, we study the method of that required multiplications are reduced with using the relation of filter coefficients. The $2(L+2)$ multipliers and $2((L+1)*2-1)$ adders are

*전북대학교 정보통신공학과
**전북대학교 영상정보공학과
***전북대학교 컴퓨터공학과
論文番號:97361-1009
接受日字:1997年 10月 9日

required in proposed architecture. We can obtain correct operation of proposed architecture though QuickLogic simulation. The required multipliers in proposed architecture can be reduced by 6 compared with conventional architecture. This means that circuit is low hardware complexity and can be operated with low power consumption.

Ⅰ. 서 론

웨이브렛 변환에 대한 연구가 발표되면서 이산 웨이브렛 변환은 Mallat 및 Daubechies에 의해 기존의 이산신호처리방법과 관련이 있음이 밝혀졌으며 이를 바탕으로 M. Vetterli등에 의해 대역분할 코딩에서 사용되는 필터 군(filter banks)과의 유사함이 밝혀졌다[2][4][5][7]. 이후 웨이브렛 변환의 실용적, 이론적인 노력이 있었으며 연구는 빠르게 성장하고 있다[10-15]. 웨이브렛 변환은 신호를 웨이브렛 기저 함수의 집합으로 분해(decomposition)하며 웨이브렛 기저함수는 모 웨이브렛 및 scale함수를 확대/축소(dilation) 그리고 천이 시켜 얻을 수 있다[1]. 모 웨이브렛을 필터 측면에서 해석하면 대역통과(band pass)필터로 스케일링(scaling) 함수는 저대역 통과(lowpass)필터로 생각할 수 있으며 대역폭의 변화는 모 웨이브렛 및 스케일링함수의 스케일링 값을 변화시킴으로써 변화시킬 수 있다. 그러므로 웨이브렛 변환에서 스케일은 주파수의 다른 개념으로 생각될 수 있다. 즉, 웨이브렛 변환은 신호를 공간적이고 스펙트럼 적으로 국부화된 기저의 집합으로 분석할 수 있도록 한다. 이산 웨이브렛 변환은 신호를 여러 주파수 대역으로 분할하며 역 이산 웨이브렛 변환에 의해 분할된 신호는 다시 합성된다[4][5][7]. Mallat에 의해 제안된 피라미드 알고리즘은 이산 웨이브렛의 효율적인 계산을 제공하였다[4]. 이 피라미드 알고리즘은 1 옥타브의 연산 사이사이에 상위 옥타브의 연산을 하므로써 실시간 처리를 가능케 하는 방식이다. 많은 논문에서 이러한 방식을 사용하여 이산 웨이브렛 변환기의 구조를 제안하였다[16][17][19-24][28][29]. Knowles의 논문에서 피라미드 알고리즘을 사용한 구조를 처음으로 제안하였다[16]. 그후 이산 웨이브렛 변환의 설계에 대한 논문들은 주로 중간 결과 값을 저장하고 route시키는 방법에 대한 논문이 주를 이루었다. 시스토릭 라우팅 네트워크[22], RAM 기반의 구조[22], 최소의 레지스터를 사용한 구조[19], 브로드 캐스팅방법을 이용한 구조[29]등이 이에 속하는 것들이다. 이러한 것들은 구조적인 측면에서 장점을 내 세웠을 뿐 연산상의 이득을 이끌어내는 구조는 아니었다.

본 논문은 정규직교 이산웨이브렛 변환을 위한 효율적인 VLSI 구조에 관한 것이다. 정규직교 이산 웨이브렛 변환을 위해 사용하는 고대역 필터와 저대역 필터의 계수간의 연관성을 이용하여 연산상에서의 계산량을 줄이는 방법 및 실시간 처리가 가능한 정규직교 이산웨이브렛 변환을 수행하는 구조를 제안하였다. 이해의 편의를 위해 각각의 레벨에서의 연산을 도식적으로 살펴보고, Dyadic 웨이브렛 구조에 사용하여 각 레벨에서의 출력을 구하였다. 분석의 구조와 더불어 수신 단에서 신호를 복원하는 합성의 구조도 제안하였다. 본 논문의 구성은 다음과 같다. Ⅱ장에서는 이산 웨이브렛 구조를 이해하는데 필요한 일반적인 특성들에 대해 살펴보고 Ⅲ장에서는 정규직교 웨이브렛에서 분석 및 합성의 계산량 감소방법을 제시하고 이를 이용한 Discrete Wavelet Transform(DWT) 구현 구조를 제안했다. Ⅳ장은 제안구조의 올바른 동작의 검증을 위해 게이트 레벨 설계 툴인 QuickLogic을 이용하여 설계하고 이를 이용한 결과를 분석한다. 마지막으로 Ⅴ장에서는 결론을 맺는다.

Ⅱ. 이산 웨이브렛 연산의 특성

〈컨볼루션 연산〉

이산 웨이브렛 변환의 연산은 웨이브렛 필터에 의한 필터링을 의미하므로 이산 웨이브렛 변환 구조에서는 효율적인 필터링이 매우 중요하다. 필터링은 필터계수와 신호값과의 컨볼루션을 의미한다. 필터의 그림 1에서 볼 수 있듯이 신호를 따라 이동하며 겹치는 부분의 신호와 연산을 하여 출력값을 발생하도록 한다. 각 스텝에서 필터 계수와 신호의 값이 곱해지고 그 후 필터의 계수와 곱해진 값들이 전부 더해져서 필터를 통과한 하나의 값이 만들어진다[28].

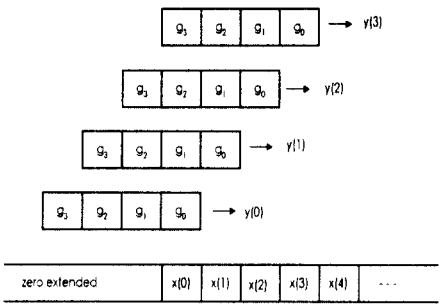
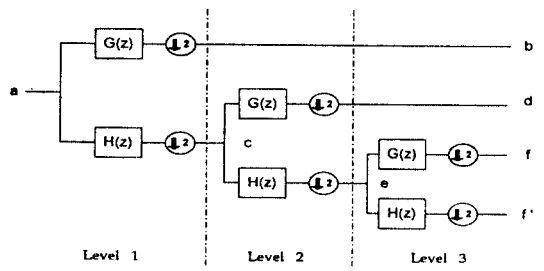


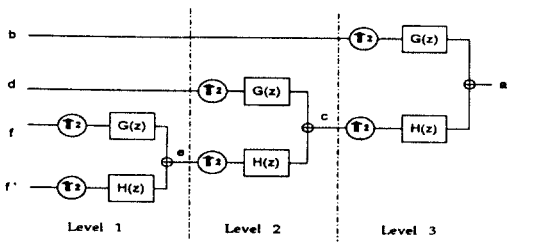
그림 1. 신호의 필터링

〈Dyadic 구조〉

그림 2에 보이는 Dyadic(옥타브-밴드)구조를 사용하여 DWT를 구현하고자 한다. Dyadic 구조는 먼저 고대역을 통과한 b에서 전체 시퀀스의 1/2이 나오고 저대역 필터를 통과한 c는 다음 옥타브 연산을 위해 입력으로 들어간다. 이러한 방식으로 저대역을 계속하여 나누어 가는 구조이다. 연산은 웨이브렛으로부터 유도된 두 개의 필터, 저대역 필터 H와 고대역 필터 G에 의해 수행된다. 설계시 고려한 형태는 Daubechies의 4-tap 필터를 사용하고 옥타브 수는 3을 선택하였다. 즉, L=3, J=3, N=8.



(a) Analysis



(b) Synthesis

그림 2. 웨이브렛 분석과 합성(옥타브 수=3)

입력으로 a에서 N개를 취해서 N개를 출력한다. 출력의 결과를 살펴보면, 가장 높은 해상도에서 N/2개, 그 다음 해상도에서는 N/4개, …, 즉, 주파수가 높으면 주파수 해상도가 낮고 낮은 주파수에서는 주파수 해상도가 높다. 반면에 높은 주파수에서는 시간 해상도가 높으며 낮은 주파수에서는 시간 해상도가 낮다. N점 이산 웨이브렛 변환의 옥타브 수는 J이며 옥타브 J는 $\log_2 N$에 의해 얻어진다. octave 1은 N/2출력을 octave 2는 N/4개의 출력을, octave J는 $N/2^J$개의 출력을 갖는다. 예로 J=3인 경우(N=8), octave 1은 8/2개를 octave-2는 8/4개, octave-3은 8/8개의 출력을 내보낸다. 그림 2에서 H와 G의 전달함수를 각각 다음과 같이 가정한다.

$$G(z)=g_0+g_1z^{-1}+g_2z^{-2}+g_3z^{-3}$$
$$H(z)=h_0+h_1z^{-1}+h_2z^{-2}+h_3z^{-3}$$

〈피라미드 알고리즘〉

피라미드 알고리즘은 이산 웨이브렛의 실시간 구현을 위해 사용되는 방식이다[4]. 그림 3은 피라미드 알고리즘을 도식적인 표현으로 나타내고 있다. 가로축은 시간의 흐름을 나타내고 세로축은 옥타브를 나타내고 있다. 검은 블록은 연산을 의미한다. 그림을 통해 첫 번째 옥타브 연산 사이사이에 상위 옥타브의 연산을 수행하는 방법임을 알 수 있다.

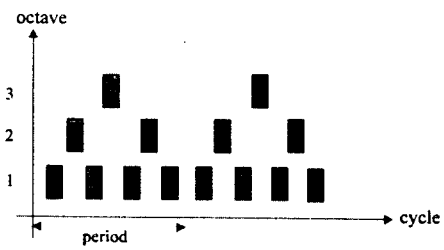


그림 3. 피라미드 알고리즘의 도식적 표현

Ⅲ. 정규직교 웨이브렛 변환을 위한 새로운 구조의 제안

3.1 분석(Analysis)의 구조

모델로 삼은 이산 웨이브렛 변환의 옥타브가 3이기

때문에(그림 2) 연산은 주기 8로 주기적이다. 즉, 계산은 시간 참조(index) 8로 분리된다. 일반적으로 계산은 $N=2^l$로 주기적이다. 그림 2(a)에서 octave-1의 고대역 필터를 통과한 값 b는 다음과 같이 연산된다.

$$\begin{aligned} b(0) &= g_0a(0) + g_1a(-1) + g_2a(-2) + g_3a(-3) \\ b(2) &= g_0a(2) + g_1a(1) + g_2a(0) + g_3a(-1) \\ b(4) &= g_0a(4) + g_1a(3) + g_2a(2) + g_3a(1) \\ b(6) &= g_0a(6) + g_1a(5) + g_2a(4) + g_3a(3) \end{aligned} \quad (3.1)$$

마찬가지로 저대역 필터를 통과한 값, c는 다음과 같다.

$$\begin{aligned} c(0) &= h_0a(0) + h_1a(-1) + h_2a(-2) + h_3a(-3) \\ c(2) &= h_0a(2) + h_1a(1) + h_2a(0) + h_3a(-1) \\ c(4) &= h_0a(4) + h_1a(3) + h_2a(2) + h_3a(1) \\ c(6) &= h_0a(6) + h_1a(5) + h_2a(4) + h_3a(3) \end{aligned} \quad (3.2)$$

여기에서 c의 값은 옥타브 2의 연산을 위해 입력으로 들어와 필터링을 거치게 된다. octave 2에서의 고대역 필터링된 값 d와 저대역 필터의 통과값 e는 다음과 같다.

$$\begin{aligned} d(0) &= g_0c(0) + g_1c(-2) + g_2c(-4) + g_3c(-6) \\ d(4) &= g_0c(4) + g_1c(2) + g_2c(0) + g_3c(-2) \\ e(0) &= h_0c(0) + h_1c(-2) + h_2c(-4) + h_3c(-6) \\ e(4) &= h_0c(4) + h_1c(2) + h_2c(0) + n_3c(-2) \end{aligned}$$

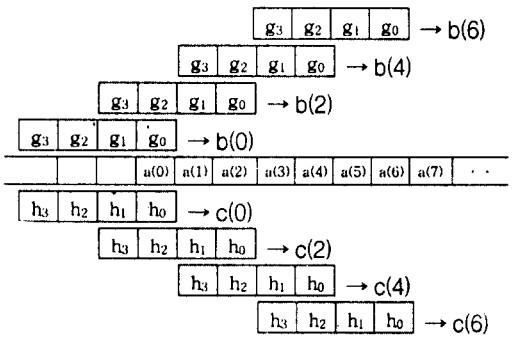
역시 e의 값은 octave-3의 연산을 위해 입력으로 들어가서 계산이 된다. octave-3의 고역 필터링된 값과 저역 필터링된 값은 다음과 같다.

$$\begin{aligned} f(0) &= g_0e(0) + g_1e(-4) + g_2e(-8) + g_3e(-12) \\ f(0) &= h_0e(0) + h_1e(-4) + h_2e(-8) + h_3e(-12) \end{aligned}$$

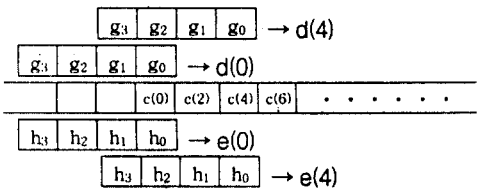
여기에서 출력으로 나가는 부분은 각 옥타브의 고역 필터링된 값들과 마지막 옥타브의 저역 필터링된 값이다. 출력되는 값은 총 8개이며 그 값은 다음과 같다.

b(0), b(2), b(4), b(6), d(0), d(4), f(0), f'(0)

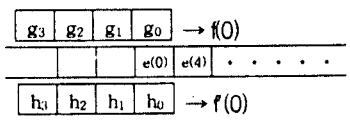
시간 참조는 연산가운데에서 한 주기로 볼 수 있으므로 음수인 것은 값이 전에 들어온 입력으로 생각할 수 있다. 위의 값들에 주기의 정수배를 더해서 전체 계산을 구할 수 있다. 그림 4는 각각의 옥타브에서의 연산을 도식적으로 표현한 것이다. 여기에서는 다운샘플링(downsampling)에 의해 각 스텝마다 두 칸씩 이동한다. 그림 4의 (a)는 옥타브 1에서의 결과 b와 c이며 그림 4의 (b)는 옥타브 d와 e, 그림 4(c)는 옥타브 3의 f와 f'이다.



(a) Octave 1



(b) Octave 2



(c) Octave 3

그림 4. 각 옥타브에서의 연산

〈분석에서의 곱셈 계산량 감소 방법〉

여기에서 분석에 이용되는 정규직교 웨이브렛의 고대역 필터와 저대역 필터간의 관계는 다음과 같다.

$$g_n=(-1)^n h_{(L-n)} \qquad \text{단, } L\text{은 필터의 차수} \qquad (3.3)$$

식(3.3)을 적용하여 b(0)과 c(0)의 계산식을 다시 정리하면 다음과 같다.

$$c(0)=h_0a(0)+h_1a(-1)+h_2a(-2)+h_3a(-3)$$
$$b(0)=h_3a(0)-h_2a(-1)+h_1a(-2)-h_0a(-3)$$

각각의 식을 두 부분으로 묶으면

$$c(0)=\underbrace{\{h_0a(0)+h_3a(-3)\}}_{①}+\underbrace{\{h_2a(-2)+h_1a(-1)\}}_{②}$$
$$b(0)=\underbrace{\{h_3a(0)-h_0a(-3)\}}_{③}+\underbrace{\{h_1a(-2)-h_2a(-1)\}}_{④}$$

①, ③식은 계속해서 다음과 같이 바꾸어 쓸 수 있다.

$$①=a(-3)(h_0+h_3)+(a(0)-a(-3))h_0=h_0a(0)+h_3a(-3)$$
$$③=(a(0)-a(-3))h_0+(h_3-h_0)a(0)=h_3a(0)-h_0a(-3)$$

②, ④식도 마찬가지 방식으로 다음과 같이 표현 가능하다.

$$②=a(-1)(h_1+h_2)+(a(-2)-a(-1))h_2=h_1a(-1)+h_2a(-2)$$
$$④=(a(-2)-a(-1))h_2+(h_1-h_2)a(-2)=h_1a(-2)-h_2a(-1)$$

따라서 $c(0)$, $b(0)$를 다음과 같이 다시 쓸 수 있다.

$$c(0)=(h_0+h_3)a(-3)+(h_1+h_2)a(-1)+\underbrace{h_0(a(0)-a(-3))+h_2(a(-2)-a(-1))}_{Ⓐ} \qquad (3.4)$$

$$b(0)=(h_3-h_0)a(0)+(h_1-h_2)a(-2)+\underbrace{h_0(a(0)-a(-3))+h_2(a(-2)-a(-1))}_{Ⓑ} \qquad (3.5)$$

식(3.4)(3.5)을 보면 항과 항이 같음을 알 수 있다. 이는 4번의 곱셈과 2번의 덧셈이 절반으로 줄어듬을 의미한다. 결과적으로 식(3.1)(3.2)에서 직접적으로 $c(0)$, $b(0)$를 구하려면 8번의 곱셈과 6번의 덧셈이 요구되지만 식(3.4)(3.5)을 이용하게 되면 6번의 곱셈과 7번의 덧셈을 통해 $c(0)$, $b(0)$를 구할 수 있게된다.

식(3.4)(3.5)을 일반화하여 표현하면 다음과 같다.

$$c(n)=(h_0+h_3)a(n-3)+(h_1+h_2)a(n-1)+h_0(a(n)-a(n-3))+h_2(a(n-2)-a(n-1)) \qquad (3.6)$$

$$b(n)=(h_3-h_0)a(n)+(h_1-h_2)a(n-2)+h_0(a(n)-a(n-3))+h_2(a(n-2)-a(n-1)) \qquad (3.7)$$

여기에 실제적으로 사용하는 Daubechies 4-tap 필터계수의 특성을 이용하면 더 효율적인 구조를 얻을 수 있다. 다음은 Daubechies 4-tap필터의 계수를 나타낸 것이다.

$$h_0=(1+\sqrt{3})/4\sqrt{2}$$
$$h_1=(3+\sqrt{3})/4\sqrt{2}$$
$$h_2=(3-\sqrt{3})/4\sqrt{2}$$
$$h_3=(1-\sqrt{3})/4\sqrt{2} \qquad (3.8)$$

필터 계수가 식(3.8)과 같음을 이용하면

$$h_1-h_2=\frac{2\sqrt{3}}{4\sqrt{2}}$$
$$h_3-h_0=-\frac{2\sqrt{3}}{4\sqrt{2}}=-(h_1-h_2)$$

과 같이 되어 위의 관계를 이용하여 다시 정리하면 식(3.4) (3.5)를 식 (3.9) (3.10)와 같이 쓸 수 있게 된다.

$$c(0)=(h_0+h_3)a(-3)+(h_1+h_2)a(-1)+h_0(a(0)-a(-3))+h_2(a(-2)-a(-1)) \qquad (3.9)$$

$$b(0)=(h_1-h_2)a(-2)-a(0))+h_0(a(0)-a(-3))+h_2(a(-2)-a(-1)) \qquad (3.10)$$

또한 식(3.9) (3.10)를 일반화하여 표현하면 다음과 같다.

$$c(n)=(h_0+h_3)a(n-3)+(h_1+h_2)a(n-1)+h_0(a(n)-a(n-3))+h_2(a(n-2)-a(n-1)) \qquad (3.11)$$

$$b(n)=(h_1-h_2)(a(n-2)-a(n))+h_0(a(n)-a(n-3))+h_2(a(n-2)-a(n-1)) \qquad (3.12)$$

고대역 및 저대역 필터의 출력을 얻는데 이제는 5번의 곱셈과 7번의 덧셈으로 충분하다는 것을 전개한 식에 의해 이해할 수 있다.

표 1. 입력에 따른 입력과 출력의 데이터 흐름표

사이클	입력	필터로 들어가는 값				HP	LP	출력
		n	n-1	n-2	n-3			
0	a(0)	a(0)	a(-1)	a(-2)	a(-3)	b(0)	c(0)	b(0)
1	a(1)	c(0)	c(-2)	c(-4)	c(-6)	d(0)	e(0)	d(0)
2	a(2)	a(2)	a(1)	a(0)	a(-1)	b(2)	c(2)	b(2)
3	a(3)	e(0)	e(-4)	e(-8)	e(-12)	f(0)	f'(0)	f(0)
4	a(4)	a(4)	a(3)	a(2)	a(1)	b(4)	c(4)	b(4)
5	a(5)	c(4)	c(2)	c(0)	c(-2)	d(4)	e(4)	d(4)
6	a(6)	a(6)	a(5)	a(4)	a(3)	b(6)	c(6)	b(6)
7	a(7)							f'(0)
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

한 주기에 대해 입력값과 이에 따른 고대역 및 저대역 필터의 출력값을 표 1에 나타냈다. 먼저 a(0)가 들어오면 b(0)와 c(0)를 계산하기 위해 필터로 a(0), a(-1), a(-2), a(-3)이 보내지고 고대역 필터에서는 b(0)이 출력되고 저대역 필터에서는 c(0)이 출력된다. a(1)이 들어오면 Octave 2에서 d(0), e(0)를 계산하기 위해 필터에 c(0), c(-2), c(-4), c(-6)를 보낸다. 즉 입력이 짝수 인덱스를 가진 값이면 1레벨의 연산이 수행되고 홀수 인덱스를 가진 값이 입력되면 2, 3레벨의 연산이 수행된다. 그러므로써 동시에 옥타브 1과 상위옥타브의 연산을 동시에 계산할 수 있는 것이다. 이는 피라미드 알고리즘을 적용한 것이다. 결과적으로 이번 절에서 제안한 방식에서 필요한 곱셈기가 5개로 줄어들게 되고 반면에 필요한 덧셈기는 하나가 늘어 7개가 됨을 알 수 있다. 하드웨어 구현에 있어 덧셈기에 대한 곱셈기의 복잡도는 몇배가 되므로 이러한 곱셈기의 감소를 통해 상당량의 하드웨어를 감소시킬 수 있다.

〈분석을 위한 제안구조의 구성도〉

분석에서의 전체적인 구조는 그림 5와 같다. 구성은 크게 메모리 블록 부분, 필터링 하는 부분 등으로 구성 되어 있다. 메모리 블록은 필터링블럭의 표 1에서 보여준 스케쥴링에 맞게 입력값을 필터링 블록에 내보내는 부분이고 필터링블럭은 실제적으로 고대역 필터링값과 저대역 필터링 값을 계산하는 부분이다. 필터링을 거친 값은 고대역필터를 통과한 값과 저대역 필터를 통과한 두 개의 출력이 나온다. 고대역 필터를 통과한 값을 그대로 출력으로 향하고 저대역 필터를 통과한 값중 c와 e에 해당하는 값은 다음 octave 연산을 위해 메모리 블록으로 재입력이 된다. 그리고 마지막 옥타브의 저대역필터를 통과한 값은 delay에 저장되었있다가 8l+7 번째에 출력으로 나간다. 그림 6은 필터링하는 부분의 상세도이며, 그림 7은 메모리 블록도이다. 필터링을 수행하는 부분은 곱셈기와 덧셈기로 구성이 되어있다. 4-tap 필터를 사용한다면 분석의 구조에서 필요한 곱셈기의 수는 5개이고 덧셈기는 7개가 된다. 메모리 블록의 크기는 필터의 Tap수, 계산할 Octave의 수와 연관이 있다. 필터로 들어가는 값은 메모리 블록에서 필터의 Tap수 만큼 병렬로 나온다. 그리고 각 Octave의 계산을 위한 입력값을 저장할 장소가 각각 따로 필요하다. 분석에서의 메모리 블록의 크기는 $J*(L+1)-1$, J는 Octave수 L은 필터의 차수 이다. 식에서 -1은 첫 번째 입력에서 곧바로 필터쪽으로 들어가기 때문이다. 표 1에서 스케쥴링을 보듯이 level 1의 입력은 2l 마다, level 2의 입력은 4l+1, level 3의 입력은 8l+3마다 필터링 블록쪽으로 값을 내보낸다. 이러한 방식은 하나의 필터링 블록을 가지고 모든 Octave연산을 처리하는 Folded구조이

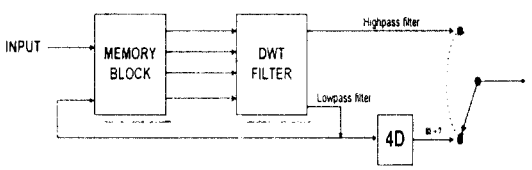


그림 5. 제안한 분석의 블록도

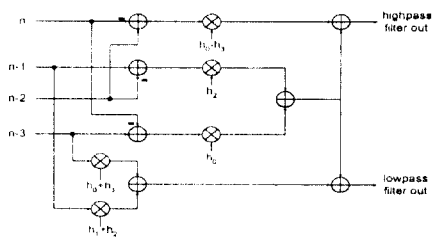


그림 6. 분석에서의 필터링 구조

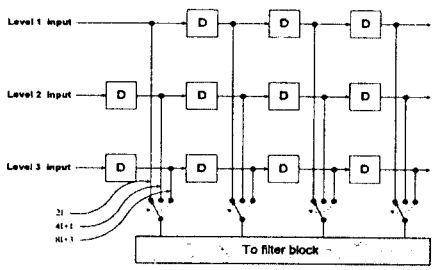


그림 7. 분석에서의 메모리 구조

다. Folded구조의 장점으로는 하드웨어를 반복 사용함으로써 효율성을 높여 구현한다는 장점이 있으나 Data path의 폭을 오버플로어가 발생하지 않도록 최대로 설정하거나 연산후 일정한 비트로 양자화해야 하는 단점을 가지고 있다[9].

3.2 합성(synthesis)의 구조

합성은 그림 2(b)에서 보듯이 analysis와는 달리 up-sampling을 먼저 수행한다. 그림 8은 0값을 삽입하는 up-sampling연산을 보여준다. 그림에서 어두운 부분이 0이 삽입되는 부분이다.

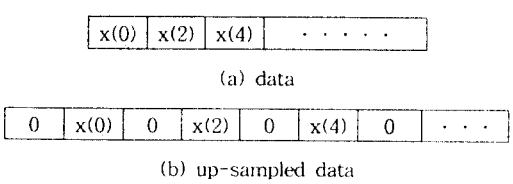
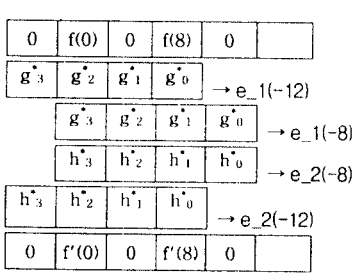


그림 8. Up-Sampling



e_1(-12) + e_2(-12) = e(-12), e_1(-8) + e_2(-8) = e(-8)

그림 9. 역이산웨이브렛 변환의 옥타브 1의 수행

그림 2(b)를 보면, 옥타브 1에서의 입력 f와 f 이 각각 up-sampling한 후 각각의 필터를 거친 후 더해져서 e를 만들고, e와 d가 또한 각각 up-sampling을 한 후 각각의 필터를 거친 후 더해져서 c를 형성하고 b와 c도 같은 방법으로 해서 원 데이터 값인 a를 복원한다. 그림 9는 그림 4와 같이 역 이산 웨이블렛 변환의 옥타브 1의 연산이다. 옥타브 2와 옥타브 3 또한 같은 방법으로 구해진다.

〈합성에서의 곱셈 계산 감소방법〉

여기에서 합성에 이용되는 정규직교 웨이브렛의 고대역 필터와 저대역 필터를 분석에서 사용하는 필터를 이용하여 표시하면 다음과 같은 관계가 된다.

$$h_0^* = h_3,\quad h_1^* = h_2,\quad h_2^* = h_1,\quad h_3^* = h_0$$
$$g_0^* = -h_0,\quad g_1^* = h_1,\quad g_2^* = -h_2,\quad g_3^* = h_3$$

분석에서 사용했던 유사한 방법으로 전개식에서 공통적인 부분을 뽑아내어 공유하는 방법을 사용하여 전개하면 다음과 같다. 점선안에 있는 식은 서로 공유되는 부분이다.

$$\begin{aligned} e(-12) &= -h_0 f(8) - h_2 f(0) + h_3 f'(8) + h_1 f'(0) \\ &= f'(0)\,[h_1 - h_0] + \quad [f'(0) - f(8)] \cdot h_0 \\ &\quad + f'(8)\,[h_3 - h_2] + [f'(8) - f(0)] \cdot h_2 \\ e(-8) &= h_1 f(8) + h_3 f(0) + h_2 f'(8) + h_0 f'(0) \\ &= f(8)\,[h_1 + h_0] + \quad [f'(0) - f(8)] \cdot h_0 \\ &\quad + f(0)\,[h_3 + h_2] + [f'(8) - f(0)] \cdot h_2 \end{aligned}$$

필터 계수간의 관계를 살펴보면 다음과 같은 관계가 있다.

$$h_1 - h_0 = \frac{3+\sqrt{3}-1-\sqrt{3}}{4\sqrt{2}} = \frac{2}{4\sqrt{2}}$$
$$h_3 - h_2 = \frac{1-\sqrt{3}-3+\sqrt{3}}{4\sqrt{2}} = -\frac{2}{4\sqrt{2}} = -(h_1 - h_0)$$

위의 관계식을 합성을 위한 식에 넣어 정리하여 a에 대해 일반화시키면 다음과 같다. e와 c에 대한 식도 비슷하게 유도 할 수 있다.

$$a(2n-1) = (h_1 - h_0)[c(2n) - c(2(n+1))]$$

$$+h_0\{c(2n)-b(2(n+1))$$
$$+h_2(c(2(n+1)))-b(2n)\}$$
$$a(2n)=(h_1+h_0)\,b(2(n+1))+(h_3+h_2)\,b(2n)$$
$$+h_0\{c(2n)-b(2(n+1))\}$$
$$+h_2\{c(2(n+1))-b(2n)\}$$

위에서 제안한 계산방법을 이용한 합성의 입력과 출력에 관한 흐름표는 표 2와 같다. 표 2를 살펴보면 분석에서의 첫 번째 입력값 a(0)의 값은 합성의 구조에서 복원 시작후 31사이클 후에 복원이 되어서 나온다. 즉 합성 구조에서 복원 시작후 처음 나오는 31개의 데이터열의 값은 0의 열이 나와야 한다. 그러므로 알맞은 복원값을 위해서는 입력되는 값은 메모리 블록에 저장하여 해당되는 순서에 맞게 스케쥴링하여 필터링 블록에 입력되어야 한다. 표 2에 의해 계산한 b를 위해 저장할 메모리의 길이는 15가 되며 d의 값을 저장하기위한 메모리의 크기는 6이 된다. f와 f′에 의해 복원되는 값 e와 d에 의해서 복원되는 값 c도 필터링 블록에 입력이 되기 위해 저장이 되어야 하며 각기 2개씩의 메모리를 요구한다. 이에 대한 자세한 그림은 그림 12를 참조하기 바란다.

표 2. 합성에서의 입·출력 흐름표

사이클	입력	필터로 들어가는 값				first	second	출력
		high_1	high_2	low_1	low_2			
0	b(0)							a(-31)
1	d(0)							a(-30)
2	b(2)							a(-29)
3	f(0)							a(-28)
4	b(4)							a(-27)
5	d(4)							a(-26)
6	b(6)							a(-25)
7	f′(0)	f(-8)	f(0)	f′(-8)	f′(0)	e(-12)	e(-8)	a(-24)
8	b(8)	b(-22)	b(-20)	c(-22)	c(-20)	a(-23)	a(-22)	a(-23)
9	d(8)	d(-16)	d(-12)	e(-16)	e(-12)	c(-18)	c(-16)	a(-22)
10	b(10)	b(-20)	b(-18)	c(-20)	c(-18)	a(-21)	a(-20)	a(-21)
11	f(8)							a(-20)
12	b(12)	b(-18)	b(-16)	c(-18)	c(-16)	a(-19)	a(-18)	a(-19)
13	d(12)	d(-12)	d(-8)	e(-12)	e(-8)	c(-14)	c(-8)	a(-18)
14	b(14)	b(-16)	b(-14)	c(-16)	c(-14)	a(-17)	a(-16)	a(-17)
15	f′(8)	f(0)	f(8)	f′(0)	f′(8)	e(-4)	e(0)	a(-16)
16	b(16)	b(-14)	b(-12)	c(-14)	c(-12)	a(-15)	a(-14)	a(-15)
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

〈제안구조의 구성도〉

합성에서의 입력은 분석에서 나온 결과의 순서와 동일하다. 한 사이클에 하나의 데이터가 입력되며 하나의 필터가 전체 옥타브의 연산을 수행한다. 그림 10은 합성의 블럭도이다. 합성의 구조도 분석의 구조와 마찬가지로 메모리 블록과 역 DWT필터 블록등 크게 2개의 블럭으로 구성되어 있다. 메모리 블록은 표 2에서 보여준 것 과 같은 스케쥴링을 통해 필터블럭으로 입력값을 전달한다. 필터를 통과하여 복원한 값은 First, Second 두 개의 값이 동시에 나온다. a의 예를 들면 a(2n-1)값이 First에 해당하고 a(2n)는 Second 값에 해당된다. 표 2에서 나타나 있듯이 짝수 사이클(2l)에서는 a값이 복원되어서 나온다. 이때 First값은 출력으로 나가고 Second 값은 레지스터에 저장이 된다. 홀수 사이클(2l+1)에서는 상위 level에 있는 값이 복원이 된다. 이 복원된 값은 하위 level의 값을 복원시키기 위해 메모리 블록으로 다시 입력이 된다. 이때 레지스터에 저장된 값(짝수 사이클에서의 Second 값)이 출력으로 나가게 된다. 즉 최종 출력값은 분석에서의 입력, a의 데이터 스트림이 나가게 된다. 그림 11은 합성을 위한 필터링 블록도를 나타낸 것이다. 분석의 구조와 마찬가지로 결과적으로 합성에서도 분석과 마찬가지로 곱셈기 5개와 덧셈기 7개가 된다. 그림 12는 메모리 블록을 나타낸 것이다. 2l에서는 a를 복원하기 위해 b와 c의 값을 필터로 보내고 4l+1에서는 d와 e를 필터블럭으로 보내 c를 복원한다. 8l+7에서는 e를 복원하기 위해 f와 f′을 필터블럭에 보낸다.

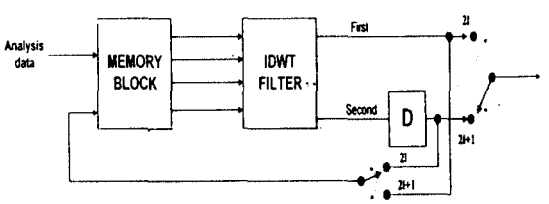


그림 10. 제안한 합성의 블록도

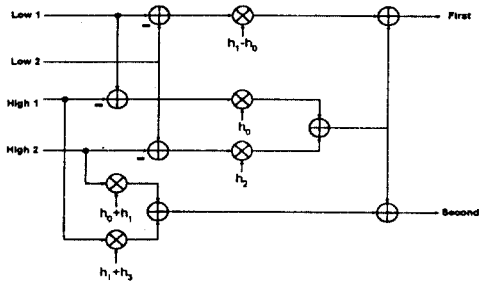


그림 11. 합성에서의 필터링 구조

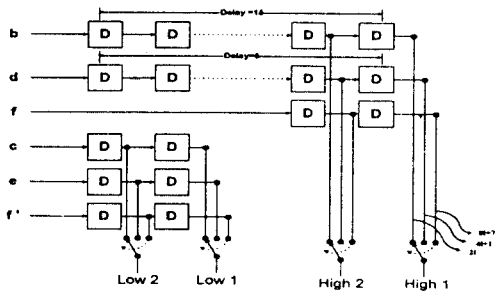


그림 12. 합성에서의 메모리 구조

Ⅳ. 설계 및 결과분석

설계에 사용한 수체계는 음수를 효율적으로 표현하기 위해 2의 보수를 사용하였고 또한 곱셈기 설계시 필터의 계수가 정해져 있으므로 일반적인 곱셈기가 아닌 상수 곱셈기를 사용할 수 있다. 계수의 형태는 기억장소와 전력 소모를 최소화하여 가장 경제적인 디지트 포맷을 제공하는 CSD 형식을 이용하였다. 덧셈기는 carry/borrow의 전달을 가속시키기 위해 Carry Lookup Adder(CLA)를 사용, 각 bit position에 해당하는 carry들을 동시에 생성할 수 있도록 설계했다.

4.1 정확성 분석

십진수와 2진수간의 변환 시에 원래의 값과 차가 존재한다.

그림 13은 DWT에서 정확성 분석에 쓰이는 Mean square error를 구하는 방법을 보여준다. 표 3은 MATLAB의 웨이브렛 툴박스를 이용해 구한 이산 웨이브

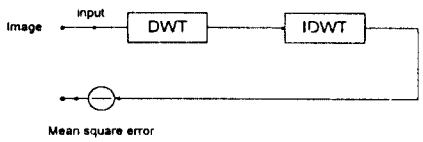


그림 13. 정확도의 분석

렛 필터의 계수의 넓이와 Mapping방법에 따른 Mean square error를 나타낸다. 표 3의 분석에 의하면 Truncation 하는 방식보다 rounding하는 방식이 오차가 더 적음을 알 수 있다. 가장 에러가 적은 쪽은 유동소수점으로 설계하는 것이다. 그러나 이 방식은 하드웨어의 부담이 커지게 된다. 그러므로 하드웨어 부담감을 고려하여 8bit rounding하는 방식이 적당함을 알 수 있다.

표 3. 계수길이와 Mapping방법에 따른 MSE

Coefficient width	Mapping Method	Mean square error
Floating-point	no	0.212667
16-bit	truncating	0.996882
16-bit	rounding	0.745101
14-bit	truncating	0.997981
14-bit	rounding	0.746206
8-bit	truncating	6.046496
8-bit	rounding	1.016466
4-bit	truncating	52.606683
4-bit	rounding	4.676830

4.2 CSD 표현과 시뮬레이션

〈CSD의 정의와 특징〉

CSD방식의 디자인은 상수를 가장 적은 non-zero비트를 사용하여 표현할 수 있다. 그리고, 기억장소와 전력 소모를 최소화하여 가장 경제적인 디지트 포맷을 제공한다. CSD의 디자인을 정의하면 다음과 같다. 만약 각각의 i마다 a_i가 {0, 1, −1}의 집합에 속한다면, 우리는 $a_0 \cdot a_1 \cdot a_2 \cdots a_{W-2} \cdot a_{W-1}$에 의해서 $\sum_{i=0}^{W-1} a_i 2^{-i}$로 표현된다. 각 a_i는 한 비트로 정의되고, $a_0 \cdot a_1 \cdot a_2 \cdots$

$a_{W-2} \cdot a_{W-1}$는 W-비트 수로 정의된다. a_i가 0, +1, -1 값을 갖고 a_i가 연속적인 non-zero값을 갖지 않도록 포맷하는 $A = a_0 \cdot a_1 \cdot a_2 \cdots a_{W-2} \cdot a_{W-1}$를 가리켜 CSD라고 한다. 만약 변수 X가 상수 $A = a_0 \cdot a_1 \cdot a_2 \cdots a_{W-2} \cdot a_{W-1}$에 의해서 곱해지면 그 결과치 A·X는 다음과 같이 표현된다.

$$A \cdot X = \sum_{i=0}^{W-1} a_i \cdot X \cdot 2^{-i} = \sum_{i=0}^{W-1} a_i \cdot X \gg i$$

$X \gg i$는 X를 i만큼의 쉬프트 함으로써 나타낼 수 있다. 더해진 항들의 수는 A항에서 non-zero수와 같다. 일반적인 2의 보수의 수이므로 모든 비트를 non-zero로 표현이 가능하게 된다. 최악의 경우 곱셈은 W항만의 덧셈과 같은 번의 연산이 필요할 수 있다.

2보수 형태인 $A = \hat{a}_0 \cdot \hat{a}_1 \cdots \hat{a}_{W-2} \cdot \hat{a}_{W-1}$를 CSD표현으로 나타낼 수 있다. CSD 표현은 다음과 같은 반복 알고리즘으로 2의 보수의 표현으로 얻을 수 있다.

$\hat{a}_W = 0$
$\gamma_W = 0$
$\hat{a}_{-1} = \hat{a}_0$
for (i = W − 1 to 0)
{
$\theta_i = \hat{a}_i \oplus \hat{a}_{i+1}$
$\gamma_i = \overline{\gamma}_{i+1} \theta_i$
$a_i = (1 - 2\,\hat{a}_{i-1})\,\gamma_i$
}

$\oplus$: Exclusive OR, $\overline{r_j}$: 보수화

예로서, 입력 $a_0 a_1 \cdots a_{W-2} a_{W-1} = 1.011110111$로 하면

i	-1	0	.....							W-1	W
$\hat{a}_i$	1	0	1	1	1	1	0	1	1	1	0
θ_i		1	0	0	0	1	1	0	0	1	
γ_i		1	0	0	0	0	1	0	0	1	0
$1-2\,\hat{a}_{i+1}$		-1	1	-1	-1	-1	-1	1	-1	-1	
a_i		-1	0	0	0	0	-1	0	0	-1	

CSD수의 근접한 비트들이 모두 논제로가 아닌 것을 볼 수 있는데 최대의 논제로 비트 수가 W/2인 것을 알 수 있다. [-1, 1]범위의 W-비트의 CSD수들의 평균 논제로 비트 수는 다음과 같다.

$$E[\#\ Non-Zero\ bits] = W/3 + 1/9 + 0(2^{-W}).$$

결과적으로 평균 비트 값은 대략 1/3로 줄었음을 알 수 있고, CSD 수는 2의 보수의 33%정도의 non-zero 비트 수를 가진다[9].

〈분석과 합성의 시뮬레이션〉

위에서 제안한 구조를 gate level 디자인 Tool인 QuickLogic을 사용하여 설계하였다. 시뮬레이션에 사용한 테스트 입력 값은 16진수의 10, 20, 30, 40, 50, 60, 70, 80의 값을 계속 반복하여 입력시켰다. Clock은 100ns로 임의로 정하였다. 다음의 표 4는 한 주기동안에 발생하는 output를 직접 계산한 결과이다. 그림 14의 분석의 시뮬레이션 결과는 출력값이 계산값과 유사함을 알 수 있다.

표 4. 분석의 첫 번째 주기에서 계산되는 결과값

index	Octave	Lowpass필터결과	Highpass필터결과	출력값
0	1	7.7274	-2.0704	-2.0704
1	2	3.7321	-0.9929	-0.9927
2	1	53.5370	0.0005	0.0005
3	3	1.8025	-0.4829	-0.4829
4	1	98.7921	0.0008	0.0008
5	2	94.2296	-18.3196	-18.3196
6	1	144.0472	0.0011	0.0011
7				1. 8025

그림 15는 합성의 결과 값이 원래 입력값의 Stream이 복원되어서 나옴을 알 수 있다.

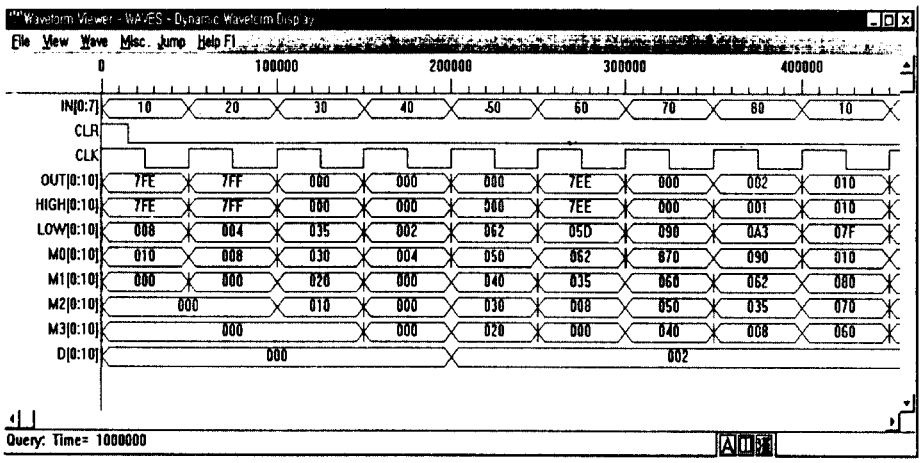


그림 14. 분석의 구조에 대한 시뮬레이션 결과

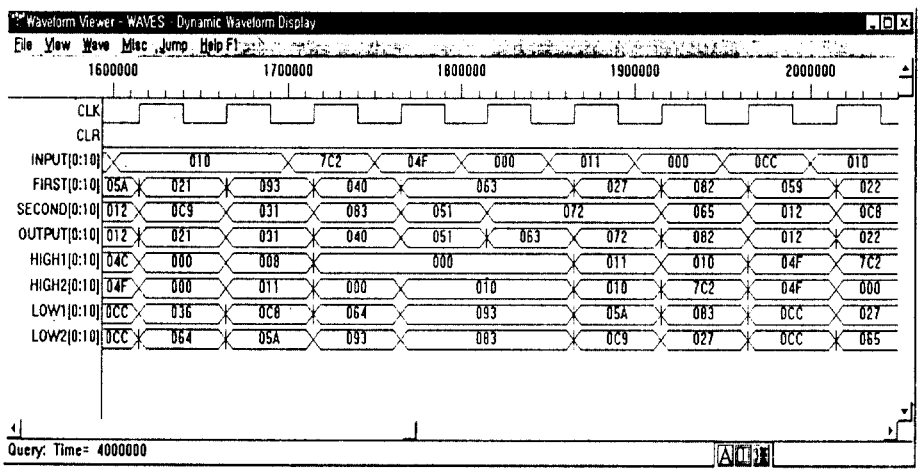


그림 15. 합성의 구조에 대한 시뮬레이션 결과

4.3 분 석

시뮬레이션을 통해 제안한 구조의 올바른 동작을 확인할 수 있었다. 표 5는 기존의 대표적인 구조와 제안 구조의 성능을 비교한 것이다.

표에서 알 수 있듯이 본 연구에서 제안한 방식의 구조는 가장 적은 수의 곱셈기만으로도 구현이 가능하며 이는 하드웨어 구현에서 곱셈기가 차지하는 면적을 고려할 때 유리한 방식임을 알 수 있다. 이러한 장점에도 불구하고 latency면에서도 제안방식은 기존 folded구조와 동일하다.

제안한 구조에서 필요한 곱셈기와 덧셈기 수를 수식화하면 다음과 같다.

곱셈기의 수:$2(L+2)$

덧셈기의 수:$2(((L+1)\times 2)-1)$

제안한 구조의 예상되는 gate의 수는 약 Digit_serial lattice 구조보다 약간 많은 13,800개이지만 Digit-serial lattice구조는 latency가 비교구조중 제일 길고 Word-length가 8의 배수 이어야 하는 제약을 가지고 있다. 동작속도를 결정하는 제안한 구조의 Critical path는 한 번의 곱셈기와 세 번의 덧셈기가 된다. 예를 들어 곱

표 5. 제안 구조와 기존 구조의 성능비교
(J = 3, L(order of filter) = 3)

Property	제안구조	Folded	Folded lattice	Digit-serial	Digit-serial lattice
곱셈기의 수	10	16	12	14	10.5
덧셈기의 수	14	12	8	10.5	7
Inter-connection	complex	complex	complex	simple	simple
latency in cycles	28	28	74	70	168
constraint on Wordlength	None	None	None	Multiple of 8	Multiple of 8
예상 gate 수	약13,800	약17,600	약14,300	약16,000	약13,100

셈기가 50ns에서 덧셈기는 10ns에 동작을 한다면 최고의 속도를 위한 Clock의 주기는 80ns가 되어야 한다. 이것을 파이프라인기법을 사용하여 조절할 수 있다.

V. 결 론

본 논문에서는 효율적인 정규직교 웨이브렛 변환의 특성을 이용하여 1차원 웨이브렛 변환의 구조를 제안했다. 먼저 정규직교 이산 웨이브렛 변환을 위해 사용하는 고대역 필터와 저대역 필터의 계수간의 연관성을 이용하여 연산 상에서의 계산량을 줄이는 방법을 연구하였고 이를 이용해 실시간 처리가 가능한 정규직교 이산웨이브렛 변환을 수행하는 구조를 제안하였다. Dyadic구조를 갖는 이산 웨이브렛 변환의 각 레벨의 출력값들의 특징을 이용하여 각 옥타브에서의 입력형태를 결정하고 필요한 필터의 수를 결정하여 필터의 구조를 설계하였다. 정규 직교 이산 웨이브렛 변환을 위해 사용하는 고대역 필터와 저대역 필터의 계수는 $g_{(n)}=(-1)^n h_{(L-n)}$과 같은 관계를 가지고 있다. 일반적으로 직접적으로 구현할 시 필요한 곱셈기와 덧셈기의 수는 각각 4(L+1), 4L가 된다. 연구한 방법에서는 고대역 필터와 저대역 필터의 관계를 이용하여 곱셈기는 2(L+2)만큼, 덧셈기는 2(((L+1)×2)-1)개가 필요하다. 새로 제안한 분석 및 합성의 구조가 제대로 동작하는지를 확인하기 위해 회로를 구성하고 이를 gate level 설계 툴인 QuickLogic을 이용해 시뮬레이션 하였다. 기존 구조에 비해 새로 제안한 구조는 곱셈기의 수를 6개정도 감소시킬 수 있고 수행속도의 저하 없이 하드웨어 구조를 단순하게 구성할 수 있었다.

본 연구내용은 디지털 신호 처리의 다양한 분야에 응용될 수 있고 고압축을 필요로 하는 무선 네트워크 상에서 이미지 전송에도 응용될 수 있다. 향후 연구방향은 제안한 구조를 이용한 2차원 DWT의 설계와 본 연구에서 고려되지 않은 레지스터 수를 최소화하기 위한 기법에 대해 연구를 계속할 예정이다.

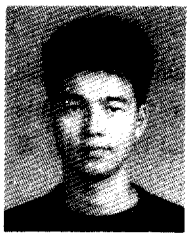
참 고 문 헌

1. P. P. Vaidyanathan, "Multirate Systems and Filter Banks", Englewood Cliffs, NJ: Prentice Hall, 1993.
2. J. Woods and S. O'Nell, "Subband Coding of Images", IEEE Transactions on Acoustics, Speech and Signal Processing, vol. ASSP-34, no. 5, 1278~1288, October 1986.
3. Stephane G. Mallat, "Multi-Frequency Channel Decomposition of Images and Wavelet Representation", IEEE Trans. on Information Theory, 11(7), July 1989.
4. Stephane G. Mallat, "A Theory for Multiresolution Signal Decomposition: The Wavelet Representation", IEEE Transactions on Pattern Analysis and Machine Intelligence, 11(7), July 1989.
5. Ingrid Daubehies, "Orthonormal bases of compactly supported wavelets", Commun. Pure Appl. Math, vol. 41, no. 7, pp. 909~996, 1989.
6. Ingrid Daubehies, "The Wavelet Transform, Time-Frequency Localization and Signal Analysis", IEEE Transactions on Information Theory, 36(5), Sep. 1990.
7. Olivier Rioul and Martin Vetterli, "Wavelets and signal processing", IEEE Signal Processing Magazine, October 1991.
8. Daniel T. L. Lee and Akio Yamamoto, "Wavelet Analysis: Theory and Applications", HP Journal, pp. 44~54, December 1994.
9. K. K. Parhi, "VLSI Digital Signal Processing: Ar-

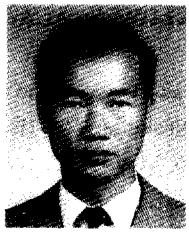
chitectures and Algorithms", KLUWER ACADEMIC PUBLISHERS, 1996.

10. Frank Hartung, "Image Coding and Wavelet Theory: Theory and Performance Evalution of FIR Wavelet Coders", RWTH Aachen, May 1992.
11. R. Coifman, Y. Meyer, S. Quake, and M. V. wickerhauser, "Signal processing and Compression with Wavelet Packets", The Conference on wavelets, Marseilles, Spring, 1989.
12. Daniele D. Giusto, et al, "SAR Image Filtering using Wavelet Transform", ICASSP-95, Vol. 4, pp. 2153~2156.
13. 임은성, 이문호, 박주용, "웨이브렛 변환영역에서 부밴드내의 방향성을 이용한 영상부호화", 제 9회 신호처리합동학술대회, 1995.
14. Eunsung Lim, Sijung chang and Moon Ho Lee, "Image Coding in Wavelet Transform Domain using Non-Linear Quincunx Interpolation", ICCT'96, Beijing, China.
15. Moon Ho Lee, S. J Chang, M. W. Kwon, "A Wavelet Transform Coding With Pre/Post-Processor for Wireless Network", MDMC '96, pp. 142~146, Seoul July 27~28, 1996.
16. G. Knowles, "VLSI Architecture for the Discrete Wavelet Transform", Electronics Lett., vol. 26, no. 15, pp. 1184~1185, July 1990.
17. A. S. Lewis and G. Knowles, "VLSI Architecture for 2-D Daubechies Wavelet Transform Without Multipliers", Electronics Lett., vol. 27, no. 2, pp. 171~173, January 1991.
18. Aware Wavelet Transform Processor reliminary. AWARE, Inc., Cambridge, MA, 1992.
19. K. K. Parhi and Takao Nishitani, "VLSI Architecture for Discrete Wavelet Transforms", IEEE Trans. on VLSI Systems, vol. 1, no. 2, June 1993.
20. T. Denk and K. Parhi, "Calculation of minumum number of registers in 2-D discrete wavelet transforms using lapped block processing", Int. Symp. on Circuits and Systems, pp. 77~81, 1994.
21. C. Nagendra, M. J. Irwin, M. Owens, "Digit Pipelined Discrete Wavelet Transform", ICASSP '94 Digital Signal Processing(VLSI), vol. 3, pp. Ⅱ-405 ~ Ⅱ-408.
22. M. Vishwanath, R. M. Owens and M. J. Irwin, "VLSI Architectures for the Discrete and Continuous Wavelet Transform", IEEE Trans. on Circuits & Systems-Ⅱ, Vol. 42, No. 5, May 1995.
23. C. Chakrabarti, M. Vishwanath and R. M. Owens, "A Survey of Architectures form the Discrete and Continuous Wavelet Transforms", ICASSP-95, Vol. 5, pp. 2849~2852.
24. Jijun Chen and Magdy A. Bayoumi, "A Scalable Systolic Array Architecture for 2-D Discrete Wavelet Transforms", VLSI Signal Processing Ⅷ pp. 303~312, 1995.
25. R. Lang, E. Plesner, H. Schr der, A. Spray, "An efficient systolic architecture for the one-dimensional wavelet transform", SPIE, vol. 2242 Wavelet Applications, pp. 925~935, 1994.
26. Kwo-Jyr Wong and C. C. Jay Kuo, "A full wavelet transform(FWT) approach to image compression", SPIE Vol. 1903 Image and Video Processing, pp. 153~164, 1993.
27. Moon ho Lee, EunSung Lim, DaeChul Park, "A VLSI Architecture for DWT", International Symposium on Consumer Electronics, Nov., 1994, Hong Kong.
28. 장시중, 이문호, 차진종, "새로운 이산웨이브렛 변환을 위한 VLSI구조", 추계전자공학회 학술대회, vol. 19, No. 2, pp. 1017~1020, Nov 1996.
29. Sijung Chang, Moon Ho Lee, "A Simple Parallel Architecture for Discrete Wavelet Transform", ISCAS-97, Vol. 4, pp. 2100~2103.
30. Sijung Chang, Moon Ho Lee, Ju Yong Park, "A High Speed VLSI Architecture of Dicrete Wavelet Transform for MPEG-4", IEEE Trans. on Consumer Electronics. vol 43, No 3. Aug 1997.

장 시 중(Si Jung Chang) 정회원
1971년 12월 9일생
1996년 2월 : 전북대학교 정보통신공학과 졸업(공학사)
1996년 3월~현재 : 전북대학교 정보통신공학과 석사과정 중
※주관심분야 : 영상신호처리, 통신용 VLSI.



김 대 용(Dae Yong Kim) 학생회원
1974년 10월 22일생
1997년 2월 : 전북산업대학교 정보통신공학과 졸업 (공학사)
1997년 3월~현재 : 전북대학교 영상정보공학과 석사과정 중
※주관심분야 : MPEG-4 동영상 처리, 영상신호처리, VLSI.



김 순 영(Kim Soon Yung) 정회원
1960년 10월 5일생
1983년 2월 : 한국항공대학교 항공기계공학과 졸업(공학사)
1987년 3월~현재 : 한국통신 인력개발본부 재직중
1994년 8월 : 전북대학교 전기통신전공 졸업(공학석사)
1997년 3월~현재 : 전북대학교 컴퓨터공학과 박사과정 중
※주관심분야 : 영상신호처리, TDX 교환기, CDMA

이 문 호(Moon Ho Lee) 정회원
1997년 22권 제 1호 참조.