논문 2005-42SP-6-21

필터 뱅크를 사용한 효율적인 short-length running convolution 알고리즘

(Efficient short-length running convolution algorithm using filter banks)

장 영 범*, 오 세 만**, 이 원 상**

(Young-Beom Jang, Se-Man Oh, and Won-Sang Lee)

요 약

이 논문에서는 FIR 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 MAC 프로세서를 사용한 소프트웨어 구현으로 역시 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 reduced, and number of instructions using MAC(Multiplication and Accumulation) processor are also 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.

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

I. 서 론

FIR 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]. 특히 [2]에서는 2의 다운 샘플링과 3 채널의 필터뱅크를 사용하여 연산량을 25% 감소시킨 구조를 제안하였다. 이 기본 구조는 3개의 부필터를 사용하는데, 이 부필터에 다시 고속 알고리즘을 반복하여 적용함으로서 연산량을 더욱 줄일 수 있는 방식이다.

* 정회원, 상명대학교 공과대학 정보통신공학과
(College of Engineering, Sangmyung University)
** 학생회원, 상명대학교 대학원 컴퓨터정보통신공학과
(Graduate School, Sangmyung University)
※ 본 논문은 정보통신부의 IT SoC 핵심설계인력양성 사업(한국소프트웨어진흥원)으로 수행한 연구결과입니다.
접수일자: 2005년5월20일, 수정완료일: 2005년9월22일

그러나 이 알고리즘은 MAC 프로세서를 사용하여 구현하면 입력 샘플을 가공하여 저장하는 메모리의 크기가 매우 커지는 단점이 있다. [3]에서는 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. 제안된 알고리즘은 기존의 고속 알고리즘보다 곱셈연산, 덧셈연산, 지연소자 등의 구현 소자를 더욱 감소시킨 구조임을 보인다. 이와 더불어 DSP를 사용하여 구현할 때에도 효과적으로 구현될 수 있음을 보인다.

Ⅱ. 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 z 변환이다. 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_2등과 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 알고리즘을 사용하면 곱셈연산의 양이 60개에서 40개로 20개가 감소됨을 알 수 있다. (5)의 식을 Toom-Cook 알고리즘을 사용하면 다음과 같이 나타낼 수 있다.

$$\begin{bmatrix} Y_0 \\ Y_1 \\ Y_2 \end{bmatrix} = \boldsymbol{C}_3 \left\{ \boldsymbol{A}_3 \begin{bmatrix} H_0 \\ H_1 \\ H_2 \end{bmatrix} * \boldsymbol{A}_3 \begin{bmatrix} X_0 \\ X_1 \\ X_2 \end{bmatrix} \right\} \tag{6}$$

이 식에서 *는 내적을 의미하며 $\boldsymbol{A}_3$와 $\boldsymbol{C}_3$는 각각 다음과 같다.

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

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

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.

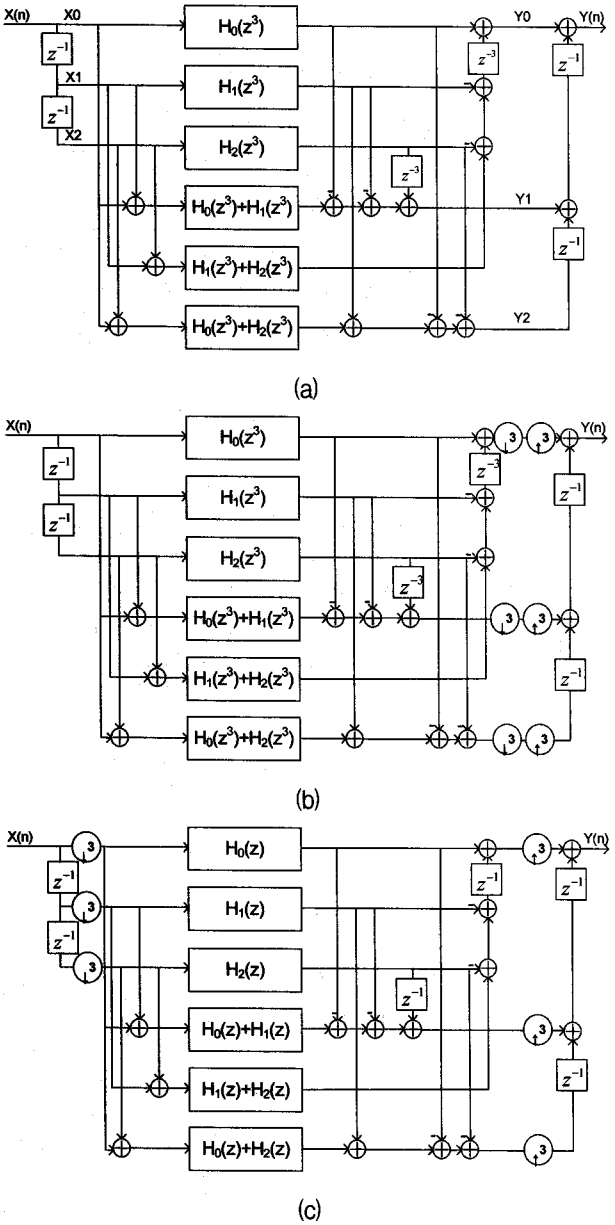


그림 1. 제안된 다운샘플링 3의 구조, (a)기본구조, (b)샘플러 삽입 구조, (c)최종구조

Fig. 1. Proposed down sampling 3 structure, (a)basic, (b)samplers inserted, (c)final.

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, 출력단에서는 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.

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

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.

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

위의 식에서 보듯이 Y_3를 만들기 위해 사용된 부필터의 수는 모두 9개이며 다음의 Y_2, Y_1, Y_0도 모두 Y_3에 사용된 부필터들의 조합으로 구성할 수 있다.

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

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

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

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.

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

이 식에서 $\boldsymbol{A_4}$와 $\boldsymbol{C_4}$는 각각 다음과 같다.

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

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

$\boldsymbol{A_4}$의 행의 수는 9이므로 9개의 부필터가 사용됨을 알 수 있으며, $\boldsymbol{C_4}$는 출력을 얻기 위한 사용된 부필터의 조합을 나타내고 있다. 식 (10)-(12)를 사용하여 필터 구조를 만들면 그림 2와 같다.

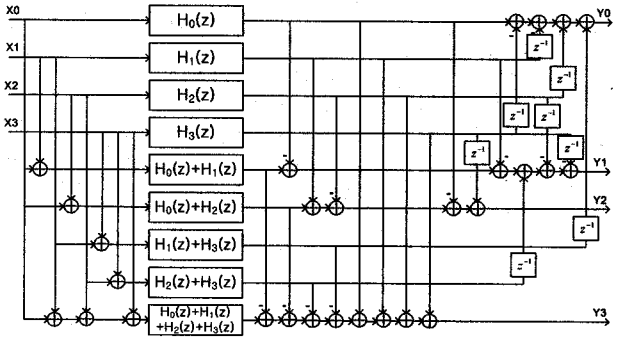


그림 2. 제안된 다운샘플링 4의 구조
Fig. 2. Proposed down sampling 4 structure.

$Y_0(z^4)$, $Y_1(z^4)$, $Y_2(z^4)$, $Y_3(z^4)$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 출력이 한 개씩 모아야 완전한 출력신호가 만들어진다.

2. 3 데시메이션 N의 알고리즘

이 절에서는 2. 1과 2. 2절에서 제안된 고속 알고리즘을 일반화하여 임의의 데시메이션 N을 사용하여 곱셈의 양을 감소시키는 알고리즘을 제안하려고 한다. 입출력과 임펄스응답을 각각 N개의 polyphase로 분해하여 Toom-Cook 알고리즘을 사용하면 다음과 같이 나타낼 수 있다.

$$\begin{bmatrix} Y_0 \\ Y_1 \\ \vdots \\ Y_{N-1} \end{bmatrix} = \boldsymbol{C_N}\left\{ \boldsymbol{A_N}\begin{bmatrix} H_0 \\ H_1 \\ \vdots \\ H_{N-1} \end{bmatrix} * \boldsymbol{A_N}\begin{bmatrix} X_0 \\ X_1 \\ \vdots \\ X_{N-1} \end{bmatrix} \right\} \tag{13}$$

이 식에서 $\boldsymbol{A_N}$의 행의 수는 사용되는 부필터의 수를 의미하므로 행의 수를 줄이는 것이 곱셈연산의 양을 줄이는 효율적인 방법이 된다. 최적의 $\boldsymbol{A_N}$을 유도하기 위하여 이미 유도된 $\boldsymbol{A_3}$를 다음과 같이 분해하여 나타내보자.

$$\boldsymbol{A_3} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{I_3} \\ \boldsymbol{P_3} \\ \boldsymbol{B_3} \\ \boldsymbol{Q_3} \end{bmatrix} \tag{14a}$$

$$\begin{aligned} &\boldsymbol{I_3} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \boldsymbol{B_3} = [\,1\ 0\ 1\,], \\ &\boldsymbol{P_3} = [\,1\ 1\ 0\,], \boldsymbol{Q_3} = [\,0\ 1\ 1\,] \end{aligned} \tag{14b}$$

위의 $\boldsymbol{A_3}$를 보면 Identity 행렬 $\boldsymbol{I_3}$가 사용되며, 좌우 대칭행인 $\boldsymbol{B_3}$가 반드시 한 개 존재함을 알 수 있다. 따라서 이를 제외한 부필터는 다음과 같다.

$$\tilde{A}_3 = \begin{bmatrix} P_3 \\ Q_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} \tag{15}$$

$\boldsymbol{A_3}$와 같은 방법으로 $\boldsymbol{A_4}$는 다음과 같이 분해할 수

있다.

$$A_4=\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\\1&1&0&0\\1&0&1&0\\1&1&1&1\\0&1&0&1\\0&0&1&1\end{bmatrix}=\begin{bmatrix}I_4\\P_4\\B_4\\Q_4\end{bmatrix} \tag{16a}$$

$$I_4=\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix},\ B_4=[\,1\ 1\ 1\ 1\,],$$
$$P_4=\begin{bmatrix}1&1&0&0\\1&0&1&0\end{bmatrix},\ Q_4=\begin{bmatrix}0&1&0&1\\0&0&1&1\end{bmatrix} \tag{16b}$$

위의 A_4에서 Identity 행렬 I_4와 좌우 대칭행인 B_4를 제외하면 $\widetilde{A_4}$는 다음과 같다.

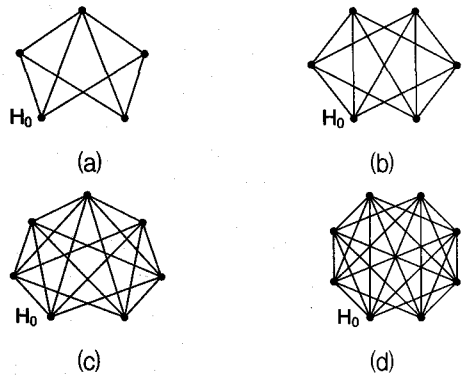
$$\widetilde{A}_4=\begin{bmatrix}P_4\\Q_4\end{bmatrix}=\begin{bmatrix}1&1&0&0\\1&0&1&0\\0&1&0&1\\0&0&1&0\end{bmatrix} \tag{17}$$

위의 $\widetilde{A_3}$과 $\widetilde{A_4}$을 관찰한 결과, 그림 3과 같은 삼각점 또는 사각점에서 대칭의 선을 구성하는 규칙을 발견할 수 있었다.

(a) (b)

그림 3. (a) 다운샘플링 3의 규칙,
(b) 다운샘플링 4의 규칙

Fig. 3. (a) down sampling 3's law,
(b) down sampling 4's law.



(a) (b)

(c) (d)

그림 4. (a) 다운샘플링 5의 규칙, (b) 다운샘플링 6의 규칙, (c) 다운샘플링 7의 규칙, (d) 다운샘플링 8의 규칙

Fig. 4. (a) down sampling 5's law, (b) down sampling 6's law, (c) down sampling 7's law, (d) down sampling 8's law.

그림 3(a)은 3개의 꼭지점이 존재하므로 3개의 선분을 만들 수 있다. 그림 3(a)에서 보듯이 H0-H1선분과 H1-H2선분은 좌우 대칭이다. 그러나 H0-H2선분은 좌우를 연결할 뿐, 좌우 대칭인 선분이 존재하지 않는다. 따라서 좌우 대칭이 아닌 H0와 H2는 연결할 수 없다. 그림 3(b)에서도 H0-H1선분과 H2-H3선분은 좌우 대칭이므로 연결하였고, H0-H2선분과 H1-H3선분도 좌우 대칭이므로 연결하였다. 그러나 좌우 대칭의 선분이 존재하지 않는 H1-H2, H0-H3는 연결될 수 없다. 이와 같은 좌우 대칭 선분의 존재에 대한 규칙성을 적용하면 어떤 다운샘플링 수에 대하여도 쉽게 선분을 그릴 수 있다. 즉, 임의의 수로 쉽게 병렬화가 가능하다. 따라서 데시메이션 5부터 8까지의 다각점에 대칭되는 선분을 그리면 그림 4와 같다.

위의 도형을 사용하여 $\widetilde{A_5}$를 유도하면 다음과 같다.

$$\widetilde{A}_5=\begin{bmatrix}P_5\\Q_5\end{bmatrix}=\begin{bmatrix}1&1&0&0&0\\1&0&1&0&0\\0&1&1&0&0\\1&0&0&1&0\\0&1&0&0&1\\0&0&1&1&0\\0&0&1&0&1\\0&0&0&1&1\end{bmatrix} \tag{18}$$

위의 $\widetilde{A_5}$는 8행이고 A_5는 총 14행이므로 14개의 부필터가 필요하다. 즉, 그림 4를 사용하면 쉽게 $\widetilde{A_N}$을 유도할 수 있다. 이와 같은 관찰결과를 사용하여 A_2부터 A_8까지 사용되는 부필터의 수를 구하면 표 1과 같다.

위의 표 1을 일반화하여 데시메이션 N인 경우의 부필터의 수를 구하는 식을 다음과 같이 만들 수 있다. 먼저 데시메이션 N인 경우에 위의 표에서 두 번째 열인 I_N의 행수는 N이 된다. 그리고 세 번째 열인 대칭행수는 1이다. 마지막으로 네 번째 열인 $\widetilde{A_N}$의 행수는

표 1. 제안된 방식의 부필터의 수

Table 1. Number of sub filter for proposed structure.

N	I_N의 행 수	대칭 행수	$\widetilde{A_N}$ 의 행수	A_N의행수 (부필터의수)
2	2	1	0	3
3	3	1	2	6
4	4	1	4	9
5	5	1	8	14
6	6	1	12	19
7	7	1	18	26
8	8	1	24	33

그림 4를 이용하여 ${}_NC_2 - \lfloor \frac{N}{2} \rfloor$으로 표시할 수 있다. 즉, ${}_NC_2$는 N각형에서 그릴 수 있는 모든 선분의 수를 표시하며, 좌우 대칭의 선분이 존재하지 않는 선분의 수는 $\lfloor \frac{N}{2} \rfloor$로 표시할 수 있다. 따라서 표 (1)의 네 번째 열인 $\widetilde{A_N}$의 행수는 ${}_NC_2 - \lfloor \frac{N}{2} \rfloor$로 표시할 수 있다. 이를 종합하면 부필터의 수, 즉 A_N의 행의 수는 다음과 같은 식으로 나타낼 수 있다.

$$\text{부필터의수} = N+1+{}_NC_2 - \lfloor \frac{N}{2} \rfloor \qquad (19)$$

위의 식에서 $\lfloor\ \rfloor$는 소수점 이하를 버린 정수를 의미한다. 그림 4의 규칙과 식(19)를 사용하면 일반 필터를 쉽게 병렬화할 수 있게 된다. 이 절에서 제안된 병렬화 방법은 블록필터를 사용하는 방법보다 연산량이 감소됨은 물론이다.

Ⅲ. 연산의 양 비교

3. 1 부필터의 수 비교

탭의 길이가 정해진 FIR 필터를 부필터를 사용하여 구현할 때에 부필터의 수가 적을수록 곱셈연산의 양이 감소하게 된다. 비교를 위하여 N=2부터 8까지의 병렬화에서 필요한 부필터의 수를 비교해보기로 한다.

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

표 2. 제안된 방식의 부필터의 수 비교 (N: 다운샘플링 수)

Table 2. Number of sub filter for proposed structure. (N: number of down sampling)

N	방식 1	방식 2	제안 방식	제안방식의 %
2	4	3	3	100
3	9	7	6	85.7
4	16	13	9	69.2
5	25	21	14	66.7
6	36	31	19	61.3
7	49	43	26	60.5
8	64	57	33	57.9

3. 2 샘플당 곱셈연산의 수 비교

고속 알고리즘의 성능을 평가하는 방법은 부필터의 수 이외에도 여러 가지가 가능하다. 이중에서 샘플당 곱셈연산의 수를 평가해 보기로 한다. 즉, 제안된 고속알고리즘에 대하여 2부터 8까지 다운샘플링 구조의 샘플당 곱셈연산을 비교하면 표 3과 같다. 표 3에서 곱셈연산의 양을 정량적으로 비교하기 위하여 840탭의 FIR 필터를 사용하여 비교하였다.

표 3에서 보듯이, 제안된 구조의 2부터 8까지의 다운샘플링을 비교한 결과 최대 샘플당 곱셈연산의 양을 49.93% 감소시킬 수 있었다. 다운샘플링의 수를 4까지 증가시킬 때에는 연산양이 매우 감소하지만 5 이상의 다운샘플링에서는 그다지 많이 감소하지 않음을 관찰할 수 있다. 이는 병렬화의 수를 5 이상으로 증가시키는 것은 비효율적임을 나타낸다. 따라서 2, 3, 4의 다운샘

표 3. 제안된 방식의 샘플당 곱셈연산의 수 (840탭의 FIR 필터)

Table 3. Number of multiplication/sample for the proposed structure. (840 tab FIR filter case)

다운샘플링	부필터 수	부필터의 탭수	총곱셈 연산	샘플당 곱셈연산	%
1	1	840	840	840	100
2	3	420	1260	630	75.0
3	6	280	1680	560	66.67
4	9	210	1890	472.5	56.25
5	14	168	2352	470.4	56.0
6	19	140	2660	443.3	52.78
7	26	120	3120	445.7	53.06
8	33	105	3365	420.6	50.07

표 4. 제안된 방식의 구현소자 수 비교

Table 4. Number of element for the proposed structure.

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

플링 방식을 반복하여 적용하는 것이 유리하다. 즉, 이론적으로는 8의 다운샘플링 구조 보다는 2의 다운샘플링 구조를 3번 반복하여 적용하는 것이 연산의 양을 더 감소시킬 수 있는 방법이다.

3. 3 하드웨어 구현소자의 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 구현면적을 비교해보기로 한다. 예제로서 3탭의 FIR 필터를 사용하였다. 3탭의 필터를 비교하면 필터의 수가 곱셈연산의 수가 되므로 하드웨어 구현면적을 쉽게 비교할 수 있다. 비교 대상은 pseudocirculant 행렬을 사용하는 3개의 고속알고리즘들을 비교한다. 즉, pseudocirculant 행렬을 변형 없이 그대로 사용하는 방식 1과 [3]에서 제안한 고속알고리즘을 사용하는 방식 3과 본 논문이 제안하는 방식의 구현소자 수를 비교하면 다음과 같다.

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

3. 4 MAC 프로세서를 사용한 구현 비교

고속 알고리즘들은 다운샘플링을 통한 병렬화로 샘플당 곱셈연산의 수는 감소시킬 수 있으나, 구현면적이 오히려 증가할 수도 있다. 그러나 어떤 고속 알고리즘이 MAC 엔진을 탑재한 프로세서로 효과적으로 구현될 수 있다면 고속 알고리즘으로서의 효용성이 증대될 것이다. 따라서 이 절에서는 36탭의 필터에 대하여 다양한 방식의 고속알고리즘을 MAC 프로세서를 사용하여 구현하였으며 그 비교 결과는 표 5와 같다. 사용한 DSP는 TMS320C6711 프로세서를 사용하였으며 code composer를 사용하여 명령어 수를 카운트하였다. 사용된 입력샘플의 수는 108개이며, 36개의 필터계수와 필터링된 후 143개의 출력샘플을 얻을 때까지의 명령어수를 비교하였다. 표 5에서 보듯이 다운샘플링 구조를 사용하지 않은 일반 scalar 구조는 408,144개의 명령어를 사용하였다. 방식 4는 [2]에서 제안된 다운샘플링 2를 사용하는 방식이며, 이를 한번 적용한 구조와 2번 반복한 구조를 코딩하였다. 그 결과 각각 335,326과 271,998 개의 명령어가 사용되었다. 방식 3은 [3]에서 제안된 다운샘플링 3을 사용하는 방식이며, 이를 한번 적용한 구조와 2번 반복한 구조를 코딩하였다. 그 결과 각각 312,021과 278,010 개의 명령어가 사용되었다. 이론적인 샘플당 곱셈연산은 방식 3의 3x3 구조가 방식 4의 2x2 구조보다 우수하나 구현 명령어수는 오히려 방식 4의 2x2 구조가 더 효과적임을 알 수 있다. 이 논문이 제안한 방식 중에서 다운샘플링 3을 사용하는 방식을 사용하였으며, 이를 한번 적용한 구조와 2번 반복한 구조를 코딩하였다. 그 결과 각각 301,288과 257,864 개의 명령어가 사용되었다. 이 결과는 방식 4나 방식 3의 구조보다 명령어 수가 감소되었음을 알 수 있다.

표 5. 제안된 고속알고리즘의 MAC프로세서 구현 명령어 수(36탭의 FIR 필터)

Table 5. Number of MAC processor instructions for the proposed algorithm.

다운샘플링	부필터수	부필터의 탭수	총곱셈연산	샘플당곱셈연산 (%)	명령어수 (%)
scalar필터	1	36	36	36 (100)	408,144 (100)
2 (방식 4)	3	18	54	27 (75)	335,326 (82.1)
2×2 (방식 4)	9	9	81	20.25 (56.25)	271,998 (66.6)
3 (방식 3)	6	12	72	24 (66.7)	312,021 (76.4)
3×3 (방식 3)	36	4	144	16 (44.4)	278,010 (68.1)
3 (제안방식)	6	12	72	24 (66.7)	301,288 (73.8)
3×3 (제안방식)	36	4	144	16 (44.4)	257,864 (63.1)
4 (제안방식)	9	9	81	20.25 (56.25)	266,886 (65.4)

Ⅳ. 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 기본

적인 N=1의 필터와 비교하여 49.93%의 샘플당 곱셈연산 감소효과를 보였다. 세 번째의 하드웨어 구현소자의 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 MAC 프로세서를 사용한 구현비교 실험을 통하여 프로세서를 사용하는 경우의 효용성을 실험하였다. N=3의 병렬화 경우에 대하여 기존의 구조와 비교하였으며, 기존구조와 비교하여 명령어의 수를 3.4% 감소시킬 수 있었다. 이와 같은 네 가지 실험결과를 통하여 제안된 구조가 일반적인 필터 구현에 효과적으로 널리 사용될 수 있음을 입증하였다.

참 고 문 헌

[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.

저 자 소 개

장 영 범(정회원)
1981년 연세대학교 전기공학과 졸업(공학사)
1990년 Polytechnic University 대학원 졸업(공학석사)
1994년 Polytechnic University 대학원 졸업(공학박사)
1981년~1999년 삼성전자 System LSI 사업부 수석연구원
2000년~2002년 이화여자대학교 정보통신학과 연구교수
2002년~현재 상명대학교 정보통신공학과 교수
<주관심분야 : 통신신호처리, 오디오 신호처리, 통신신호처리용 SoC 설계>

이 원 상(학생회원)
2004년 상명대학교 컴퓨터 시스템 공학과 졸업(공학사)
2004년~현재 상명대학교 대학원 컴퓨터정보통신공학과 석사과정
<주관심분야 : 통신신호처리, 통신신호처리용 SoC 설계>

오 세 만(학생회원)
2005년 상명대학교 정보통신 공학과 졸업(공학사)
2005년~현재 상명대학교 대학원 컴퓨터정보통신공학과 석사과정
<주관심분야 : 통신신호처리, 통신신호처리용 SoC 설계>