2003년도 대한전자공학회 하계종합학술대회 논문집 제26권 제1호

Lifting-Based Scheme 을 이용한 DWT 의 개선된 ROW Processor 구현

최영철* 정영식 장영조
한국기술교육대학교 정보기술공학부

Improved Row Processor of DWT using a Lifting-Based Scheme

Choi Young Chul*, Jeung Young Sic, Jang Young Jo
Korea University of Technology and Education
ycchoi@kut.ac.kr sikiga@hanmail.net yjjang@kut.ac.kr

Abstract

본 논문에서는 Lifting-Based Scheme을 이용한 DWT(Discrete Wavelet Transform) 의 개선된 행 처리기의 구조를 제안 하였다. 제안된 행 처리기는 3개의 Adder 와 2개의 shifter를 사용 하였고 dual-port RAM을 사용하여 파이프 라인 구조를 취하여 각 클럭마다 열 처리기에서 사용할 데이터를 발생 한다. 이러한 행 처리기의 파이프 라인 구조를 개선하여 Adder를 줄이고 행 처리기의 이용률을 최대로 하여 하드웨어의 공간적 비용 절감 효과를 가져 왔다. 제안된 구조는 Verilog를 사용하여 RTL설계를 한뒤 시뮬레이션으로 그 동작을 확인 하였다.

I. 서 론

DWT는 전통적으로 컨볼루션 에 의해 구현 되어져 왔지만, 구현함에 있어서 많은 계산량과 큰 저장 용량을 필요로 하므로 빠른 스피드나 저전력 응용 분야에 대해서는 바람직하지 않다. 최근에 DWT구현에 있어 계산량이 더 적은 Lifting-Based Scheme [2][3][4] 이 제안되었다.

II. 본 론

2.1 Lifting-Based Scheme

Lifting-scheme의 기본 원리는 웨이블릿 필터의 polyphase행열를 인수 분해 하는 것이다. [2]

$\tilde{h}(z)$ 와 $\tilde{g}(z)$ 을 lowpass와 highpass 분해 필터라 하고 $h(z)$ 와 $g(z)$ 를 lowpass와 highpass 합성 필터라 가정하면 polyphase행열 $\tilde{P}(z)$ 와 $P(z)$ 는 식(1)과 같이 정의 된다.

$$\tilde{P}(z)=\begin{bmatrix}\tilde{h}_e(z) & \tilde{h}_o(z)\\ \tilde{g}_e(z) & \tilde{g}_o(z)\end{bmatrix} \quad P(z)=\begin{bmatrix}h_e(z) & g_e(z)\\ h_o(z) & g_o(z)\end{bmatrix} \quad \text{--- 식(1)}$$

만약 $(\tilde{h},\tilde{g})$ 가 상보필터 쌍 이면 $\tilde{P}(z)$ 는 lifting단계에서는 식(2)나 식(3)과 같은 식으로 항상 인수분해 될수 있다.

$$\tilde{P}_1(z)=\begin{bmatrix}K & 0\\ 0 & 1/K\end{bmatrix}\prod_{i=1}^{m}\begin{bmatrix}1 & \tilde{s}_i(z)\\ 0 & 1\end{bmatrix}\begin{bmatrix}1 & 0\\ \tilde{t}_i(z) & 1\end{bmatrix} \quad \text{--- 식(2)}$$

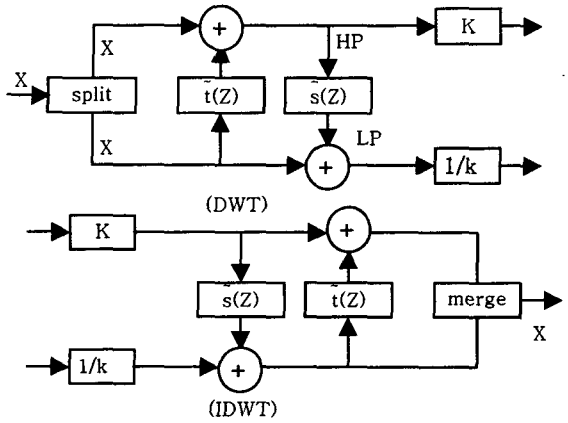
$$\tilde{P}_2(z)=\begin{bmatrix}K & 0\\ 0 & 1/K\end{bmatrix}\prod_{i=1}^{m}\begin{bmatrix}1 & 0\\ \tilde{t}_i(z) & 1\end{bmatrix}\begin{bmatrix}1 & \tilde{s}_i(z)\\ 0 & 1\end{bmatrix} \quad \text{--- 식(3)}$$

(여기서 K는 상수)

$\tilde{P}_1$ 의 scheme을 나타내 보면 그림(1)과 같이 3단계 로 구성된 것을 알수 있다.

1) Predict 단계 : $\tilde{t}(z)$ 의 시간 영역 에서 발생한 값과 짝수 샘플 값이 곱해진후 홀수 샘플 값과 더해진다.
2) Update 단계 : $\tilde{s}(z)$ 의 시간 영역 에서 발생한 값과 update 된 홀수 샘플 값과 곱해진후 짝수 샘플 값과 더해진다.

3) Scaling 단계 : 짝수 샘플은 1/K 와 곱해지고 홀수 샘플은 K 와 곱해진다.



그림(1) – Lifting Scheme 의 3 단계

(5,3) 필터의 경우를 생각한다면, 필터 계수는 다음과 같다

Highpass : (-1/8, 2/8, 6/8, 2/8, -1/8)

Lowpass : (-1/2, 1, -1/2)

Polyphase행열 $\tilde{P}_1(z)$ 는

$$\tilde{P}_1(z)=\begin{bmatrix} -\frac{1}{8}z+\frac{6}{8}-\frac{1}{8}z^{-1} & \frac{2}{8}+\frac{2}{8}z \\ -\frac{1}{2}-\frac{1}{2}z^{-1} & 1 \end{bmatrix}$$ 로 나타내어지고,

$\tilde{P}_1(z)$ 를 인수 분해 하고 시간영역 상에 나타내면

$$\tilde{P}_1(z)=\begin{bmatrix} 1 & 0.25(1+z) \\ 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 \\ -0.5(1+z^{-1}) & 1 \end{bmatrix}$$ 이다.

시간영역 상에서 행열로 해석하면

$$y_{2i+1}=-0.5(x_{2i}+x_{2i+2})+x_{2i+1}$$

$$y_{2i}=0.25(y_{2i+1}+y_{2i+3})+x_{2i},\ 0\le i\le N/2$$

여기서 x_i 는 입력 신호값이고, y_i 는 변환된 신호값이다. 이를 행열로 표현 하면 식(4)와 같다.

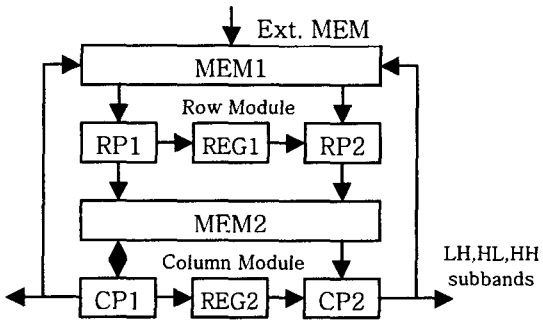
$$\mathbf{M}_1=\begin{bmatrix} 1 & a & 0 & . & . & . & . & 0 \\ 0 & 1 & 0 & 0 & . & . & . & . \\ 0 & a & 1 & a & 0 & . & . & . \\ . & 0 & 0 & 1 & 0 & 0 & . & . \\ . & . & 0 & a & 1 & a & 0 & . \\ . & . & . & 0 & 0 & 1 & 0 & 0 \\ . & . & . & . & 0 & a & 1 & a \\ 0 & . & . & . & . & 0 & a & 1 \end{bmatrix} \quad \mathbf{M}_2=\begin{bmatrix} 1 & 0 & 0 & . & . & . & . & 0 \\ 0 & 1 & b & 0 & . & . & . & . \\ 0 & 0 & 1 & 0 & 0 & . & . & . \\ . & 0 & b & 1 & b & 0 & . & . \\ . & . & 0 & 0 & 1 & 0 & 0 & . \\ . & . & . & 0 & b & 1 & b & 0 \\ . & . & . & . & 0 & 0 & 1 & 0 \\ 0 & . & . & . & . & 0 & 0 & 1 \end{bmatrix}$$

(a=-0.5 , b = 0.25) ---- 식(4)

DWT변환 값 $\mathbf{Y} = \mathbf{X} * \mathbf{M}_1 * \mathbf{M}_2$ 이고 IDWT변환한 원 신호 값 $\mathbf{X} = \mathbf{Y} * \mathbf{M}_2 * \mathbf{M}_1$ 이 된다.

2.2 제안된 구조

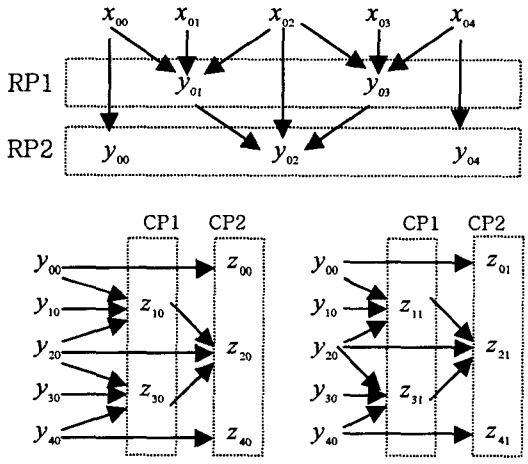
2.2.1 [1]에서 제안한 전체 블록도



그림(2)- DWT구조의 전체 블록도

그림(2)는 [1]에서 제안한 DWT구조의 전체 블록도 이다. 이 구조 에서는 N X N 크기의 블록 데이터를 행-열 방식으로 실행하여 각각의 분해 단계(decomposition level))에서 LH,HL,HH 데이터를 출력하고 LL 데이터는 다음 분해 단계 에서 사용 된다.

2.2.2 변환 module 형태



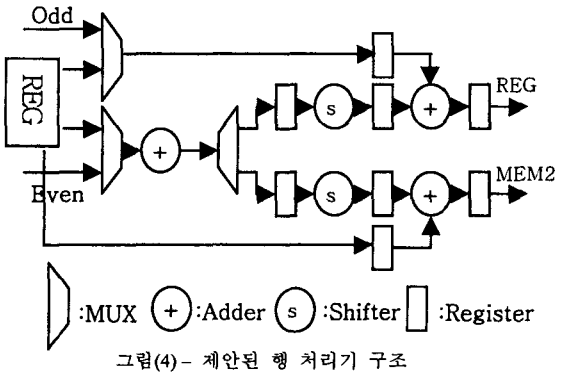
그림(3) – 행-열 처리기의 데이터 접근 방식

RP1에서는 열 방향으로 highpass(홀수) 요소인 $y_{01}, y_{03}, \ldots$ 등을 계산하고, RP2는 lowpass(짝수) 요소인 $y_{00}, y_{02}, y_{04}, \ldots$ 등을 계산한다. CP1 은 홀수행을 따라 highpass와 lowpass요소인 $z_{10}, z_{11}, \ldots; z_{30}, z_{31}, \ldots$ 등을 계산하고, CP2는 짝수행을 따라 highpass와 lowpass 요소

인 $z_{00}, z_{01}, \ldots; z_{20}, z_{21}, \ldots; z_{40}, z_{41}, \ldots$ 등을 계산 한다. CP1과 CP2는 RP1과 RP2에 의해서 발생되어진 요소가 계산 조건에 만족되었을 때 계산을 시작한다.

결과적으로 RP1과 RP2는 행 방향으로 각각 highpass와 lowpass를 계산하고, CP1과 CP2는 열 방향으로 각각 highpass와 lowpass를 계산한다.

2.2.3 제안된 행 처리기 구조



그림(4) - 제안된 행 처리기 구조

<표 1>-(5,3) 필터가 적용된 9x9 블록 행 처리기 표

Time	RP1			RP2	
	Adder1	Shifter	Adder2	Shifter	Adder2
1	-	-	-	-	-
2	$x_{0,0}+x_{0,2}$	-	-	-	-
3	$x_{0,2}+x_{0,4}$	RA1	-	-	-
4	$x_{0,4}+x_{0,6}$	RA1	RS- $x_{0,1}$	-	-
5	$x_{0,6}+x_{0,8}$	RA1	RS- $x_{0,3}$	-	-
6	$y_{0,1}+y_{0,3}$	RA1	RS- $x_{0,5}$	-	-
7	$x_{2,0}+x_{2,2}$	-	RS- $x_{0,7}$	RA1	$y_{0,3}+y_{0,5}$
8	$x_{2,2}+x_{2,4}$	RA1	$y_{0,5}+y_{0,7}$	RA1	RS+ $x_{0,2}$
9	$x_{2,4}+x_{2,6}$	RA1	RS- $x_{2,1}$	RA1	RS+ $x_{0,4}$
10	$x_{2,6}+x_{2,8}$	RA1	RS- $x_{2,3}$	-	RS+ $x_{0,6}$
11	$y_{2,1}+y_{2,3}$	RA1	RS- $x_{2,5}$	-	
12	$x_{1,0}+x_{1,2}$	-	RS- $x_{2,7}$	RA1	$y_{2,3}+y_{2,5}$
13	$x_{1,2}+x_{1,4}$	RA1	$y_{2,5}+y_{2,7}$	RA1	RS+ $x_{2,2}$
14	$x_{1,4}+x_{1,6}$	RA1	RS- $x_{1,1}$	RA1	RS+ $x_{2,4}$

본 논문에서는 그림(4)와 같이 행 처리기 구조를 제안 하였고, 또한 행 처리기 구조의 동작을 <표1>에 나타냈다. [1]의 행 처리기 에서 Adder의 하드웨어 사용률은 4개의 Adder모두 80%만 사용하였다. <표1>의 음영부분이 [1]의 Adder에서 사용하지 않은 20%의 공간이다. 본 논문에서는 이렇게 낭비되는 부분을 RP1과 RP2에서 재사용 함으로써 같은 시간 클럭에 하드웨어 이용률을 100%로 만들었고, 또한 하드웨어의 제작에 있어 Adder보다 상대적으로 싼 MUX를 사용하여 하드웨어의 비용 절감을 할수 있게 했다.

2.2.4 성능 비교

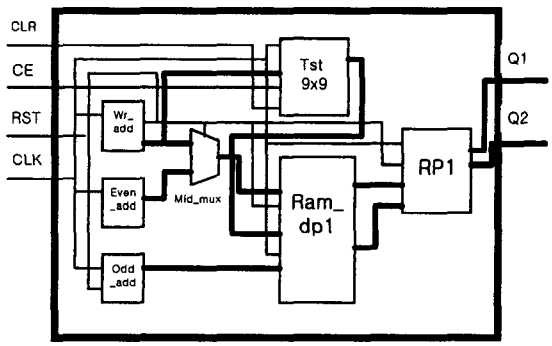
기존의 행 처리기와 본 논문에서 제안된 행 처리기의 하드웨어 사용 비교한 것을 <표 2>에서 보인다.

<표 2> 기존의 행 처리기와 제안된 행 처리기 비교

	기존의 행 처리기	제안된 행 처리기
Adder	**4**	**3**
Shifter	**2**	**2**
Register	**8**	**8**
MUX	**0**	**3**
H/W 사용률	**RP1:80% RP2:80%**	**RP1:100% RP2:80%**

(*실행 시간 동일)

2.2.5 구현

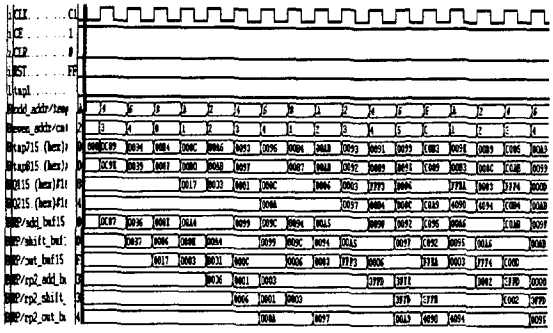


그림(5) 개선된 행 처리기 블록도

본 논문에서 제안한 개선된 행 처리기의 블록도는 그림(5)와 같고, 각 블록의 동작 사항은 다음과 같다.

Test를 위해 9x9 블록 단위의 영상 정보를 ROM에 저장한후 RST 신호가 발생하면 Wr_add에 의해 dual-port ram(Ram_dp1)블록에 ROM에 저장된 데이터를 저장한다. Ram_dp1에 저장된 데이터를 짝수와 홀수로 나누기 위해 Even_add와 Odd_add의 주소 발생기를 만들어 이에 따라 행처리기블록(RP1)에 짝수 및 홀수

데이터를 파이프 라인으로 제공 한다. 행처리기 블록에서 이 데이터를 처리하여 *CP1*과 *CP2*에서 처리하기 위한 데이터를 Q1과 Q2를 통해 발생한다. 여기서 Ram_dp1블록을 사용함으로써 행처리기의 파이프라인 구조를 만들고 행 처리기의 이용률을 100%로 만들수 있다.



그림(6) - 개선된 행 처리기의 시뮬레이션 파형

본 논문에서 제안한 행 처리기의 동작 확인을 위해 xilinx Foundation 3.1i 의 시뮬레이터를 사용하였고, 시뮬레이션 결과는 그림(6) 과 같이 입력을 파이프라인으로 받고 일정 클럭 이후의 출력 결과가 Q1과 Q2에서 각각 4개 및 5개의 출력값이 반복 되는 것을 확인 할수 있다. 이것을 <표1>과 비교 하면 동일 결과임을 확인 할수 있고, 출력값은 행 처리기의 C 시뮬레이션한 값과 비교 하였을 때 0.01~0.05의 오차를 나타냈고, 이 오차는 각 단계에서 계산후 나오는 소수점 처리 과정에서 발생한 것이다. Verilog에서 소수점 처리를 할 때 반올림 처리를 수행하기 때문에 이러한 결과가 발생하였다.

III. 결 론

최근에 많은 DWT구조의 실시간 처리 요구가 제안되어지고, 저전력소모와 하드웨어의 소형화로 인하여 하드웨어 구조의 이용률을 높일 필요가 대두 된다. 본 논문에서는 DWT의 행 처리기의 파이프 라인 구조를 개선하여 Adder를 줄이고 행 처리기의 이용률을 최대로 하여 하드웨어의 공간적 비용 절감 효과를 가져왔다.

III. 참고 문헌

[1] Kishore Andra, Chaitali Chakrabarti,Tinku Acharya, "A VLSI Architecture for Lifting-Based Forward and Inverse Wavelet Transform" ,IEEE Transactions on Signal Process, vol. 50, No. 4, April 2002

[2] K.andra, C. Chakrabarti, and T. Acharya, " A VLSI architecture for lifting based wavelet transform," in Proc. IEEE Workshop Signal Process. Syst., Oct. 2000, pp.70-79.

[3] W. Sweldens, "The lifting scheme: A new philosophy in biorthogonal wavelet constructions," in *Proc. SPIE*, vol. 2569, 1995, pp. 68–79.

[4] K. K. Parhi and T. Nishitani, "VLSI architectures for discrete wavelet transforms," *IEEE Trans. VLSI Syst.*, vol. 1, pp. 191–202, June 1993.