논문 2005-42SD-4-3

리프팅 스킴의 2차원 이산 웨이브릿 변환 하드웨어 구현을 위한 고속 프로세서 구조 및 2차원 데이터 스케쥴링 방법

(A Fast Processor Architecture and 2-D Data Scheduling Method to Implement the Lifting Scheme 2-D Discrete Wavelet Transform)

김 종 욱*, 정 정 화**

(Jong Woog Kim and Jong Wha Chong)

요 약

본 논문에서는 리프팅 스킴의 2차원 고속 웨이블릿 변환에서 2차원 처리 속도를 향상시키고, 내부 메모리 사이즈를 감소 시키는 병렬 처리 하드웨어 구조를 제안한다. 기존의 리프팅 스킴을 이용한 병렬 처리 2차원 웨이블릿 변환 구조는 행 방향의 예측, 보상 연산 모듈과 열 방향의 예측 보상 연산 모듈로 구성 되며, 2차원 웨이블릿에서 열 방향 변환을 위해서는 행 방향의 결과가 나와야 하고, 열 방향 연산을 위한 데이터가 연속적으로 발생하는 것이 아니라 행 방향의 샘플 데이터 수만큼의 시차를 갖고 발생함으로 내부 버퍼를 사용하고 있다. 이에 제안하는 구조에서는 행 방향 연산에 있어서 짝수 행과 홀수 행을 동시에 할 수 있도록 하드웨어 구조와 데이터 흐름을 구성하여 속도를 향상 시키고, 열 방향 연산의 시작 지연 시간을 단축 시켰다. 그리고, 행 방향 처리 결과를 버퍼에 저장하지 않고 열 방향 연산의 입력으로 사용할 수 있도록 열 방향 처리 모듈을 개선하였다. 제안하는 구조는 입력 데이터를 4개의 분할 셋으로 분할하여 기존의 2개의 입력 데이터를 동시에 처리하는 방식에서 4개의 입력 데이터를 동시에 받아 처리 할 수 있도록 데이터의 흐름과 각 모듈의 연산 제어를 구성하였다. 그 결과 행 방향 연산 속도를 향상 시키고, 열 방향 연산 수행의 지연을 줄여 내부 버퍼 메모리를 절반으로 줄일 수 있었다. 제안하는 데이터 흐름과 하드웨어 구조를 이용하여 VHDL을 이용하여 설계한 결과 기존의 $N^2/2+\alpha$ 의 전체 처리 시간을 $N^2/4+\beta$ 로 줄이는 결과를 얻었고, 내부 메모리 역시 기존의 방법에 비해 최대 50%까지 줄이는 결과를 얻을 수 있었다.

Abstract

In this paper, we proposed a parallel fast 2-D discrete wavelet transform hardware architecture based on lifting scheme. The proposed architecture improved the 2-D processing speed, and reduced internal memory buffer size. The previous lifting scheme based parallel 2-D wavelet transform architectures were consisted with row direction and column direction modules, which were pair of prediction and update filter module. In 2-D wavelet transform, column direction processing used the row direction results, which were not generated in column direction order but in row direction order, so most hardware architecture need internal buffer memory. The proposed architecture focused on the reducing of the internal memory buffer size and the total calculation time. Reducing the total calculation time, we proposed a 4-way data flow scheduling and memory based parallel hardware architecture. The 4-way data flow scheduling can increase the row direction parallel performance, and reduced the initial latency of starting of the row direction calculation. In this hardware architecture, the internal buffer memory didn't used to store the results of the row direction calculation, while it contained intermediate values of column direction calculation. This method is very effective in column direction processing, because the input data of column direction were not generated in column direction order. The proposed architecture was implemented with VHDL and Altera Stratix device. The implementation results showed overall calculation time reduced from $N^2/2+\alpha$ to $N^2/4+\beta$, and internal buffer memory size reduced by around 50% of previous works.

Keywords : Wavelet, Hardware, fast architecture

I. 서 론

최근 들어 정지 영상과 동영상의 압축을 위한 변환 기법으로 웨이블릿 변환이 각광을 받고 있고, MPEG-4,

* 학생회원, ** 평생회원, 한양대학교 전자공학과
(Dept., of Electrical Engineering, Hanyang Univ.)
접수일자: 2004년6월4일, 수정완료일: 2005년4월1일

JPEG2000과 같은 표준화 기구에서 영상 압축에 있어서 DWT(Discrete Wavelet Transform)를 채택하고 있다[1][2]. 기존의 블록 DCT 기반의 압축 코덱에서는 압축률이 올라감에 따라서 블록의 경계 부분에 심한 열화 현상이 나타나는 블로킹 현상(blocking artifacts)이 나타나는 단점을 가지고 있었다. 그리고, 블록 DCT 기반의 코덱은 다양한 전송 환경에 따라 비례 축소가 가능한(scalability) 코덱을 구현하는데 약점을 가지고 있었다. 이에 반해 웨이블릿은 하나의 프레임을 블록 단위로 분할하지 않고 전체 영상에 대해 변환하기 때문에 블록킹 현상을 줄일 수 있고, 변환의 결과가 웨이블릿의 레벨에 따라 밴드 별로 나누어지기 때문에 전송 레벨의 조정과 양자화 스텝 변화를 통해 화상 크기와 화질에 따라 가변적인 코덱을 구현 할 수 있다는 장점을 가지고 있다. 이러한 웨이블릿 기반의 압축 코덱에 있어서 가장 큰 문제점은 변환의 연산이 전체 영상이 대하여 수행해야 하기 때문에 2차원의 DWT(Discrete Wavelet Transform)의 최종 결과를 얻기 위한 지연 시간이 DCT에 비해 상대적으로 길고, 1차원 처리 결과를 이용하여 2차원 처리를 수행해야 함으로 중간의 1차원 처리 결과를 저장해 놓기 위해 한 프레임만큼의 버퍼가 필요하다는 단점을 가지고 있다.

웨이블릿 변환의 문제를 해결하기 위해 다양한 형태의 웨이블릿 기반의 하드웨어 아키텍처가 제안 되고 있다. 기존의 제안된 하드웨어 구조를 보면 크게 두가지로 나누어지는데, 한 가지는 필터를 2차원 구조로 구성하는 시스톨릭 어레이 구조로 구성하는 방법과[3], 병렬처리 구조를 이용하는 방법으로 구분할 수 있다[4][6][8][10]. 이중 시스톨릭 어레이 구조의 경우는 각 프로세서 모듈이 메모리를 갖고 그 외에 버퍼 메모리가 또 필요한 단점을 가지고 있다. 이에 비해 병렬 처리 모듈의 시스톨릭 어레이에 비해 연산 모듈의 하드웨어 코스트를 줄일 수 있다는 장점을 갖지만, 여전히 내부 버퍼 메모리 문제와 처리속도의 향상에 있어서도 최대 $N^2/2+\alpha$에 머물러 있다는 것이다. 현재 연산 모듈을 구성하는 필터 모듈에 대해서는 곱셈기 대신에 쉬프터(shifter)와 덧셈기 만으로 구성하는 방법과 리프팅 스킴의 필터 계수를 이용하여 하드웨어 코스트를 줄이는 하드웨어 구조가 계속 제안 되고 있다[4][5][6]. 이러한 기존의 방법들은 하드웨어 코스트를 줄이고 내부 버퍼 메모리를 줄이고 있지만, 1차원 웨이블릿을 수행 하는 과정에 있어서 데이터 분할을 2개로만 하기 때문에 처리속도의 향상에 있어서 $N^2/2$이하로 줄이는 것에 한계를 가지고, 내부 버퍼 메모리가 많이 필요한 가장 큰 이유는 2차원의 경우 1차원의 처리를 기다려야 하기 때문에 1차원의 결과 지연이 많이 될수록 많은 버퍼가 필요하게 된다.

제안하는 구조에서는 이러한 처리속도와 내부 메모리 문제를 해결하기 위해서 기존의 구조에서 사용하는 1차원 적인 분할 방법을 2차원으로 분할하여 1차원의 처리 결과를 기존의 방법에 비해 2배 빠르게 얻을 수 있도록 하드웨어를 구성하였다. 1차원 결과의 빠른 처리는 2차원 연산 시작의 지연을 최소화 할 수 있고, 2차원 연산이 시작되어 결과가 나오면 1차원 연산의 결과를 저장할 필요가 없으므로 내부 버퍼 메모리를 줄일 수 있다. 그리고 2차원 처리에 있어서 열 방향 연산의 처리가 매 클럭 마다 일어 날 수 있도록 버퍼 메모리를 구성하였다. 이러한 방법을 통해 전체 연산 시간을 줄이고, 내부 버퍼 메모리를 감소시키는 결과를 얻을 수 있었다.

본 논문의 구성은 Ⅱ장에서 리프팅 스킴의 웨이블릿 변환을 설명하고, Ⅲ장에서는 제안하는 분할방법, 병렬처리 하드웨어 아키텍처와 지연 시간과 수행 흐름을 설명하였다. 그리고 최종적으로 Ⅳ장에서는 기존의 구조와의 결과 비교를 기술하고, Ⅴ장에서 결론을 맺는다.

Ⅱ. 리프팅 스킴의 웨이블릿 변환

기존의 웨이블릿 변환에 있어서 변환 과정은 웨이블릿 필터 계수를 이용하여 컨벌루션 필터 연산을 통해 웨이블릿 변환 결과를 얻게 된다. 이에 비해 리프팅 스킴은 웨이블릿 변환 필터를 폴리페이스 필터형태로 기술하고 이것을 이용하여 웨이블릿 변환을 수행하는 구조를 갖는다.

1. 리프팅 스킴의 웨이블릿 필터

웨이블릿 필터를 폴리페이즈 필터 형태의 리프팅 스킴의 웨이블릿 필터로 구현 하였을 때, 기존의 컨벌루션 필터 방식에 비해 필터 탭수를 줄일 수 있고, 보다 간단한 구조의 하드웨어를 구성할 수 있기 때문에 장점을 갖는다[7]. 본 장에서는 기존의 웨이블릿 필터를 통해서 폴리페이즈 필터 형식의 리프팅 스킴에 대하여 설명한다.

컨벌루션 형태의 웨이블릿 필터는 다음의 식(1)과 같이 구성된다. 여기서 h와 g는 각각 저주파 필터와 고주파 필터를 의미하고, L_h와 L_g는 각각 저주파 필터의 탭

수와 고주파 필터의 탭수를 의미한다.

$$x_L(k) = \sum_{i=0}^{L_h-1} h(i)x(2k-i)$$
$$x_H(k) = \sum_{i=0}^{L_g-1} g(i)x(2k-i) \quad (1)$$

리프팅 스킴은 기존의 웨이블릿 필터를 유한한 스탭을 갖는 필터들로 구성할 수 있다는 것으로 이것은 기존 필터를 폴리페이즈 필터로 인수 분해 하는 과정을 통해 얻을 수 있다. 식(2)와 같이 기존의 필터는 짝수 항과 홀수 항으로 표현하고, 식(3)의 P 메트릭스를 정의하고, P 메트릭스를 폴리페이즈 인수 분해 과정을 거치면 $s_i(z)$와 $t_i(z)$로 구성되는 결과 식을 얻을 수 있다. 여기서 i는 폴리페이즈 필터의 인수 분해 스텝을 의미하고 이것이 곧 리프팅 스킴의 탭을 의미한다. 그리고, $s_i(z)$와 $t_i(z)$가 리프팅 스킴에서의 예측과 보상 필터을 의미하게 된다.

$$h(z) = h_e(z^2) + z^{-1}h_o(z^2)$$
$$g(z) = g_e(z^2) + z^{-1}g_o(z^2) \quad (2)$$

$$P(z) = \begin{matrix} h_e(z) & g_e(z) \\ h_o(z) & g_o(z) \end{matrix}$$
$$= \prod_{i=1}^{m} \begin{matrix} 1 & s_i(z) \\ 0 & 1 \end{matrix} \begin{matrix} 1 & 0 \\ t_i(z) & 1 \end{matrix} \begin{matrix} K & 0 \\ 0 & \frac{1}{K} \end{matrix} \quad (3)$$

이러한 리프팅 스킴을 이용하여 (9,7) 웨이블릿 필터를 리프팅 스킴으로 변환하면 다음과 같은 간단한 형태의 리프팅 스킴 웨이블릿 필터를 구할 수 있다.

저역 통과 필터: $\left[\frac{1}{4} \quad 1 \quad \frac{1}{4}\right]$

고역 통과 필터: $\left[-\frac{1}{16} \quad \frac{9}{16} \quad 1 \quad \frac{9}{16} \quad -\frac{1}{16}\right]$

리프팅 스킴의 웨이블릿 필터에서 저역 통과 필터를 보상(update) 필터라 하고, 고역 통과 필터를 예측(prediction) 필터라 한다. 각 필터를 그림 1과 같이 구성하여 리프팅 스킴의 웨이블릿을 구성하게 된다. 그림 1에서 각각 보상(update)와 예측(prediction) 과정이 웨이블릿 변환 과정이다.

2. 기존의 하드웨어 구조

기존의 병렬 구조에서는 2차원 웨이블릿 변환을 위하여 행 방향 예측, 보상 프로세서와 열 방향 예측, 보상 프로세서를 이용하는 구조를 갖는다[10][4]. 그 외에 하드웨어 자원을 공유하는 방법이 제안되고 있다[9]. 리프팅 구조를 이용한 기존의 웨이블릿 하드웨어 구조는 2개의 보상 필터 모듈과 2개의 예측 필터 모듈을 이용하여 행 방향과 열 방향의 변환 과정을 병렬로 수행 하도록 구성하고 있다. 즉, 한 쌍의 예측과 보상 필터를 행 방향 연산에 사용하고, 다른 한 쌍을 열 방향 연산에 사용하는 구조를 갖는다. 그림 2의 구조와 같이 행 방향 입력을 받아서 그 결과가 내부 버퍼 메모리에 저장되고, 열 방향 연산이 수행 되게 된다.

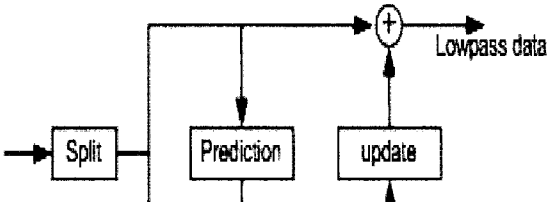


그림 1. 리프팅 스킴의 웨이블릿 변환 구성도
Fig. 1. Transform diagaram of lifting scheme based wavelet transform.

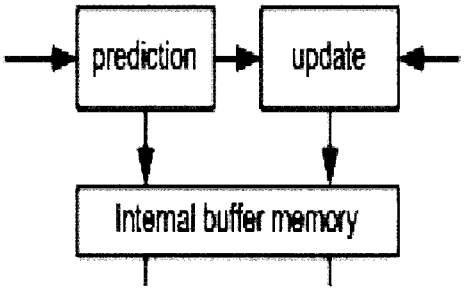


그림 2. 기존의 병렬 처리 하드웨어 구조의 간략한 구성
Fig. 2. A simple configuration of the previous parallel hardware architecture.

그림 2와 같은 구조에서 전체 처리에 필요한 시간은 $N^2/2+\alpha$만큼의 시간이 필요하게 된다. 이것은 입력 NxN 데이터가 $(N/2)\times N$의 데이터로 분할되어 예측과 보상이 병렬로 처리 되고, 초기 α만큼의 시간 지연 뒤에는 열 방향 처리가 병렬로 처리 된다.

Ⅲ. 제안하는 하드웨어 구조

제안하는 구조는 기존의 구조와 달리 3쌍의 예측과 보상 필터 쌍으로 구성되고, 이로 인한 하드웨어 증가 사이즈를 줄이기 위해 필터 계수의 대칭성을 이용한 하드웨어 구조를 제안하고, 처리 속도와 내부 버퍼 메모리의 크기를 줄이기 위해 트랜스포즈(transpose) 방식

의 FIR 필터 구조를 사용하는데 적합한 데이터 흐름제어를 제안한다.

1. 제안하는 전체 하드웨어 구조

일반적인 리프팅 스킴은 NxN의 입력 데이터를 두개의 데이터 세트(set)로 나누어 처리하게 되는데, 짝수(even) 위치의 데이터를 이용해 홀수(odd) 위치의 데이터 값을 예측하여 고역 통과 필터의 결과를 구하고, 그 결과를 이용하여 짝수 위치의 값에 보상(update) 하는 과정을 통해 저역 통과 필터연산을 수행한다. 이러한 일반적인 과정에서 행 방향 처리 속도를 향상시키기 위해 2차원 분할을 사용하는 방법이 제안 되었다[9]. 2차원 분할을 이용 하여 행 방향 연산을 수행할 경우 식(4)와 같은 일반적인 리프팅 스킴을 그림 3과 같이 분할하면 식(5)와 같이 행 방향 연산을 수행 할 수 있다.

$$\begin{aligned} \hat{x}_H &= x_H - P(x_L) \\ \hat{x}_L &= x_L + U(\hat{x}_H) \end{aligned} \tag{4}$$

$$\begin{aligned} \hat{x}_{LH} &= x_{LH} - P(x_{LL}) \\ \hat{x}_{HH} &= x_{HH} - P(x_{HL}) \\ \hat{x}_{LL} &= x_{LL} + U(\hat{x}_{LH}) \\ \hat{x}_{HL} &= x_{HL} + U(\hat{x}_{HH}) \end{aligned} \tag{5}$$

식(5)에서 $\hat{x_{LL}}$와 $\hat{x_{HL}}$은 동시에 연산을 수행 할 수 있고, $\hat{x_{HH}}$와 $\hat{x_{LH}}$역시 동시에 수행이 가능하다. 2차원 분할을 사용하였을 때, 완전한 병렬 처리를 위해서는 4쌍의 예측, 보상 필터가 필요한데 이것을 3쌍의 필터로 처리가 가능하도록 전체 구조를 제안하였다.

제안하는 전체 하드웨어 구조는 세 쌍의 예측과 보상 필터로 구성되고, 전체 구성은 그림 4와 같다. 그림 4에서 DWT과정을 수행하기 위하여 입력 데이터는 그림 3의 분할의 결과 데이터가 사용된다. 이러한 구성은 초기 행 방향의 연산 처리 속도를 2배로 향상 시켜 기존의 구조들이 갖는 열 방향 연산의 초기 지연을 감소시킬 수 있다. 이러한 구조를 이용하면 행 방향 초기 지연시간 $L_h(L_h$: 고주파 필터의 탭수) 이후에 나오는 출력값을 이용하여 열 방향 연산을 시작할 수 있고, L_hN 시간 이후에는 최초의 2차원 DWT 결과를 얻을 수 있다.

행 방향 DWT에서 행과 행 사이에 상호 연관성이 없으므로, 2개의 행에 대하여 병렬 연산이 수행된다. 일반적인 병렬 처리 구조에서 2쌍의 예측과 보상 필터 모듈을 사용하고 있고, 1쌍을 행 방향 1쌍을 열 방향 연산에 할당하고 있기 때문에 2개의 행을 동시에 처리하는 것이 어렵게 되어 있다[7]. 이러한 구조는 열 방향의 연산의 결과를 얻기 위해서 $2L_h$ 행만큼을 처리한 후에야 열 방향 연산의 결과를 얻을 수 있게 된다. 결국 $2L_hN$ 만

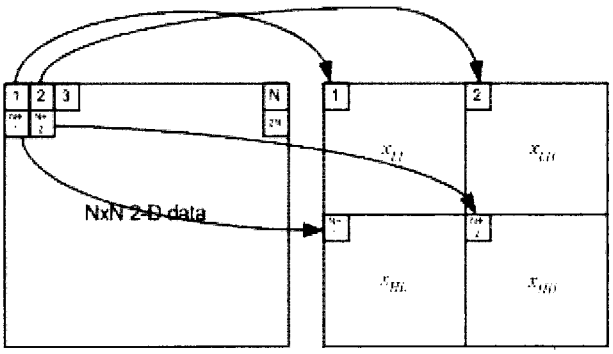


그림 3. 2차원 분할도
Fig. 3. The 2-D splitting diagram.

$$\hat{x}_{2n,2m+1} = x_{2n,2m+1} - (a_0 x_{2n,2m-4} + a_1 x_{2n,2m-2} + a_2 x_{2n,2m} + a_3 x_{2n,2m+2}) \tag{6}$$

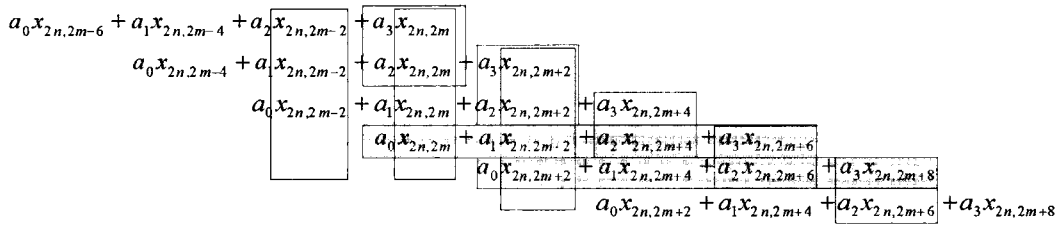


그림 6. 필터 연산과 데이터의 중복성
Fig. 6. The redundancy of the coefficient calculation and the data dependency.

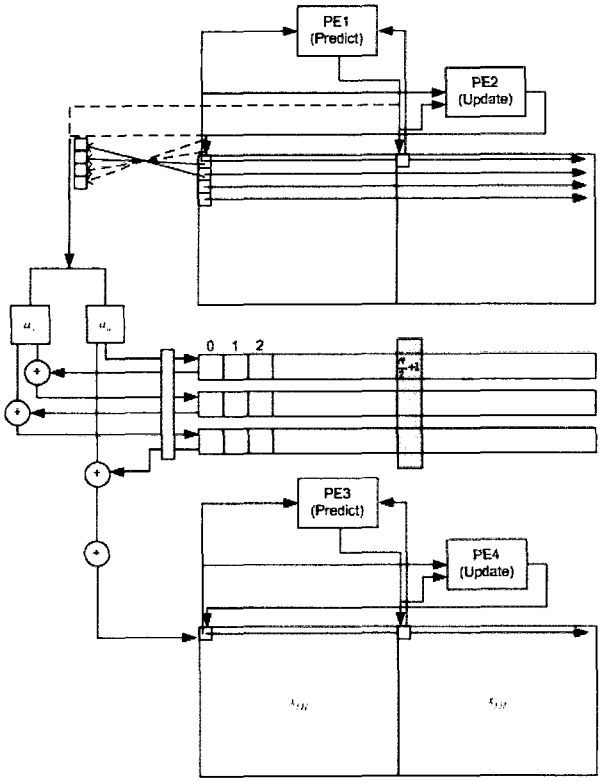


그림 4. 전체 하드웨어 구성도
Fig. 4. The configuration diagram of the overall hardware.

큼의 버퍼 메모리가 필요하게 된다. 제안하는 구조는 1쌍의 열 방향 필터 모듈을 추가하여 행 방향 연산의 결과를 바로 열 방향 연산의 수행의 입력으로 사용하고, 기존의 구조들이 내부 버퍼에 행 방향 연산의 결과를 저장하고 있는데, 이것을 열 방향 연산의 필터 중간 값들을 저장하도록 구성하여 열 방향의 연산이 갖는 단점을 개선하였다.

2 필터 모듈 구조

각 필터는 기본적으로 FIR (Finite Impulse Response) 필터로 구성되면 웨이블릿의 종류에 따라서 필터의 탭 수가 결정된다. 만약 5탭의 예측 필터를 사용하게 된다면 그림 5와 같이 하나의 예측 결과 값을 얻기 위해 5개의 데이터 값이 필요하게 된다. 5개의 데이터 값은 하나의 odd 위치의 데이터 값과 4개의 even 위치의 데이터 값을 사용하게 된다. 행 방향의 변환 과정을 수식으로 구성하면 식(6)과 같이 기술 할 수 있다. 이 식에서 n, m은 각각 $0 = n = N/2$ (N: image width) $0 = m = N/2$ (N: image height)의 범위를 갖고, $\hat{x}_{2n,2m+1}$은 예측 필터에 의해서 얻어 지는 변환의 결과값을 의미하게 된다.

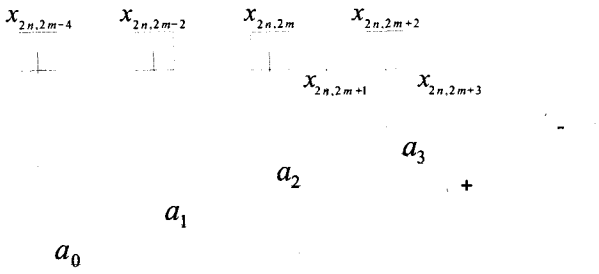


그림 5. 일반적인 5탭 FIR 필터의 데이터 흐름
Fig. 5. A figure of data flow of the general 5 tap FIR filter.

식(6)을 이용하여 연속되는 연산을 표현 하면 그림 6과 같다. 그림 6에서 매 연산의 입력 데이터는 다음의 연산의 수행을 위해서도 필요하게 되는데, 보통의 FIR 필터에서는 입력 데이터를 지연시켜 다음의 연산에 사용하게 된다. 이렇게 할 경우 입력 데이터를 저장하기 위한 버퍼가 필요하게 되고 필터연산에 필요한 모든 데이터가 들어 왔을 때 모든 계수 값을 이용하여 연산을 수행 하게 된다. 연산에서 필터의 탭수 만큼의 승산기와 덧셈기를 필로 한다. 리프팅 스킴의 웨이블릿 필터 계수의 특성상 계수들은 서로 대칭이 되는 구조를 갖게 되고, 그림 6에서와 같이 입력된 하나의 데이터는 결국 모든 필터의 계수와 곱해진 결과값이 필요하게 된다. 그런데, 리프팅 웨이블릿 변환 필터의 대칭 특성상 $a_0 = a_3$, $a_1 = a_2$의 특성을 갖는다. 즉 $a_0 x_{2n,2m}$은 새로 연산을 수행 하지 않고, 이미 저장 되어 있던 값 $a_3 x_{2n,2m}$을 사용할 수 있다. 즉 하나의 값이 들어와서 하나의 연산 결과를 출력하는 구조에서 모든 필터의 계수의 연산을 수행하기 위한 승산기가 필요한 것이 아니고, 그 절반에 해당하는 승산기만으로 필터 연산을 수행 할 수 있게 된다. 그리고, 각 연산의 수행과정에 있어서 하나의 필터 값을 계산하기 위해서는 그림 5와 같이 필터 탭수 만큼의 값이 필요하고 각 값을 지연시키기 위한 레지스터가 필요하게 된다. 본 구조에서는 입력 값을 지연시키는 것이 아니고, 하나의 입력 값이 들어 왔을 때 그 값이 사용되는 모든 출력의 경우를 차례로 연산하여 내부 레지스터에 저장하여 두게 된다.

다음의 그림 7은 일반적인 컨벌루션 필터의 연산을 수행하기 위한 하드웨어 구조와 그 하드웨어에 의해서 발생되는 중간 값에 대한 관계를 기술 하였다.

위에서 언급한 (9,7) 리프팅 웨이블릿 필터 계수에서 고주파 필터의 경우 2개의 승산기가 필요하게 된다. 행 방향 연산에 있어서 예측과 보상은 연속 되는 처리 구

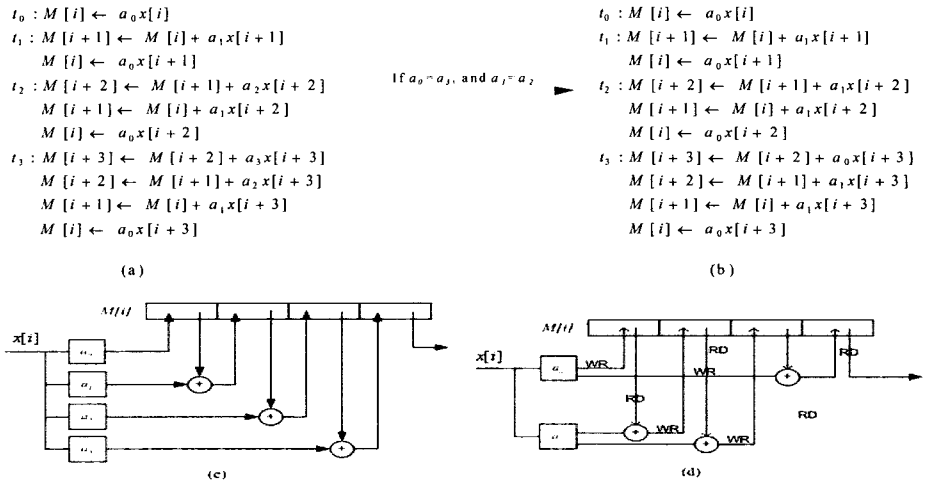


그림 7. 컨벌루션필터의 구조와 계수 대칭성을 이용한 필터 구조(a) 컨벌루션 필터을 이용하였을 때 메모리에 저장되는 값 (b) 계수 대칭성을 이용하였을 때 메모리에 저장되는 값 (c) 컨벌루션 필터 구조 (d) 계수 대칭성을 이용한 필터 구조

Fig. 7. The filter architecture of the convolution filter and the filter architecture using coefficients symmetric characteristics. (a) a memory value which is stored in the convolutionfilter architecture, (b) a memory value which is stored in the filter architecture using coefficients symmetric, (c) a convolution filter architecture (d) a filter architecture using coefficients symmetric characteristics.

조를 갖게 되는데, 이러한 구조를 이용하여 처리할 경우 불필요한 메모리 액세스를 줄일 수 있다. 그림 8은 필터 계수의 대칭성을 이용한 예측과 보상 필터의 전체 구조를 나타내고 있다. 그리고 보상 필터에는 예측의 필터 탭 수만큼의 FIFO를 이용하여 한번 읽어온 데이터를 이용하도록 구성하였다.

행 방향 필터의 경우 행 방향 데이터가 연속적으로 발생함으로 각 필터에 사용되는 버퍼 역시 행 방향으로 필터 탭수 만큼만을 가지고 있다. 보상 필터의 경우 입력으로 사용하여야 할 데이터가 예측 필터에 입력으로 들어갔던 데이터와 필터의 결과 같이 필요하게 된다. 그러므로 필터의 입력으로 들어간 데이터도 저장하여야 하고, 보상 필터 자체적으로 발생하는 지연을 저장하여야 하기 때문에 그림 8에서와 같이 입력 데이터를 저장하는 $L_h + L_g$ 만큼의 FIFO와 $L_g/2$만큼의 중간 값 저장 버퍼가 필요하다.

열 방향 필터의 경우 행 방향 필터와 유사한 하드웨어 구조를 갖는데, 열 방향 필터의 경우 입력 데이터의 값의 발생순서가 열 방향으로 발생하는 것이 아니고 행 방향으로 발생하는 문제가 있다. 기존의 하드웨어 구조에서는 행 방향 데이터를 저장하는 형태로 구성하여 $2(L_g + L_h)N$ 만큼의 버퍼를 사용한다. 제안하는 하드웨어에서는 버퍼 사이즈를 줄이기 위해 열 방향 필터 구조를 변경하여 버퍼 사이즈를 줄였다.

앞서 언급한 4분할 데이터를 이용한 행 방향 연산 결과를 $\hat{x}_{LL}$, $\hat{x}_{LH}$, $\hat{x}_{HL}$, $\hat{x}_{HH}$라고 했을 때, 열 방향 연산은 다음의 식(7)~(10)과 같이 표현 된다. 식(7)과 식(9)의 입력 데이터로 사용되는 $\hat{x}_{LL}$와 $\hat{x}_{LH}$는 행 방향 연산에서 시차를 두고 행 방향으로 연속적으로 발생하게 되고, $\hat{x}_{HL}$와 $\hat{x}_{HH}$도 동시에 발생하게 된다. 즉 4개의 입력 데이터가 같은 시간 지연을 가지고 동시에 발생하기 때문에 열 방향 연산은 2개씩의 데이터를 쌍으로 묶어서로 다른 클럭 천이를 이용하여 설계하고 두 개의 데이터 쌍을 저장하는 버퍼를 완전히 분리하여 설계하게 된다.

$$\tilde{x}_{HL} = \hat{x}_{HL} - P(\hat{x}_{LL}) \tag{7}$$

$$\tilde{x}_{LL} = \hat{x}_{LL} + U(\tilde{x}_{HL}) \tag{8}$$

$$\tilde{x}_{HH} = \hat{x}_{HH} - P(\hat{x}_{LH}) \tag{9}$$

$$\tilde{x}_{LH} = \hat{x}_{LH} + U(\tilde{x}_{HH}) \tag{10}$$

열 방향 연산에서 가장 큰 문제는 식(7)을 예로 들었

$$\tilde{x}_{HL}(n,m)=\hat{x}_{HL}(n,m)-(a_0\hat{x}_{LL}(n-2,m)+a_1\hat{x}_{LL}(n-1,m)+a_2\hat{x}_{LL}(n,m)+a_3\hat{x}_{LL}(n+1,m)) \quad (11)$$

을 때 하나의 결과 값을 다음의 식(11) 과 같이 표현 할 수 있다. 이 식에서 $x_{LL}(n-L,m)$, $\hat{x}_{LL}(n-1,m)$, $\hat{x}_{LL}(n,m)$, $x_{LL}(n+1,m)$는 연속적인 발생을 하는 것이 아니고, 각각의 값이 최소 N(data width) 클럭 사이클 만큼의 시차를 두고 발생하게 된다. 즉 결과 값을 열 방향으로 연속 적으로 연산하는 것이 아니고, 열 방향 연산도 행 방향으로 처리 하여야 한다.

제안하는 하드웨어에서는 행 방향 연산의 결과를 메모리에 담아 두는 것이 아니고, 이것을 바로 열 방향 연산 모듈의 입력으로 넣어 열 방향 연산의 중간 결과를 버퍼에 넣어 놓게 된다. 그리고, 식(11)에서 $\hat{x}_{HL}(n,m)$는 통상 마지막 단계에서 사용하게 되는데, 기존의 구조에서는 $\hat{x}_{HL}(n,m)$을 담아 놓기 위한 버퍼가 또 필요하게 된다. 그래서 제안하는 구조에서는 다음의 그림 9와 같이 입력이 들어 왔을 때 그 값을 바로 사용하여 연산을 수행하고 그 값을 버퍼에 담아 놓게 된다. 각 버퍼는 행 단위로 들어오는 데이터를 연산하여 하나의 버퍼에 다 저장하고 다음 행의 데이터가 들어 올 때 저장해 놓았던 값을 읽어 현재의 값과 연산을 수행하여 다음 단의 버퍼에 저장해 놓게 된다. 하나의 버퍼 메모리 뱅크는 입력이 들어는 행 방향의 데이터를 연산을 수행하여 저장해 놓게 되고, 전체 필터 탭수 만큼의 데이터 행이 처리되면 마지막 버퍼 메모리 뱅크에 최종 결과를 저장하게 된다.

3. 데이터의 스케쥴링

제안하는 구조의 1차원 웨이블릿 변환을 위한 데이터의 흐름을 행 방향 웨이블릿 변환을 수행할 때 데이터 입력과 출력 관계는 그림 10과 같다. 전체 4개의 입력을 받아서 그림 8과 같은 필터 구조에 입력으로 사용하게 된다. 행 방향 처리 속도를 향상 시킨 이유는 전체 처리 속도의 향상과 내부 버퍼 메모리 크기를 줄이기 위해서 필요한 열 방향 처리의 시작 지연을 줄이기 위해서 이다.

열 방향 웨이블릿을 처리하는데 있어서 가장 큰 문제점은 열 방향 데이터의 입력으로 사용되어야 할 행 방향처리 결과의 발생순서가 그림 11 (a)와 같이 행 방향으로 발생하고, 열 방향 처리를 위해 필요한 데이터의 순서는 그림 11(b)와 같이 열 방향의 데이터 입력이

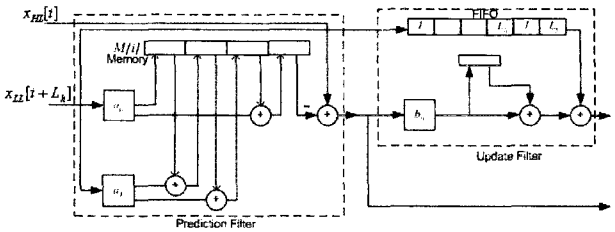


그림 8. (9, 7) 리프팅 스킴의 웨이브릿 예측 보상 필터 구조

Fig. 8. The prediction and update filter architecture of (9, 7) lifting scheme wavelet filter.

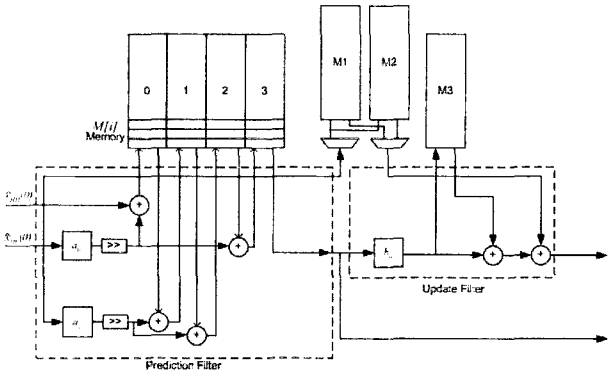


그림 9. 열 방향 웨이블릿 변환을 위한 필터 구조

Fig. 9. The filter architecture for row direction wavelet transform.

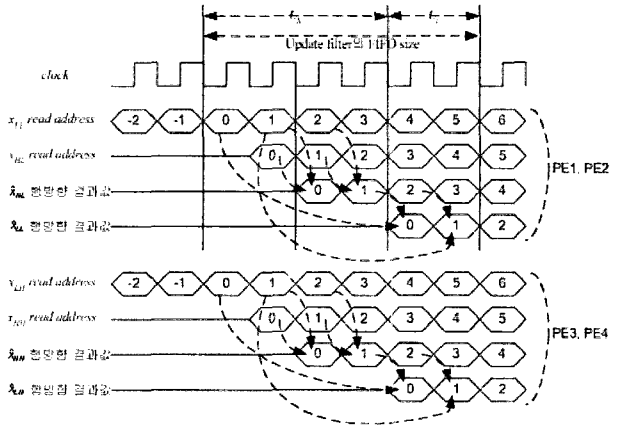


그림 10. 행 방향 예측 보상 필터인 PE1, PE2, PE3, PE4의 입력과 출력 데이터 타이밍도

Fig. 10. The data timing diagram of PE1 ~ PE4's input and output in the row direction prediction and update filter.

필요하게 된다.

열 방향의 웨이블릿 변환에 필요한 데이터가 그림 11(a)와 같이 행 단위로 나오기 때문에 열 방향 처리를 위해서는 *(L_g+L_h)N(L_h : High Pass Filter 탭수, L_g:*

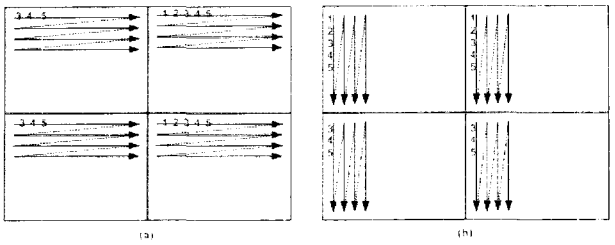


그림 11. 열 방향 연산을 위한 데이터 발생순서와 열 방향 연산에 필요한 데이터 순서(a) 열 방향 연산 모듈의 입력 데이터 발생순서, (b) 열 방향 연산에 필요한 데이터 순서

Fig. 11. The input data order for column direction transform, and ideal data order for column direction transform. (a) the input data order of column direction transform, (b) ideal data order for column direction transform.

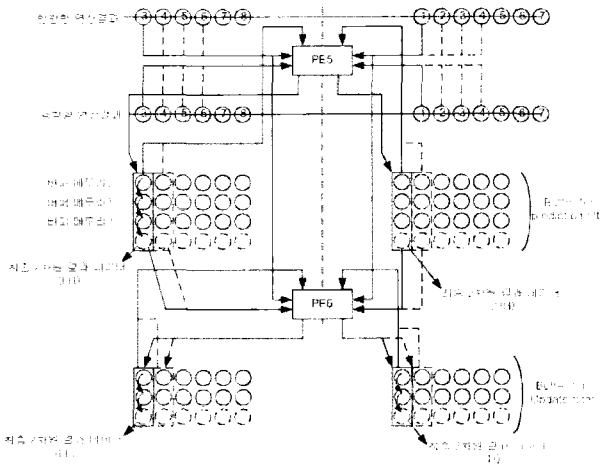


그림 12. 열 방향 연산의 데이터 흐름

Fig. 12. The data flow diagram of the column direction transform.

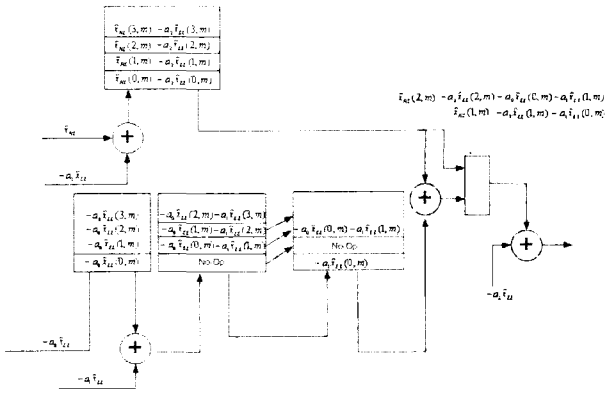


그림 13. 열 방향 연산의 메모리에 저장되는 데이터

Fig. 13. The data which is stored in the column direction filter memory buffer.

Low Pass Filter의 탭수, N : 행 데이터 개수) 만큼의 버퍼 메모리가 하다. 그런데 기존의 하드웨어 구조에서는 행 방향 처리에 1 쌍의 예측과 보상 필터 쌍 만을 이용하고, 열 방향 연산에 있어서 $2L_h$ 행만큼이 필요하므로 최소 $2(L_g+L_h)N$ 만큼의 버퍼를 한다. 제안하는 구조에서는 행 방향의 연산에 2쌍의 예측과 보상 필터를 할당함으로써 행 방향의 처리를 2배 향상 시키도록 구성하였다. 행 방향 연산의 결과는 그림 12와 같은 흐름으로 열 방향 연산의 입력으로 사용하게 된다.

전체 구조에서 볼 수 있듯이 행 방향 연산이 시작되면 4개의 결과 데이터가 나오게 되고 열 방향 연산 모듈의 입력으로 사용 된다. 하지만 열 방향 연산 모듈은 한 쌍의 예측과 보상 필터로 구성되어 있기 때문에 한번에 처리 할 수 있는 입력 데이터는 2개로 한정되어 있다. 결국 한쌍의 열 방향 처리 모듈을 추가 하든가 나머지 2개의 데이터를 저장하기 위한 버퍼 메모리를 필요로 한다. 그런데, 입력되는 데이터를 4개의 데이터가 서로 쌍으로 연관 관계를 갖고 있다. 제안하는 구조에서는 4개의 입력 데이터를 처리하기 위해 클럭의 상승과 하강 에지(rising and falling edges)를 모두 사용하는 구조를 구성하였다. 통상적인 동기 하드웨어 설계에 있어서 하나의 모듈은 클럭의 하나의 에지 만을 사용하고, 두개의 에지를 모두 사용할 경우 set-up hold 위반을 범할 우려가 많기 때문이다. 그래서 본 구조에서는 이러한 위험 요소를 없애기 위해 필터의 연산 모듈은 공유를 하지만 연산의 결과나 데이터를 저장하는 메모리 요소를 rising edge를 사용하는 부분과 falling edge를 사용하는 부분으로 완전히 분리하여 처리하게 된다. 즉 filter의 연산 모듈은 논리조합 회로로 구성되므로 이 회로의 최대 지연경로(critical path)가 클럭의 반주기 이내에 들어온다면 이 회로의 동작에 있어서 문제가 없음을 알 수 있다.

Ⅳ. 설계 결과 및 고찰

제안하는 구조는 VHDL로 기술하여 synplify pro를 이용하여 합성하여 Quartus II의 Altera Stratix device를 이용하여 P&R를 한 후 ModelSim을 이용하여 회로 지연을 고려한 post-layout simulation을 수행하였고, 동작을 확인하였다. 그 중 앞서 언급했던 열 방향 처리 모듈의 회로를 그림 14에 나타내었고, 그림 15에 ModelSim의 시뮬레이션 결과를 나타내었다. 그림 15에서 두개의 커서 사이의 간격이 필터 연산 회로의 지연을 나타내는 것으로 Stratix device를 이용한 경우 최대 지연 경로의 지연시간이 10ns의 결과를 얻을 수 있었

표 1. 기존 구조와 멀티플라이, 덧셈기 수, 내부 메모리 크기, 전체 연산시간의 비교표
Table 1. The comparison of the multiplier, adder resource, memory size, total timing to the previous architecture ($\alpha \ll N^2, \beta \ll \frac{N^2}{4}, \gamma \ll \frac{N^2}{2}$)

	Internal Memory	Hardware		Timing
		Multiplier	Adder	
ANDRA*[4]	$\frac{N^2}{2}+x$	$\frac{2}{3}(L_g+L_h)$	$2(L_g+L_h-2)$	$N^2+\alpha$
FERRETTI[10]	$2N(2L_g-1+L_h)$	$2(L_g+L_h)$	8	$\frac{N^2}{2}+\gamma$
proposed	$N(L_g-1+L_h)$	$\frac{3}{2}(L_g+L_h)$	$3(L_g+L_h-1)$	$\frac{N^2}{4}+\beta$

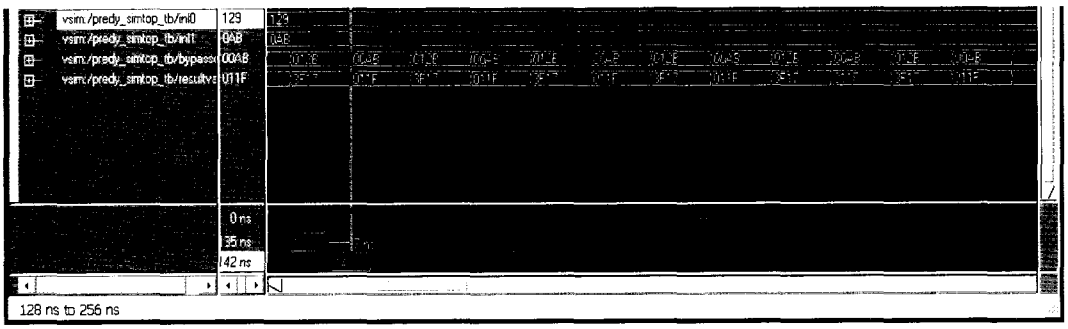


그림 15. 그림 14의 시뮬레이션 파형
Fig. 15. The simulation waveform of the fig. 14.

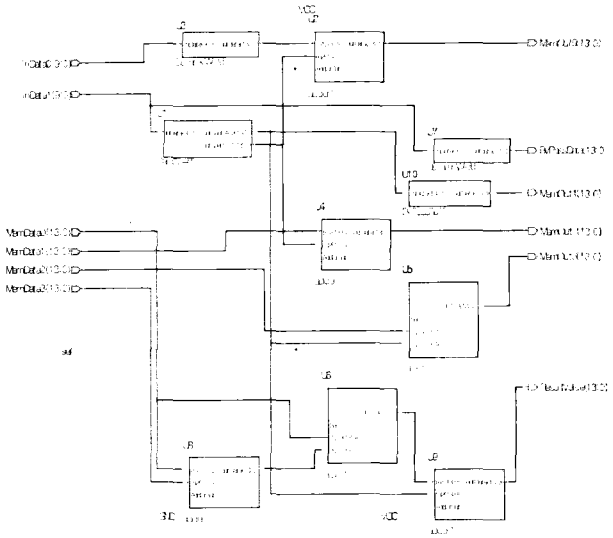


그림 14. 열 방향 예측 필터의 설계도
Fig. 14. The design diagram of the row direction update filter.

다. 즉 이 회로는 최대 50MHz까지 동작 시킬 수 있다. 제안하는 구조는 기존의 병렬 처리 구조에 비하여 하드웨어는 증가 하였지만 시스톨릭 어레이를 이용한 구조에 비하여서는 감소된 하드웨어 구조를 얻을 수 있었다. 그리고, 제안하는 구조는 새로운 분할 방법을 이용한 병렬 처리 효율을 증가시켜 내부 메모리 버퍼의 용량을 감소시켰으며, 초기 지연 시간을 줄일 수 있도록 1차원 DWT의 병렬 처리를 증가 시켰다. 그 결과 초기 지연시간을 50% 정도 감소시킬 수 있었다. 표 1은 기존의 구조와 제안하는 구조와의 멀티플라이어 수와 메모리 사이즈, 그리고, 지연시간에 대한 비교표이다. 표1. 에서 알 수 있듯이 하드웨어 자원의 멀티플라이어의 수가 기존의 병렬 처리 방법에 비하여 50% 정도 증가 하였다. 하지만 내부 메모리 버퍼 사이즈는 감소하였고 초기 병렬 처리 속도 향상과 자원 공유를 통한 2차원 DWT의 병렬 처리로 인해 전체 지연 시간을 기존의 방법의 1/4로 줄일 수 있었다.

V. 결 론

본 논문에서는 기존의 데이터 분할을 사용하지 않고, 2차원적인 4-way 데이터분할방법을 이용하여 하드웨어 병렬 처리 속도를 향상 시키고, 향상된 처리속도의 결과 내부 메모리를 감소시키는 데이터의 흐름과 하드웨어 구조를 제안하였다.

제안하는 4분할 기반의 데이터 흐름과 병렬 처리 연산 구조를 사용하여 전체 시스템을 구성하였을 때, 전체 연산 속도를 기존의 $N^2/2$에서 $N^2/4$로 100% 향상된 속도를 얻을 수 있고, 내부 버퍼 메모리 역시 기존

의 $2(L_h+L_g)N$에서 $(L_h+L_g)N$로 줄일 수 있게 되었다. 반면에 제안하는 하드웨어에서는 기존의 방법에 비해 연산에 필요한 하드웨어 코스트가 증가하지만 이것은 감소된 내부 버퍼 메모리의 하드웨어 코스트에 의 해 상쇄되고, 전체 적으로 메모리 하드웨어의 코스트 감소분이 기존의 하드웨어 구조에 비해 크게 나타남을 알 수 있었다.

그리고, 제안하는 구조를 VHDL을 이용하여 기술하고, Altera Stratix device를 이용하여 시뮬레이션한 결과 제안하는 구조는 최대 66MHz까지 동작 할 수 있음을 알 수 있었다.

참 고 문 헌

[1] ISO / IEC JTC1 / SC29 / WG11, 14496 - 2 : 2001 Information Technology-Coding of Audio-Visual Objects-Part 2: Visual.

[2] ITU-T Rec. T.800 FCD15444-1:2000 Information Technology JPEG2000 Image Coding System.

[3] M. Vishwanath, R.M. Owens, and M.N. Irwin, "VLSI Architecture for the Discrete Wavelet Transform," IEEE Trans. on Circuits and Systems-II: Analog and Digital Signal Processing, vol. 42, No. 5, pp. 305~316, 1995.

[4] Kishore Andra, Chaitali Chakrabarti, and Tinku Acharya, "A VLSI Architecture for Lifting-Based Forward and Inverse Wavelet Transform", IEEE Trans. on Signal Processing, vol. 50, No. 4, April, 2002, pp. 966~977.

[5] A. Carreira, T. W. Fox, "The Multiplier Tree FIR Filter Architecture," Field-Programmable Technology (FPT), 2003. Proceedings. 2003 IEEE International Conference, Dec. 2003, pp. 447~450.

[6] Chao-Tsung Huang, Po-Chih Tseng, and Liang-Gee Chen, "Flipping Structure: An Efficient VLSI Architecture for Lifting-Based Discrete Wavelet Transform," IEEE Trans. on Signal Processing, vol. 52, no. 4, April, 2004, pp. 1080~1089.

[7] Ingrid Daubechies, and Wim Sweldens, "Factoring wavelet transforms into lifting schemes," J. Fourier Anal. Appl., vol. 4, pp.247~269, 1998.

[8] Taegeun Park, and Sunkyung Jung, "High speed lattice based VLSI architecture of 2D discrete wavelet transform for real-time video signal processing," IEEE trans. on Consumer Electronics, vol. 48, no. 4, Nov. 2002, pp. 1026~1032.

[9] 김종욱, 정정화, "Lifting scheme을 이용한 고속 병렬 2D-DWT 하드웨어 구조,"대한전자공학회 논문지 SD편, 제40권, 7호, pp. 50~56, 2003.

[10] M. Ferretti, and D. Rizzo, "A parallel architecture for the 2-D discrete wavelet transform with integer lifting scheme," Journal of VLSI signal processing, vol 28, pp. 165~185, 2001.

저 자 소 개

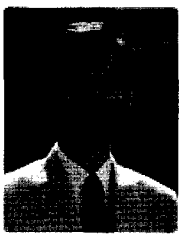
김 종 욱(학생회원)

1992년 한양대학교 전자공학과 학사 졸업.

1994년 한양대학교 대학원 전자공학과 석사 졸업.

2005년 한양대학교 전자공학과 박사 과정.

<주관심분야 : 영상 압축, 하드웨어 구조 설계, H.264, JPEG2000 >



정 정 화(평생회원)

1975년 한양대학교 전자공학과 학사 졸업

1977년 한양대학교 대학원 전자공학과 석사 졸업

1981년 일본 와세다대학교 대학원 전자공학과 박사 졸업

<주관심분야 : HW/SW Co-design, wireless communication system, MPEG encoder/decoder chip design>