한국산학기술학회논문지
Vol. 7, No. 4, pp. 625-634, 2006

필터 뱅크를 사용한 저전력 short-length running convolution 필터 설계 및 구현

장영범[1*]

Design and Implementation of low-power short-length running convolution filter using filter banks

Young-Beom Jang[1*]

요 약 이 논문에서는 FIR(Finite Impulse Response) 필터의 연산의 양을 줄이는 효율적인 직접방식의 고속 알고리즘을 제안하였다. 제안된 알고리즘은 임의의 다운샘플링 크기로 병렬화가 가능하며, 다운샘플링의 크기가 결정되면 쉽게 구조를 유도할 수 있다. 특히 제안된 알고리즘은 이론적인 샘플당 곱셈연산의 수를 감소시킴과 동시에 실제 구현에 있어서도 효과가 있음을 실험을 통하여 입증하였다. 이론적으로 연산의 양이 감소함을 보이기 위하여 부필터의 수와 샘플당 곱셈연산의 수를 기존의 고속 알고리즘과 비교하였으며, 실제적으로 구현의 효과를 입증하기 위하여 하드웨어 구현소자의 수와 Verilog-HDL (Hardware Description Language) 구현으로 기존의 방식들과 비교하여 제안된 구조가 효과적임을 보였다.

Abstract In this paper, an efficient and fast algorithm to reduce calculation amount of FIR(Finite Impulse Responses) filtering is proposed. Proposed algorithm enables arbitrary size of parallel processing, and their structures are also easily derived. Furthermore, it is shown that the number of multiplication/sample is remarkably reduced. For theoretical improvement, numbers of sub filters are compared with those of conventional algorithm. In addition to the theoretical improvement, it is shown that number of element for hardwired implementation are reduced comparison to those of the conventional algorithm.

Key Words : short-length FIR filter, running convolution, filter banks, pseudocirculant matrix

1. 서론

FIR(Finite Impulse Response) 필터에서 연산의 복잡도를 줄이기 위한 연구가 활발히 진행되어 왔다. 필터 연산의 복잡도를 줄이는 고속 알고리즘은 FFT(Fast Fourier Transform) 등의 주파수 영역에서 연산을 수행하는 간접방식과 시간영역에서 연산의 양을 줄이는 직접방식이 있다[1]. 그러나 고속 알고리즘들이 널리 사용되지 못하고 있는 것은 하드웨어 구현이 파이프라인 구조를 요구하므로 구현 면적이 커지게 되고, DSP(Digital Signal Processor)를 사용하는 소프트웨어 구현에서도 효과적이지 않기 때문이다. 다시 말하면 이와 같은 고속 알고리즘들의 연산량 감소가 비효율적인 구조의 대가로 얻어진 결과이기 때문에 효용성이 떨어지게 된다. 그러나 고속 알고리즘이 MAC(Multiplication and Accumulation)을 사용하는 프로세서로 효과적으로 구현될 수 있다면 연산량의 감소가 큰 의미가 될 수 있다. 따라서 이와 같은 MAC의 연산량을 감소시키는 직접방식의 고속 알고리즘들이 발표되었다[2]-[3]. 이와 더불어 FFT를 사용하는 간접방식이면서도 pseudocirculant 행렬을 사용하는 고속 알고리즘도 발표되었다[4]-[5]. 또한 고속 알고리즘을 응용하는 논문들도 발표되었다[7]-[8]. M. Vetterli[2]에서는 2의 다운 샘플링과 3채널의 필터 뱅크를 사용하여 연산량을 25% 감소시킨 구조를 제안하였다. 이 기본 구조는 3개의 부필터를 사용

이 논문은 2005년 상명대학교 교내연구비의 지원에 의하여 연구되었음.

[1]상명대학교 정보통신공학과
*교신저자: 장영범(ybjang@smu.ac.kr)

하는데, 이 부필터에 다시 고속 알고리즘을 반복하여 적용함으로서 연산량을 더욱 줄일 수 있는 방식이다. 그러나 이 알고리즘은 MAC 프로세서를 사용하여 구현하면 입력 샘플을 가공하여 저장하는 메모리의 크기가 매우 커지는 단점이 있다. Z. Mou[3]는 2의 다운샘플링 뿐 아니라 3, 5의 다운 샘플링을 사용하는 고속 알고리즘을 제안하였다. 이 방식도 pseudocirculant 행렬을 사용하는 직접방식으로서 입력 샘플의 병렬화를 통한 잉여연산을 제거하는 알고리즘이다. 그러나 이 논문은 병렬화를 하기 위하여 다운샘플링을 2, 3, 5의 수만을 사용하므로 2, 3, 5의 공배수가 아닌 수로 병렬화가 안 되는 단점이 있다. 본 논문에서는 설계자가 원하는 임의의 수로 병렬화하는 직접방식의 고속 알고리즘을 제안한다. 기존의 연구가 연산량을 감소시키는 것에 초점을 맞추었으나, 실제 구현시에는 구조가 복잡해지는 단점이 발생하였다. 그러나 제안된 알고리즘은 기존의 고속 알고리즘보다 곱셈연산, 덧셈연산, 지연소자 등의 구현 소자를 감소시킴과 동시에 반도체로 구현한 결과도 게이트 카운트가 감소됨을 보인다.

2. 제안된 short-length running convolution 알고리즘

2.1 데시메이션 3의 알고리즘

일반적으로 필터는 주파수 영역에서 다음과 같이 표현된다. 즉,

$$Y(z)=H(z)X(z) \tag{1}$$

이 식에서 $Y(z)$, $H(z)$, $X(z)$는 각각 출력신호 $y[n]$, 임펄스응답 $h[n]$, 입력신호 $x[n]$의 z변환이다. 다운 샘플링 3을 사용하는 알고리즘을 만들기 위하여 먼저 위의 입력, 출력, 전달함수 등의 3개의 다항식들을 다음과 같이 3의 polyphase로 분해한다.

$$\begin{aligned} X(z) &= X_0(z^3)+z^{-1}X_1(z^3)+z^{-2}X_2(z^3) \\ Y(z) &= Y_0(z^3)+z^{-1}Y_1(z^3)+z^{-2}Y_2(z^3) \\ H(z) &= H_0(z^3)+z^{-1}H_1(z^3)+z^{-2}H_2(z^3) \end{aligned} \tag{2}$$

이와 같은 3의 polyphase 분해식 (2)를 식 (1)에 대입하면 다음과 같은 z 영역에서의 입출력 관계식을 얻을 수 있다.

$$\begin{aligned} &Y_0+z^{-1}Y_1+z^{-2}Y_2 \\ &=[H_0X_0+z^{-3}H_2X_1+z^{-3}H_1X_2] \\ &+z^{-1}[H_1X_0+H_0X_1+z^{-3}H_2X_2] \\ &+z^{-2}[H_2X_0+H_1X_1+H_0X_2] \end{aligned} \tag{3}$$

이 식의 모든 다항식들은 (z^3)을 생략하고 표현하였다. 즉 H_0는 $H_0(z^3)$을 나타낸다. 이 식을 행렬식으로 표현하면 다음과 같이 나타낼 수 있다.

$$\begin{bmatrix} Y_0 \\ Y_1 \\ Y_2 \end{bmatrix} = \begin{bmatrix} H_0 & z^{-3}H_2 & z^{-3}H_1 \\ H_1 & H_0 & z^{-3}H_2 \\ H_2 & H_1 & H_0 \end{bmatrix} \begin{bmatrix} X_0 \\ X_1 \\ X_2 \end{bmatrix} \tag{4}$$

이 식의 우변을 pseudocirculant 행렬이라고 부르며 9개의 부필터를 사용하여 구현할 수 있음을 알 수 있다. 사용되는 부필터의 수를 줄이기 위하여 식(4)의 Y_2, Y_1, Y_0를 각각 다음과 같이 가공한다.

$$\begin{aligned} Y_2 &= H_2X_0+H_1X_1+H_0X_2 \\ &= H_1X_1+(H_0+H_2)(X_0+X_2) \\ &- H_0X_0-H_2X_2 \end{aligned} \tag{5a}$$

$$\begin{aligned} Y_1 &= H_1X_0+H_0X_1+z^{-3}H_2X_2 \\ &= z^{-3}H_2X_2+(H_0+H_1)(X_0+X_1) \\ &- H_0X_0-H_1X_1 \end{aligned} \tag{5b}$$

$$\begin{aligned} Y_0 &= H_0X_0+z^{-3}H_2X_1+z^{-3}H_1X_2 \\ &= H_0X_0+z^{-3}(H_1+H_2)(X_1+X_2) \\ &- z^{-3}H_1X_1-z^{-3}H_2X_2 \end{aligned} \tag{5c}$$

이와 같이 식 (5)를 사용하여 가공하면 H_0, H_1, H_2, H_0+H_1, H_1+H_2, H_0+H_2 등과 같이 6개의 필터를 사용하여 계산할 수 있음을 알 수 있다. 만약 원래의 주어진 필터가 60탭의 FIR 필터인 경우에 60개의 곱셈연산이 필요하다. (5)의 식을 사용하면 6개의 각각의 필터는 20탭의 필터이므로 총 120개의 곱셈연산이 요구된다. 그러나 3분의 1의 저속으로 동작하므로 샘플당 40개의 곱셈연산이 요구됨을 알 수 있다. 따라서 제안된 알고리즘

을 사용하면 곱셈연산의 양이 60개에서 40개로 20개가 감소됨을 알 수 있다. (5)의 식을 Toom-Cook 알고리즘을 사용하면 다음과 같이 나타낼 수 있다.

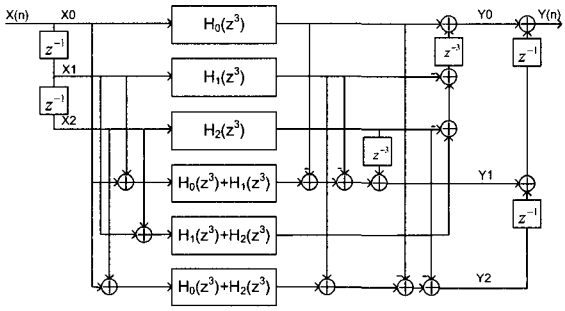
$$\begin{bmatrix} Y_0 \\ Y_1 \\ Y_2 \end{bmatrix} = C_3 \left\{ A_3 \begin{bmatrix} H_0 \\ H_1 \\ H_2 \end{bmatrix} * A_3 \begin{bmatrix} X_0 \\ X_1 \\ X_2 \end{bmatrix} \right\} \tag{6}$$

이 식에서 *는 내적을 의미하며 A_3와 C_3는 각각 다음과 같다.

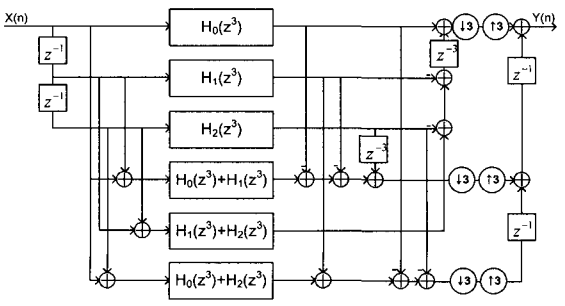
$$A_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix},$$

$$C_3 = \begin{bmatrix} 1 & -z^{-3} & -z^{-3} & 0 & 0 & z^{-3} \\ -1 & -1 & z^{-3} & 1 & 0 & 0 \\ -1 & 1 & -1 & 0 & 1 & 0 \end{bmatrix} \tag{7}$$

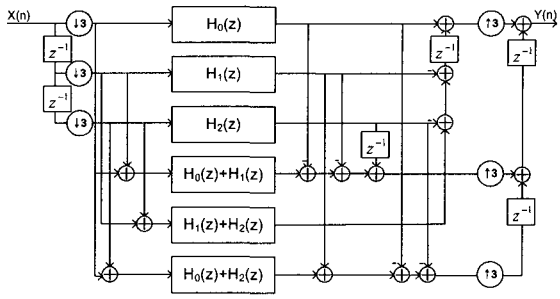
A_3의 행의 수는 사용되는 부필터의 수를 나타내며, C_3는 출력을 표시하기 위한 부필터의 조합을 나타내고 있다. 식 (6)과 (7)을 사용하여 필터 구조를 만들면 그림 1(a)와 같다.



(a)



(b)



(c)

그림 1. 제안된 다운샘플링 3의 구조, (a)기본구조, (b)샘플러 삽입 구조, (c)최종구조

그림 1(a)에서 $Y_0(z^3)$, $Y_1(z^3)$, $Y_2(z^3)$는 모두 제로 샘플들이 삽입된 신호이므로 3의 다운샘플러와 3의 업샘플러를 그림 1(b)와 같이 삽입할 수 있다. 그리고 그림 1(b)에서 z^{-3}의 필터와 3의 다운샘플러는 Noble Identity를 사용하여 서로 위치를 바꿀 수 있으므로 최종적으로 그림 1(c)의 효과적인 구조를 유도할 수 있다. 그림 1(c)에서 보듯이 입력 샘플들은 다운샘플러와 지연소자를 사용하여 3분의 1의 저속으로 감속되며, 출력 단에서는 지연소자와 업샘플러로 출력이 한 개씩 모아진다.

2.2 데시메이션 4의 알고리즘

다운샘플링 4를 사용하는 알고리즘을 만들기 위하여 식(1)을 4의 polyphase로 분해한 뒤에 입출력 관계를 pseudocirculant 행렬로 표현하면 다음과 같이 나타낼 수 있다.

$$\begin{bmatrix} Y_0 \\ Y_1 \\ Y_2 \\ Y_3 \end{bmatrix} = \begin{bmatrix} H_0 & z^{-4}H_3 & z^{-4}H_2 & z^{-4}H_1 \\ H_1 & H_0 & z^{-4}H_3 & z^{-4}H_2 \\ H_2 & H_1 & H_0 & z^{-4}H_3 \\ H_3 & H_2 & H_1 & H_0 \end{bmatrix} \begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \end{bmatrix} \tag{8}$$

이 행렬식의 모든 엘리먼트들도 역시 (z^4)을 생략하고 표현하였다. 위의 행렬식에서 보듯이 16개의 부필터가 필요함을 알 수 있다. 이 식의 사용되는 필터의 수를 줄이기 위하여 먼저 Y_3을 다음과 같이 가공할 수 있다.

$$\begin{aligned} Y_3 &= H_3X_0 + H_2X_1 + H_1X_2 + H_0X_3 \\ &= (H_0+H_1+H_2+H_3)(X_0+X_1+X_2+X_3) \\ &\quad -(H_0+H_1)(X_0+X_1)-(H_2+H_3)(X_2+X_3) \\ &\quad -(H_0+H_2)(X_0+X_2)-(H_1+H_3)(X_1+X_3) \\ &\quad +H_0X_0+H_1X_1+H_2X_2+H_3X_3 \end{aligned} \tag{9a}$$

위의 식에서 보듯이 Y_3을 만들기 위해 사용된 부필터의 수는 모두 9개이며 다음의 Y_2, Y_1, Y_0도 모두 Y_3에 사용된 부필터들의 조합으로 구성할 수 있다.

$$\begin{aligned} Y_2 &= H_2X_0 + H_1X_1 + H_0X_2 + z^{-4}H_3X_3 \\ &= H_1X_1 + z^{-4}H_3X_3 + (H_0+H_2)(X_0+X_2) \\ &\quad - H_0X_0 - H_2X_2 \end{aligned} \tag{9b}$$

$$\begin{aligned} Y_1 &= H_1X_0 + H_0X_1 + z^{-4}H_3X_2 + z^{-4}H_2X_3 \\ &= (H_0+H_1)(X_0+X_1) \\ &\quad + z^{-4}(H_2+H_3)(X_2+X_3) \\ &\quad - H_0X_0 - H_1X_1 - z^{-4}H_2X_2 - z^{-4}H_3X_3 \end{aligned} \tag{9c}$$

$$\begin{aligned} Y_0 &= H_0X_0 + z^{-4}H_3X_1 + z^{-4}H_2X_2 + z^{-4}H_1X_3 \\ &= H_0X_0 + z^{-4}H_2X_2 \\ &\quad + z^{-4}(H_1+H_3)(X_1+X_3) \\ &\quad - z^{-4}H_1X_1 - z^{-4}H_3X_3 \end{aligned} \tag{9d}$$

식 (9a)-(9d)와 같이 가공하면 9개의 부필터를 사용하여 출력을 계산할 수 있음을 알 수 있다. 만약 원래의 주어진 필터가 60탭의 FIR 필터인 경우에 위의 (9a)-(9d)의 식을 사용하면 9개의 각각의 부필터는 15탭의 필터이므로 총 135개의 곱셈연산이 요구된다. 그런데, 4분의 1의 저속으로 동작하므로 샘플당 33.75개의 곱셈연산이 요구됨을 알 수 있다. 따라서 제안된 알고리즘을 사용하면 샘플당 곱셈연산의 양이 60개에서 33.75개로 감소됨을 알 수 있다. (9a)-(9d)의 식을 Toom-Cook 알고리즘을 사용하면 다음과 같이 나타낼 수 있다.

$$\begin{bmatrix} Y_0 \\ Y_1 \\ Y_2 \\ Y_3 \end{bmatrix} = C_4 \left\{ A_4 \begin{bmatrix} H_0 \\ H_1 \\ H_2 \\ H_3 \end{bmatrix} * A_4 \begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \end{bmatrix} \right\} \tag{10}$$

이 식에서 A_4와 C_4는 각각 다음과 같다.

$$A_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} \tag{11}$$

$$C_4 = \begin{bmatrix} 1 & -z^{-4} & z^{-4} & -z^{-4} & 0 & 0 & 0 & z^{-4} & 0 \\ -1 & -1 & -z^{-4} & -z^{-4} & 1 & 0 & 0 & 0 & z^{-4} \\ -1 & 1 & -1 & z^{-4} & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & -1 & -1 & 1 & -1 & -1 \end{bmatrix} \tag{12}$$

A_4의 행의 수는 9이므로 9개의 부필터가 사용됨을 알 수 있으며, C_4는 출력을 얻기 위한 사용된 부필터의 조합을 나타내고 있다. 식 (10)-(12)를 사용하여 필터 구조를 만들면 그림 2와 같다.

$Y_0(z^4)$, $Y_1(z^4)$, $Y_2(z^4)$, $Y_3(z^4)$는 모두 제로 샘플들이 삽입된 신호이므로 4의 다운샘플러와 4의 업샘플러를 삽입할 수 있으며, Noble Identity를 사용하여 최종적으로 그림 2의 효과적인 구조를 유도할 수 있다. 그림 2에서의 입력 샘플들은 다운샘플러와 지연소자를 사용하여 4분의 1의 저속으로 감속된 것들이며, 출력들은 지연소자와 업샘플러로 출력을 한 개씩 모아야 완전한 출력신호가 만들어진다.

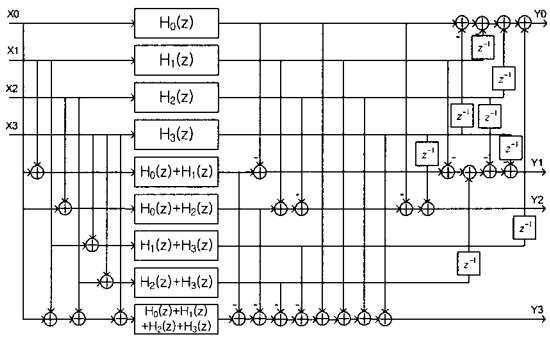


그림 2. 제안된 다운샘플링 4의 구조

3. 연산의 양 비교

3.1 부필터의 수 비교

탭의 길이가 정해진 FIR 필터를 부필터를 사용하여 구

현할 때에 부필터의 수가 적을수록 곱셈연산의 양이 감소하게 된다. 비교를 위하여 N=2부터 4까지의 병렬화에서 필요한 부필터의 수를 비교해보기로 한다.

표 1. 제안된 방식의 부필터의 수 비교
(N: 다운샘플링 수)

N	방식 1	방식 2	제안 방식	제안 방식의 %
2	4	3	3	100
3	9	7	6	85.7
4	16	13	9	69.2

비교를 위하여 pseudocirculant 행렬을 사용하는 3개의 고속 알고리즘들을 살펴본다. 즉, pseudocirculant 행렬을 변형 없이 그대로 사용하는 방식 1과 [4]에서 제안한 고속 알고리즘을 사용하는 방식 2와 본 논문이 제안하는 방식의 부필터 수를 비교하면 표 2와 같다. 방식 2는 부필터의 수를 N(N-1)+1로 나타낼 수 있다. 표 2에서 제안 방식의 %는 방식 2에 대한 %를 나타낸다. 표 2에서 보듯이, 2부터 8까지의 다운샘플링을 방식 2와 비교한 결과 부필터의 수를 평균적으로 28.3% 감소시킬 수 있었다.

3.2 샘플당 곱셈연산의 수 비교

고속 알고리즘의 성능을 평가하는 방법은 부필터의 수 이외에도 여러 가지가 가능하다. 이 중에서 샘플당 곱셈연산의 수를 평가해 보기로 한다. 즉, 제안된 고속 알고리즘에 대하여 2부터 4까지 다운샘플링 구조의 샘플당 곱셈연산을 비교하면 표 3과 같다. 표 3에서 곱셈연산의 양을 정량적으로 비교하기 위하여 12탭의 FIR 필터를 사용하여 비교하였다.

표 2. 제안된 방식의 샘플당 곱셈연산의 수
(12탭의 FIR 필터)

다운샘플링	부필터수	부필터의 탭수	총 곱셈연산	샘플당 곱셈연산	%
1	1	12	12	12	100
2	3	6	18	9	75.0
3	6	4	24	8	66.67
4	9	3	27	6.75	56.25

표 3에서 보듯이, 제안된 구조의 2부터 4까지의 다운샘플링을 비교한 결과 최대 샘플당 곱셈연산의 양을 43.25% 감소시킬 수 있었다.

3.3 하드웨어 구현 소자의 수 비교

3.1과 3.2의 절에서는 이론적인 수치인 부필터의 수와 샘플당 곱셈연산의 양을 비교하였다. 이 절에서는 다운샘플링 3을 사용한 구조를 통하여 필터의 수, 샘플당 곱셈연산의 수, 덧셈연산의 수, 지연소자의 수를 비교함으로써 좀 더 실제적인 구현 면적을 비교해보기로 한다. 예제로서 3탭의 FIR 필터를 사용하였다. 3탭의 필터를 비교하면 필터의 수가 곱셈연산의 수가 되므로 하드웨어 구현 면적을 쉽게 비교할 수 있다. 비교 대상은 pseudocirculant 행렬을 사용하는 3개의 고속 알고리즘들을 비교한다. 즉, pseudocirculant 행렬을 변형 없이 그대로 사용하는 방식 1과 Z. Mou[3]가 제안한 고속 알고리즘을 사용하는 방식 3과 본 논문이 제안하는 방식의 구현 소자 수를 비교하면 다음과 같다.

표 3. 제안된 방식의 구현 소자 수 비교

N=3	방식 1	방식 3	제안 방식
곱셈기수	9	6	6
덧셈기수	8	10	12
지연소자의 수	5	10	6
다운샘플러의수	3	3	3
업샘플러의 수	3	3	3

표 4에서 보듯이, 방식 3이나 제안 방식 모두 6개의 부필터를 사용하므로 부필터마다 곱셈기를 따로 사용한다고 가정하면 6개의 곱셈기가 필요하다. 제안 방식의 지연소자 수는 방식 3에 비하여 4개가 감소하였으나, 덧셈기의 수는 2개가 증가하였다.

3.4 Verilog-HDL 구현 비교

고속 알고리즘들은 다운샘플링을 통한 병렬화로 샘플당 곱셈연산의 수는 감소시킬 수 있으나, 구현 면적이 확실히 줄어드는지는 알 수 없다. 그러므로 Verilog-HDL 구현을 통해 구현 면적과 구조의 효용성을 알아볼 필요가 있다. 구현에 사용된 Tool은 Xilinx ISE 6.2i를 사용하여 시뮬레이션 하였다. 시뮬레이션 필터의 스펙은 탭 수 36탭, Sampling Frequency 8000Hz, Cutoff Frequency 4800Hz, Passband ripple -0.1dB, Stopband ripple -40dB로 하여 Filter coefficient를 구했다. 36탭 필터에 대하여 다양한 방식의 고속 알고리즘을 Verilog-HDL로 구현하였다. scalar 필터의 RTL구조는 그림 3과 같다.

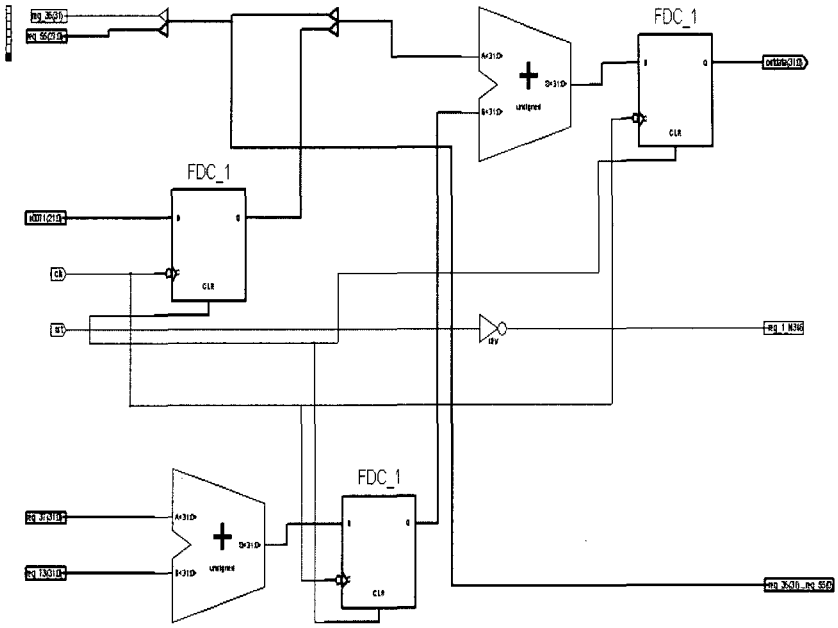


그림 3. Scalar RTL

scalar 구조 Hardwired 구현 결과를 보면 Registers : 106을 사용하였고 그 중 16-bit register : 35, 32-bit register : 71을 사용하였다. Adders/Subtracters는 총 35개를 사용하였으며 32-bit adder를 사용하였다. 면적에 가장 큰 부분을 차지하는 Multipliers는 총 36개를 사용하였다. 사용은 각각 다음과 같다.

16x10-bit multiplier: 3　　16x11-bit multiplier: 2
16x12-bit multiplier: 2　　16x15-bit multiplier: 2
16x16-bit multiplier: 17　　16x6-bit multiplier: 4
16x7-bit multiplier: 4　　16x9-bit multiplier: 2

앞의 결과들을 총 종합하여 게이트 카운트를 보면 셀 카운트가 15700셀이 나온다. 아래의 표는 셀의 사용을 보여준다.

표 4. Scalar filter Cell Usage

GND	1	XORCY	5110
LUT1	307	Flipflop/Latches	2636
LUT1_L	350	FDC_1	2636
LUT2	1346	Clock Buffers	1
LUT2_L	3174	BUFGP	1
LUT3	10	IO Buffers	49
LUT3_L	129	IBUF	17
MUXCY	5273	OBUF	32

96개의 입력을 넣으면 아래 그림 4와 같은 결과가 나온다.

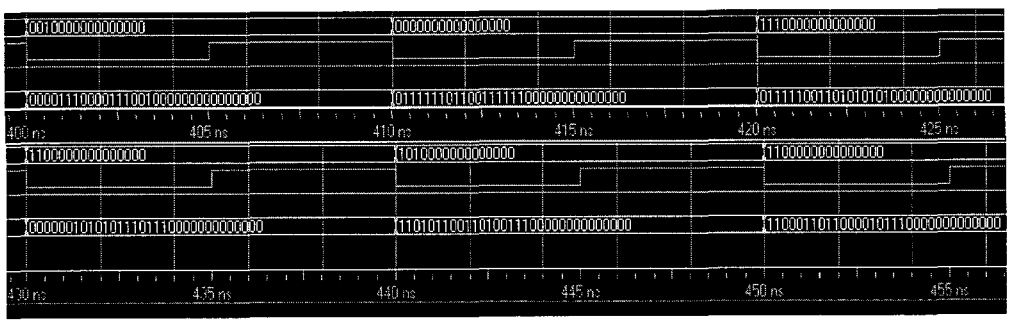


그림 4. scalar 필터 시뮬레이션 결과

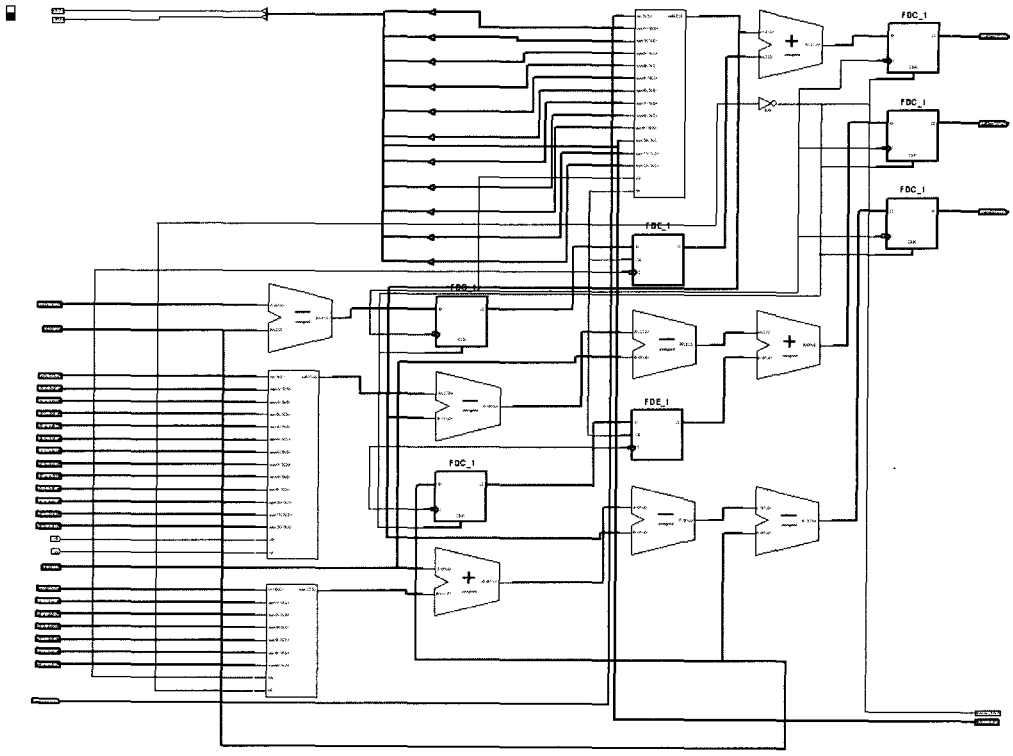


그림 5. 3구조 RTL

scalar 필터는 12개의 출력이 나오는데 120ns가 걸렸다. 다음으로 3구조의 시뮬레이션을 살펴본다. 3구조는 부필터 h1과 h0+h2는 FIR 필터의 선형 위상특성을 만족시키는 필터의 계수가 대칭을 이룬다. 그래서 부필터 h1과 h0+h2는 전치 직접형(transposed direct form)구조를 사용하여 구현하였고 나머지 부필터는 직접형(direct form)구조를 사용하여 구현하였다. 다음 그림 5는 3구조의 RTL을 보여 주고 있다.

3구조의 Final Report를 보면 Registers : 190을 사용하였고 그 중 16-bit register : 43, 32-bit register : 147을 사용하였다. Adders/Subtracters는 총 78개를 사용하였으며 32-bit adder를 사용하였다. 면적에 가장 큰 부분을 차지하는 Multipliers는 16x16-bit multiplier를 사용하였고 총 52개를 사용하였다. 앞의 결과들을 종합하여 게이트 카운트를 보면 셀 카운트가 26042셀이 나온다. 아래의 표는 셀의 사용을 보여준다.

표 5. 3구조의Cell Usage

GND	1	XORCY	8482
LUT1	470	Flipflop/Latches	5218
LUT1_ L	564	FDC_1	2636
LUT2	2814	Clock Buffers	1
LUT2_ D	1	BUF	1
LUT2_ L	2721	BUFGP	1
LUT3	50	IO Buffers	145
LUT3_ L	176	IBUF	49
LUT4	710	LUT4_ L	1330
MUXCY	8721	OBUF	96
vcc	1		

96개의 입력을 넣으면 아래 그림 6과 같은 결과가 나온다.

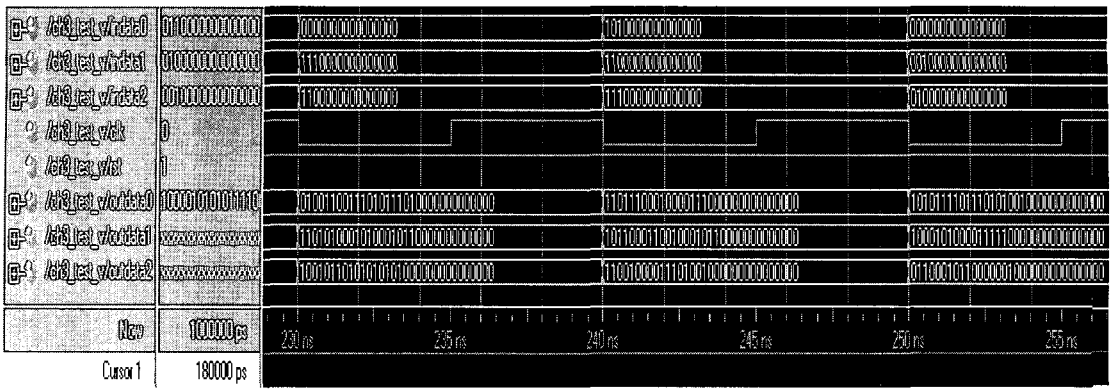


그림 6. 3구조의 시뮬레이션 결과

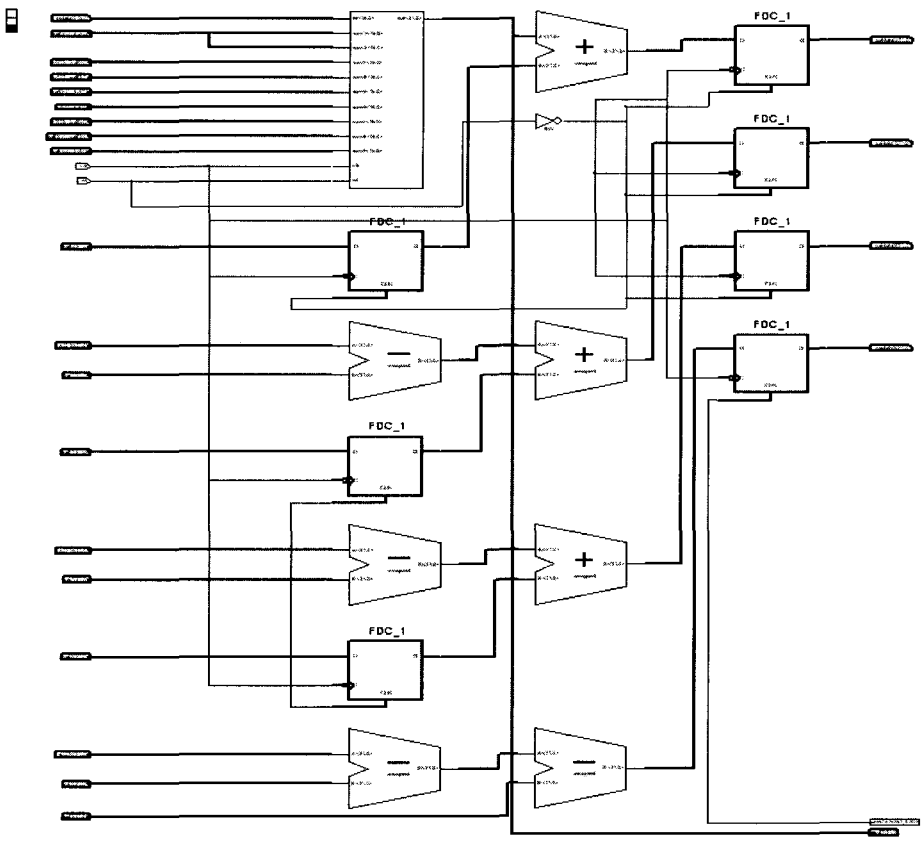


그림 7. 4구조 RTL 구조

3구조에서는 시스템이 안정화 상태에 들어간 후에 12개의 출력이 나오는데 40ns의 시간이 걸림을 알 수 있다. 각각의 구조의 알고리즘의 성능을 비교하기 위한 논문이므로 scalar 필터와 동일 조건으로 구현을 하면 셀 카운트가 29049인 곱셈기는 60개를 사용하였다. 마지막으로 4구조의 결과를 보인다. 4구조는 9개의 부필터를 사용하여 구현한다. 4구조의 RTL은 다음 그림 7과 같다.

4구조는 9개의 부필터를 사용한다. 각각은 9탭으로 구성되어있다. 4구조의 시뮬레이션 결과를 보면 Registers : 241을 사용하였고 그 중 16-bit register : 77, 32-bit

register : 164를 사용하였다. Adders/Subtracters 는 총 99개를 사용하였으며 16-bit adder : 6, 32-bit adder : 81, 32-bit subtracter : 12를 사용하였다. Multipliers는 16x16-bit multiplier를 사용하여 총 81개를 사용하였다. 앞의 결과들을 종합하여 게이트 카운트를 보면 셀 카운트가 35580셀이 나온다. 아래의 표는 셀의 사용을 보여준다.

표 6. 4구조의Cell Usage

GND	1	XORCY	11574
LUT1	1354	Flipflop/Latches	5869
LUT1_ L	53	FDC_1	5869
LUT2	6904	Clock Buffers	1
LUT2_ D	2	LUT4_ D	1
LUT2_ L	459	BUFGP	1
LUT3	303	IO Buffers	193
LUT3_ L	23	IBUF	65
LUT4	2937	LUT4_ L	12
MUXCY	11924	OBUF	128
vcc	1		

4구조의 출력 결과를 살펴보면 그림 8과 같다.

4구조는 안정화 상태에 들어간 후에 12개의 입력을 관찰하였을 때 30ns의 시간이 걸림을 알 수 있다.

시뮬레이션을 통해 각 필터구조에서 12개의 출력이 나오는 동안 걸리는 시간을 측정하였고, 구조들의 상대 전력 소모를 비교하기 위하여 다음의 식을 사용하였다.

$$P_{dyna} = P_t \cdot C_L \cdot V_{dd}^2 \cdot f_{clk} \tag{13}$$

위 식을 본 논문의 상대 전력 소모에 적용하기 위하여 다음과 같이 변형하였다.

$$P_{dyna} = \sum(\text{동작속도} \times \text{면적}) \tag{14}$$

구현 면적과 전력소모율을 비교한 결과는 표 8과 같다. 고속 알고리즘은 scalar 필터에 비해 구현 면적은 크지만 전력 소모율이 감소하였음으로 더 효율적이라 할 수 있다. 또한 다운샘플링 3을 사용하는 제안 방식과 방식 3을 비교한 결과 제안한 방식이 구현 면적과 전력 소모율이 적음으로 더 효율적인 구조라 할 수 있다. 다운샘플링 4를 사용하는 제안 구조가 전력 소모가 가장 적음을 알 수 있다.

표 7. HDL 구현 면적과 전력 소모 비교

다운샘플링	면적 (Cell Usage)	동작 속도(ns)	전력 소모(%)	비고
scalar 필터	15700(100%)	120	100	기준
3 (방식 3)	31540(201%)	40	67	전력 33%감소
3 (제안 방식)	29049(185%)	40	62	전력 38%감소
4 (제안 방식)	35580(226%)	30	57	전력 43%감소

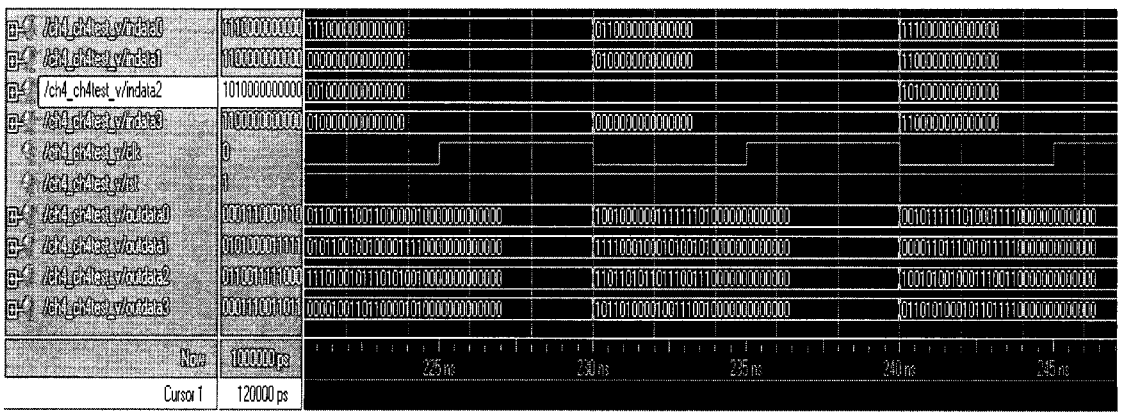


그림 8. 4구조의 시뮬레이션 결과

4. 결 론

본 논문에서는 FIR 필터의 연산의 양을 줄이는 효율적인 직접방식의 고속 알고리즘을 제안하였다. 제안된 알고리즘은 체계적인 방법으로 임의의 다운샘플링 크기로 병렬화하는 것이 가능하다. 따라서 다운샘플링의 크기, 즉 병렬화의 크기가 결정되면 본 논문에서 제안된 알고리즘을 사용하여 쉽게 구조를 유도할 수 있게 되며, 부필터의 수도 간단히 구할 수 있는 관계식을 유도하였다. 제안된 고속 알고리즘의 효율성을 검증하기 위하여 부필터의 수, 샘플당 곱셈연산의 수, 하드웨어 구현소자의 수, 그리고 MAC 프로세서를 사용한 구현 등 4가지 측면에서 비교하였다. 첫 번째로 부필터의 수 실험에서 N=8의 병렬화 경우, 기존의 방식과 비교하여 42.1%의 감소효과를 보였다. 두 번째의 샘플당 곱셈연산의 수에 대한 실험에서는 N=8의 병렬화 경우에 기본적인 N=1의 필터와 비교하여 49.93%의 샘플당 곱셈연산 감소효과를 보였다. 세 번째의 하드웨어 구현 소자의 수 실험에서는 N=3의 병렬화 경우에 대하여 기존의 구조와 비교하였다. 실험결과 곱셈기의 수는 동일하며, 지연소자의 수는 10개에서 6개로 감소하였으며 덧셈기의 수는 10개에서 12개로 증가함을 보였다. 마지막으로 Verilog-HDL 구현을 통하여 효용성을 실험하였다. 이와 같은 네 가지 실험결과를 통하여 제안된 구조가 일반적인 필터 구현에 효과적으로 널리 사용될 수 있음을 입증하였다.

참고문헌

[1] S. Winigrad, "Arithmetic complexity of computations," CBMS-NSF Regional Conf. Series in Applied Mathematics, SIAM Pub. 33, 1980.

[2] M. Vetterli, "Running FIR and IIR filtering using multirate filter banks," IEEE Trans. Acoust., Speech, Signal Processing, vol. 36, pp. 730-738, May 1988.

[3] Z. Mou, and P. Duhamel, "Short-length FIR filters and their use in fast nonrecursive filtering," IEEE Trans. Signal Processing, vol. 39, pp. 1322-1332, Jun. 1991.

[4] M. Teixeira, and D. Rodriguez, "A class of fast cyclic convolution algorithms based on block pseudocirculants," IEEE Signal Processing Letters, vol. 2, pp. 92-94, May 1995.

[5] M. Teixeira, and D. Rodriguez, "A New Method Mathematically Links Fast Fourier Transform Algorithms with Fast Cyclic Convolution Algorithms," Proc. 37th Midwest Symposium on Circuit and Systems, Lafayette, Lousiana, Aug. 1994.

[6] E. Vijay, K. Madisetti, and Douglas B. Williams, "Digital Signal Processing Handbook," chapter 8, CRC Press LLC, 1999.

[7] B. L. Eaton, R. J. Frank, L. Bolinger, and T. J. Grabowski, "Flexible algorithm for real-time convolution supporting dynamic event-related fMRI," Proc. of SPIE, pp. 452-459, April. 2002.

[8] 이채욱, 오신범, "적응 빔포밍 기법을 적용한 보청기 시스템의 성능 향상에 관한 연구," 한국통신학회논문지 '04-5 Vol.29 No.5C pp. 673-682, 2004년 5월.

장 영 범(Young-Beom Jang) **[정회원]**

- 1981년 2월 : 연세대학교 전기공학과 (공학사)
- 1990년 : Polytechnic University 대학원 졸업 (공학석사)
- 1994년 : Polytechnic University 대학원 졸업 (공학박사)
- 1983년 ~ 1999년 : 삼성전자 System LSI 사업부 수석연구원
- 2002년 3월 ~ 현재 : 상명대학교 정보통신공학과 교수

<관심분야>
통신신호처리, 비디오신호처리, SoC 설계