논문 2004-41SD-3-7

고속 멀티미디어 통신시스템을 위한 효율적인 FFT 알고리즘 및 하드웨어 구현

(Efficient FFT Algorithm and Hardware Implementation for High Speed Multimedia Communication Systems)

정 윤 호*, 김 재 석*

(Yunho Jung and Jaeseok Kim)

요 약

본 논문에서는 고속 멀티미디어 통신 시스템을 위한 효율적인 FFT 알고리즘과 이의 하드웨어 구현 결과를 제시한다. 제안된 알고리즘은 radix-4 버터플라이 연산자를 기반으로 구현되어 기존의 radix-2 버터플라이 연산자 기반의 알고리즘에 비해 2배의 처리율(processing rate)을 갖으며, 또한 radix-2^3 알고리즘의 비단순 승산기의 수를 줄이는 특성을 그대로 이용하므로, 같은 처리율을 갖는 radix-4 알고리즘에 비해 저면적 구현이 가능한 장점을 갖는다. 제안된 알고리즘의 하드웨어 구현 및 검증을 위해 VHDL 언어를 이용하여 MDC 파이프라인 구조를 갖는 64-point FFT 프로세서를 설계하였다. 0.6㎛ 공정을 이용하여 논리 합성한 결과, 제안된 알고리즘을 이용하여 구현한 경우, 기존의 radix-4 알고리즘을 이용하여 구현하는 경우보다 약 30% 정도 면적 면에서 이득을 얻을 수 있음을 확인하였다. 고속 동작이 가능하며 동시에 면적 효율적인 특성으로 인해, 제안된 알고리즘은 무선 LAN 시스템, DAB 및 DVB 시스템, ADSL/VDSL 시스템 등 고속 멀티미디어 통신 시스템에 적합한 알고리즘이라 할 수 있다.

Abstract

In this paper, we propose an efficient FFT algorithm for high speed multimedia communication systems, and present its pipeline implementation results. Since the proposed algorithm is based on the radix-4 butterfly unit, the processing rate can be twice as fast as that based on the radix-2^3 algorithm. Also, its implementation is more area-efficient than the implementation from conventional radix-4 algorithm due to reduced number of nontrivial multipliers like using the radix-23 algorithm. In order to compare the proposed algorithm with the conventional radix-4 algorithm, the 64-point MDC pipelined FFT processor based on the proposed algorithm was implemented. After the logic synthesis using 0.6㎛ technology, the logic gate count for the processor with the proposed algorithm is only about 70% of that for the processor with the conventional radix-4 algorithm. Since the proposed algorithm can be achieve higher processing rate and better efficiency than the conventional algorithm, it is very suitable for the high speed multimedia communication systems such as WLAN, DAB, DVB, and ADSL/VDSL systems.

Keywords : FFT algorithm, Pipeline implementation, WLAN, DAB/DVB, ADSL/VDSL

I. 서 론

90년대 인터넷의 광범위한 보급에 따라 고속 멀티미디어 통신 시스템에 대한 관심은 급속도로 증가하였다[1]. 다양한 서비스와 전송속도로 다양한 품질의 데이터를 제공하기 위해, 고속 멀티미디어 통신 시스템에 대한 여러 연구가 진행되고 있는데, 이 중 대표적인 것으로 유럽 RACE 프로젝트에서 연구된 MBS, ACTS의 MEDIAN, ETSI의 HiPERLAN, IEEE 802.11a의 WLAN등을 들 수 있다. 이와 같은 연구들은 고속 멀티미디어 통신 시스템의 요구 조건을 만족하기 위하여 단

* 정회원, 연세대학교 전기전자공학과
(Department of Electrical and Electronic Engineering, Yonsei University)
※ 본 연구는 대학 IT연구센터 육성 지원 사업의 연구결과로 수행되었음.
접수일자 : 2002년11월1일, 수정완료일 : 2004년2월26일

일 채널을 통한 단일 반송파 통신 방법보다 다중 반송파를 이용한 통신 방법을 고려하고 있으며, 이 중 OFDM 방식은 높은 대역 효율성, 다중 경로 페이딩에 대한 저항성과 같은 이점으로 인해 고속 멀티미디어 통신 시스템의 변조 기법으로 각광받고 있다[2].

OFDM 전송 방식의 기본 개념은 직렬로 입력되는 데이터 열을 N개의 병렬 데이터 열로 변환한 후, 이를 각각 분리된 부반송파(subcarrier)에 실어 전송함으로써 데이터 전송률을 높이는 것이다. 이때 부반송파는 직교성을 유지할 수 있도록 적절히 선택되어야 한다. 이러한 부반송파는 송·수신단에서 IFFT와 FFT 프로세서를 이용하여 생성 가능하며, 따라서 고속의 FFT 프로세서를 구현하는 것이 다중반송파 방식의 고속 멀티미디어 통신 시스템을 구현하기 위한 핵심사항이라 할 수 있다[3].

60년대 Cooley와 Tukey의 FFT 알고리즘이 개발된 이래로 고속 FFT를 구현하기 위한 노력은 계속되어 왔다. 알고리즘 측면에서 Goertzel 알고리즘, split-radix 알고리즘, Winograd 알고리즘, QFT(Quick Fourier Transform) 알고리즘, Radar-Brenner 알고리즘, radix-2^2, radix-2^3 알고리즘[4] 등이 개발되었고, 구조적 측면에서 단일 버터플라이 연산자 구조[5]를 이용하는 방법, 파이프라인 구조[6]를 이용하는 방법, 완전 셔플(perfect shuffle), 시스토릭 어레이(systolic array)와 같은 병렬 구조[7]를 이용하는 방법 등이 제안되었다. 이 중 구조적으로는 면적과 처리율의 교환관계(trade-off)를 적절히 만족시키는 파이프라인 구조가 이용되고 있으며, 알고리즘은 고속 구현이 가능하면서 동시에 비단순(nontrivial) 승산기의 수를 줄일 수 있는 radix-2^3 알고리즘이 고속 통신 시스템을 위해 사용되고 있다[8].

Radix-2^3 알고리즘은 FFT 알고리즘의 일반적 유도 방법인 트위들 팩터 분해법(twiddle factor decomposition technique)을 통해 비단순 승산기의 수를 줄임으로써 면적 측면에서 이득을 얻고자 한 알고리즘이다. 그러나 비단순 승산기의 수를 기존의 알고리즘에 비해 줄일 수 있다 하더라도, radix-2 버터플라이 연산자에 기반하여 구현되기 때문에, radix-4 버터플라이 연산자에 기반한 알고리즘에 비해 낮은 처리율을 갖는 단점이 있다. 이러한 단점은 앞서 언급된 고속 멀티미디어 통신을 위한 OFDM 시스템 구현에 치명적인 약점이 될 수 있다. 따라서 본 논문에서는 radix-2^3 알고리즘의 비단순 승산기의 수를 줄이는 특성을 그대로 이용하면서도, radix-4 버터플라이 연산자에 기반한 효율적인 알고리즘을 제안한다. 제안된 알고리즘은 radix-4 버터플라이 연산자에 기반하기 때문에 radix-2^3 알고리즘에 비해 2배의 처리율을 갖으며, 또한 비단순 승산기의 수를 줄일 수 있으므로 기존의 radix-4 알고리즘에 비해 저면적의 구현이 가능한 특징을 갖는다.

본 서론에 이어 II장에서는 FFT 프로세서 구현을 위한 파이프라인 구조에 대해 언급하고, III장에서는 제안된 효율적인 FFT 알고리즘에 대해 설명한다. 이어 IV장에서는 제안된 알고리즘을 파이프라인 구조를 이용하여 설계한 결과를 제시하고, V장에서는 FPGA 기반 검증 결과 및 논리합성 결과를 통해 기존 알고리즘과의 성능 분석을 수행한다. 마지막으로 VI장에서 본 논문의 결론을 맺는다.

II. FFT 프로세서의 파이프라인 구조

고속 시스템을 위한 FFT 알고리즘의 구현 방법으로 파이프라인 방식이 일반적이다. 이는 파이프라인 방식이 면적 대 처리율의 교환 관계를 적절히 만족시켜 주기 때문이다. 그림 1은 파이프라인 방식 FFT 프로세서의 기본 구조를 보여준다.

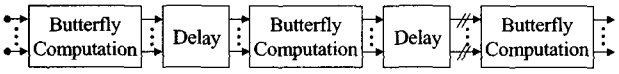


그림 1. 파이프라인 방식 FFT 프로세서의 구조
Fig. 1. Pipelined FFT processor architecture.

그림 1에서 볼 수 있는 것처럼, 파이프라인 방식의 FFT 프로세서는 지연 교환기(delay commutator)에 의해 분리된 여러 개의 버터플라이 연산자들로 구성되어진다. 지연 교환기는 FFT 각 stage의 데이터 정렬을 위해 필요하며, 메모리 소자를 이용하여 구현된다.

파이프라인 구조로 FFT 프로세서를 구현하는 경우, 무엇보다 알고리즘의 적절한 선택이 중요하게 된다. 이는 알고리즘의 적절한 선택에 따라 프로세서의 전체 면적 및 처리율이 결정되기 때문이다. 우선, 처리율의 측면에서 데이터의 입력 및 출력 속도를 R이라 가정하면, radix-2 버터플라이 연산자는 2개의 데이터를 한번에 처리하므로 프로세서의 처리율은 $2R$이 되고, 반면 radix-4 버터플라이 연산자는 4개의 데이터를 한번에 처리하므로 처리율은 $4R$이 된다. 따라서 radix-4 알고리즘을 이용하는 경우, radix-2 알고리즘을 이용하는 경우에 비해 2배의 처리율을 얻을 수 있음을 알 수 있다[9].

면적의 측면에서 두 알고리즘을 비교해 보면, radix-2 알고리즘의 경우, radix-2 버터플라이 연산자는 2개의 복소수 가산기와 1개의 복소수 승산기로 구현되고, N-point FFT 연산을 위해 요구되는 버터플라이 연산자의 수는 $\log_2 N$개이므로 필요한 가산기의 수는 $2\log_2 N$개이며, 승산기의 수는 $\log_2 N$개이다. 승산기 중 마지막 3 stage는 단순(trivial) 승산기로 구현 가능하므로 요구되는 비단순 승산기의 수는 $(\log_2 N-3)$개 이다. Radix-4 알고리즘을 이용하는 경우, radix-4 버터플라이 연산자는 8개의 복소수 가산기와 3개의 복소수 승산기로 구현되고, N-point FFT 연산을 위해 요구되는 버터플라이 연산자의 수는 $\log_4 N$개이므로, 구현을 위해 필요한 가산기의 총수는 $8\log_4 N$개이고, 승산기의 수는 $3\log_4 N$개이다. Radix-2 알고리즘과 마찬가지 방법으로, 승산기 중 마지막 stage는 단순 승산기로 구현 가능하므로, 요구되는 비단순 승산기의 총수는 $3(\log_4 N-1)$개이다. 표 1은 radix-2 알고리즘과 radix-4 알고리즘을 이용하여 파이프라인 방식의 FFT 프로세서를 구현하는 경우 요구되는 면적 및 처리율을 정리하여 보여준다.

표 1. 알고리즘에 따른 면적 및 처리율 비교 결과
Table 1. Area and processing rate comparison results

알고리즘	면 적		처리율
	복소 가산기	비단순 승산기	
radix-2	$4\log_4 N$	$2\log_4 N-3$	$2R$
radix-4	$8\log_4 N$	$3(\log_4 N-1)$	$4R$

표 1을 통해 파이프라인 방식으로 FFT 프로세서를 구현하는 경우, radix-2 알고리즘을 이용하면 면적 면에서 이득을 얻을 수 있고, radix-4 알고리즘을 이용하면 처리율면에서 이득을 얻을 수 있음을 알 수 있다. 서론에서 언급된 것처럼 고속 멀티미디어 통신 시스템을 위한 FFT 프로세서를 구현하기 위해서는 처리율면에서 이득을 갖는 radix-4 알고리즘을 이용하는 것이 유리함을 알 수 있다.

III. 제안된 FFT 알고리즘

N-point DFT(Discrete Fourier Transform)은 식(1)과 같이 정의된다.

$$X(k)=\sum_{n=0}^{N-1} x(n)\ W_N^{nk},\quad n,k=0,1,\cdots,N-1 \qquad (1)$$

여기서 W_N은 $e^{-j\frac{2\pi}{N}}$이다. 제안된 알고리즘을 유도하기 위해, 3차원 인덱스 맵(3-dimension index map)을 적용하면, 두 인덱스 n, k는 식(2)와 같이 표현된다.

$$\begin{aligned} n&= \frac{N}{4}n_1+\frac{N}{8}n_2+n_3,\\ &\qquad 0\le n_1<3,\ 0\le n_2<1,\ 0\le n_3<\frac{N}{8}-1\\ k&= k_1+4k_2+8k_3,\\ &\qquad 0\le k_1<3,\ 0\le k_2<1,\ 0\le k_3<\frac{N}{8}-1. \end{aligned} \qquad (2)$$

분해된 인덱스를 DFT의 정의식 (1)에 대입하면,

$$\begin{aligned} X(k)&=X(k_1+4k_2+8k_3)\\ &=\sum_{n_3=0}^{\frac{N}{8}-1}\sum_{n_2=0}^{1}\sum_{n_1=0}^{3}\left(\begin{array}{c} x(\frac{N}{4}n_1+\frac{N}{8}n_2+n_3)\cdot\\ W_N^{(\frac{N}{4}n_1+\frac{N}{8}n_2+n_3)(k_1+4k_2+8k_3)}\end{array}\right)\\ &=\sum_{n_3=0}^{\frac{N}{8}-1}\sum_{n_2=0}^{1}\left(\begin{array}{c} BF_4(\frac{N}{8}n_2+n_3,k_1)\cdot\\ W_N^{(\frac{N}{8}n_2+n_3)k_1}\ W_N^{(\frac{N}{8}n_2+n_3)(4k_2+8k_3)}\end{array}\right) \end{aligned} \qquad (3)$$

이다. 여기서 $BF_4(\frac{N}{8}n_2+n_3,k_1)$는

$$\begin{aligned} &BF_4(\frac{N}{8}n_2+n_3,k_1)\\ &=\sum_{n_1=0}^{3}x(\frac{N}{4}n_1+\frac{N}{8}n_2+n_3)\ W_N^{\frac{N}{4}n_1k_1}\\ &=\sum_{n_1=0}^{3}x(\frac{N}{4}n_1+\frac{N}{8}n_2+n_3)\ W_4^{n_1k_1}\\ &=\left(\begin{array}{c} x(\frac{N}{8}n_2+n_3)+x(\frac{N}{4}+\frac{N}{8}n_2+n_3)\ W_4^{k_1}\\ +x(\frac{N}{2}+\frac{N}{8}n_2+n_3)\ W_4^{2k_1}\\ +x(\frac{3N}{4}+\frac{N}{8}n_2+n_3)\ W_4^{3k_1}\end{array}\right) \end{aligned} \qquad (4)$$

이고, 이는 radix-4 버터플라이 연산자에 해당하며, $W_N^{(\frac{N}{8}n_2+n_3)k_1}$는 첫 stage의 트위들 팩터에 해당한다. 식(3)에서 이 트위들 팩터를 승산한 후 n_2에 대한 연산을 수행하고, 이어서 분해 과정을 진행한다면, 이는 일반적인 radix-4+2 mixed radix 알고리즘이 된다. 제안된 알고리즘의 핵심 아이디어는 식(3)에서의 트위들 팩터를 바로 연산하지 않고, 이 연산을 다음 분해과정-즉, n_2의 연산-에서 수행함으로써 첫 stage의 트위들 팩터를 비단순 승산에서 단순 승산으로 변경하는 것이다. 이를 수식을 통해 유도해보면 우선 식(3)의 트위들 팩터는

$$W_N^{(\frac{N}{8}n_2+n_3)(k_1+4k_2+8k_3)} = W_N^{Nn_2k_3}\, W_N^{\frac{N}{8}n_2(k_1+4k_2)}\, W_N^{n_3(k_1+4k_2)}\, W_N^{8n_3k_3} = W_8^{n_2(k_1+4k_2)}\, W_N^{n_3(k_1+4k_2)}\, W_{\frac{N}{8}}^{n_3k_3} \quad (5)$$

이고, 이를 식(3)에 다시 대입하여 정리하면,

$$\begin{aligned} X(k) &= X(k_1+4k_2+8k_3) \\ &= \sum_{n_3=0}^{\frac{N}{8}-1}\sum_{n_2=0}^{1}\left(BF_4(\frac{N}{8}n_2+N_3,k_1)\cdot W_8^{n_2(k_1+4k_2)} \cdot W_N^{n_3(k_1+4k_2)}\cdot W_{\frac{N}{8}}^{n_3k_3}\right) \quad (6) \\ &= \sum_{n_3=0}^{\frac{N}{8}-1}\left(\left[\sum_{n_2=0}^{1}BF_4(\frac{N}{8}n_2+N_3,k_1)\, W_8^{n_2(k_1+4k_2)}\right] \cdot W_N^{n_3(k_1+4k_2)}\cdot W_{\frac{N}{8}}^{n_3k_3}\right) \\ &= \sum_{n_3=0}^{\frac{N}{8}-1}\left[H(k_1,k_2,n_3)\, W_N^{n_3(k_1+4k_2)}\right] W_{\frac{N}{8}}^{n_3k_3} \end{aligned}$$

이다. 여기서 $H(k_1, k_2, n_3)$는

$$\begin{aligned} H(k_1,k_2,n_3) &= \sum_{n_2=0}^{1}BF_4(\frac{N}{8}n_2+n_3,k_1)\, W_8^{n_2(k_1+4k_2)} \\ &= BF_4(n_3,k_1)+BF_4(n_3+\frac{N}{8},k_1)\, W_8^{(k_1+4k_2)} \end{aligned} \quad (7)$$

으로, FFT SFG(Signal Flow Graph)상의 처음 두 stage 버터플라이 연산을 나타낸다. 또한 식(7)에서 트위들 팩터 $W_8^{(k_1+4k_2)}$는 다음 식(8)과 같다.

$$\begin{aligned} W_8^{(k_1+4k_2)} &= \left(\frac{1}{\sqrt{2}}(1-j)\right)^{(k_1+4k_2)} \\ &= \left(\frac{1}{\sqrt{2}}(1-j)\right)^{k_1}\cdot(-1)^{k_2} \end{aligned} \quad (8)$$

식(8)에서 $1/\sqrt{2}$는 간단한 상수 스케일러(scaler)이므로 식(8)에 대한 승산은 단순 승산으로 정의 내릴 수 있다. 따라서 식(7)의 연산은 단순 승산기로 구현 가능하며, 식(7)의 연산 후-즉, FFT SFG상의 처음 두 stage 후-트위들 팩터 $W_N^{n_3(k_1+4k_2)}$를 승산하기 위해 비단순 승산기가 요구된다. 지금까지의 분해 과정을 남은 $N/8$-point에 계속하여 적용하면 제안된 알고리즘을 얻을 수 있다.

그림 2는 N=64인 경우, 제안된 알고리즘의 FFT SFG를 보여준다. 그림에서 마름모는 단순 승산을 의미하며, 삼각형은 비단순 승산을 나타낸다. 그림 2에서 볼 수 있듯이, 제안된 알고리즘은 기존의 알고리즘과 달리, 단지 2번째 stage 버터플라이 연산후에 비단순 승산을 가짐을 알 수 있다.

제안된 알고리즘과 기존의 radix-4 알고리즘을 파이프라인 기법 중 하나인 MDC(Multi-path Delay Feedback) 기법에 적용한 구조가 그림 3, 4에 보여진다. 그림을 통해 N=64인 경우, 제안된 알고리즘이 단지 4개의 비단순 승산기를 요구하는 반면, 기존의 radix-4 알고리즘은 6개의 비단순 승산기를 요구함을 알 수 있다.

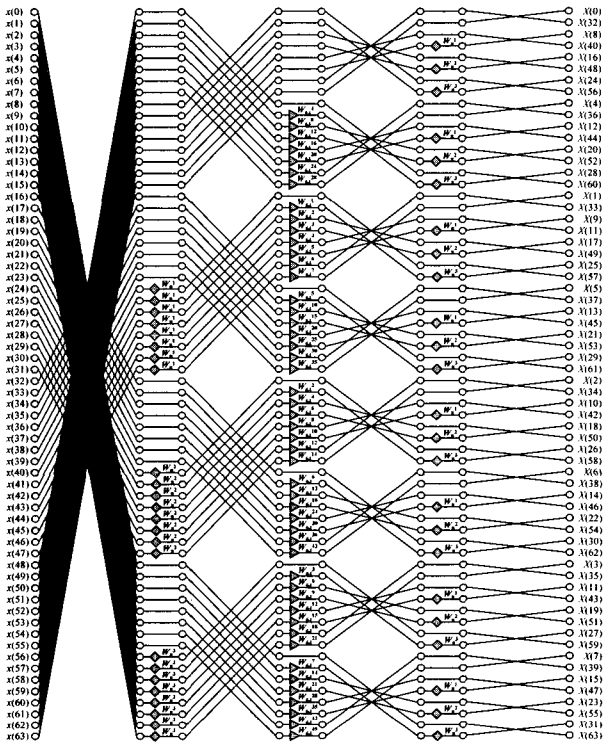


그림 2. N=64인 경우, 제안된 알고리즘의 FFT SFG
Fig. 2. FFT SFG of proposed FFT algorithm (N=64).

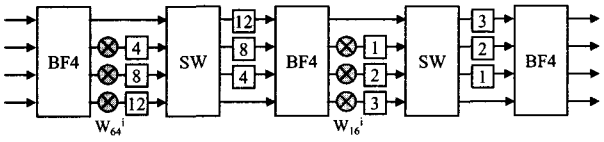


그림 3. Radix-4 알고리즘을 적용한 경우, MDC 파이프라인 방식 FFT 프로세서의 하드웨어 구조
Fig. 3. MDC pipelined FFT processor hardware architecture applying the radix-4 FFT algorithm.

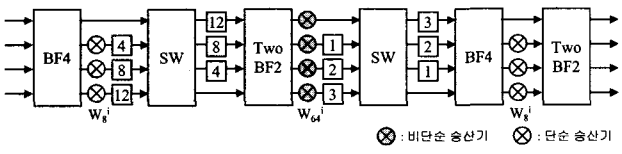


그림 4. 제안된 알고리즘을 적용한 경우, MDC 파이프라인 방식 FFT 프로세서의 하드웨어 구조
Fig. 4. MDC pipelined FFT processor hardware architecture applying the proposed FFT algorithm.

Ⅳ. 제안된 알고리즘을 이용한 FFT 프로세서의 설계

1. 하드웨어 구조의 결정

FFT 프로세서의 구현을 위한 파이프라인 방식은 MDC, SDF(Single-path Delay Feedback), SDC (Single-path Delay Commutator) 등으로 나눌 수 있다. MDC 방식은 다중경로를 통해 데이터를 전달하며, 지연교환기(delay commutator)로 SFG에 적합한 형태의 데이터 정렬을 수행한다. 이러한 특징으로 인해 세 방식 중 가장 큰 처리율을 갖으며, 콘트롤이 쉬운 장점이 있지만, 경로당 하나의 복소 승산기를 요구하므로 복잡도가 증가하는 단점이 있다. SDF 방식은 단일 경로를 통해 데이터를 전달하며, 변경된 버터플라이 연산자를 통해 데이터의 정렬을 수행한다. 단일 경로에 하나의 복소 승산기를 요구하므로, 복잡도는 작아지나, 단일 경로라는 특징으로 인해 처리율이 낮고, 버터플라이 연산자의 구조가 복잡하므로 콘트롤이 어려운 단점을 갖는다. SDC 방식은 두 방식의 특징을 결합한 형태인데 다중 입력을 갖으며 단일 데이터를 출력하는 변경된 지연교환기를 통해 SDF방식과 같은 수의 복소 승산기를 갖는다. 그러나 여전히 단일 경로로 인해 처리율이 낮은 단점이 있다.

제안된 알고리즘은 기존의 알고리즘에 비해 비단순 복소 승산기의 수를 크게 줄일 수 있는 장점을 갖기 때문에, 본 논문에서는 처리율이 크고, 콘트롤이 쉬운 MDC 방식의 FFT 프로세서를 구현하였다.

2. FFT 프로세서의 설계 및 검증

제안된 알고리즘을 이용한 MDC 파이프라인 방식의 64-point FFT 프로세서는 크게 radix-4 버터플라이 연산자, radix-2 버터플라이 연산자, 지연교환기, 트위들 팩터 승산기(twiddle factor multiplier) 등으로 구성된다. 그림 5는 설계된 FFT 프로세서의 전체 블록도를 보여준다.

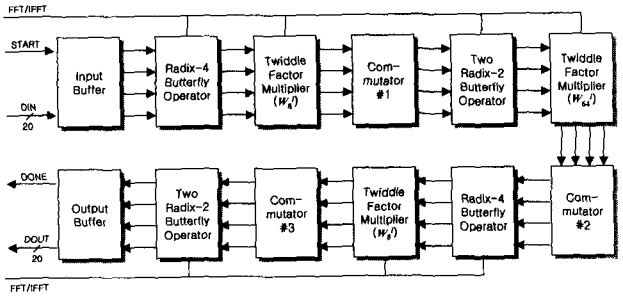
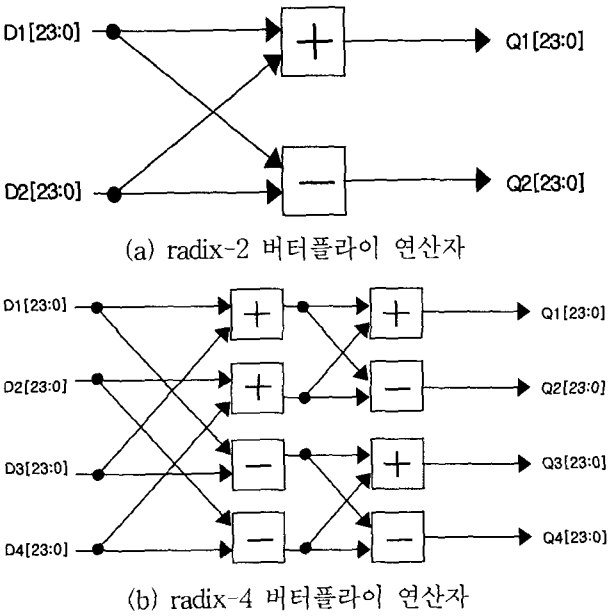


그림 5. 제안된 FFT 프로세서의 전체 블록도
Fig. 5. Block diagram of the proposed FFT processor.

가. 버터플라이 연산자

트위들팩터를 승산하는 부분을 별도의 모듈로 설계하였기 때문에, 버터플라이 연산자 모듈은 단지 입력데이터의 가산 및 감산 기능을 수행한다. 그림 6은 radix-2 및 radix-4 버터플라이 연산자의 블록도를 보여준다.



(a) radix-2 버터플라이 연산자

(b) radix-4 버터플라이 연산자

그림 6. 버터플라이 연산자의 블록도
Fig. 6. Block diagram of the butterfly operator.

나. 트위들팩터 승산기

(1) $W_8^{\pm i}$ 승산부

그림 2의 SFG를 보면, 트위들팩터 $W_8^{\pm i}$를 승산하는 경우 i는 ±1, ±2, ±3의 값을 가짐을 알 수 있다. 또한 이 값은 MDC 파이프라인 방식의 각 경로에서 고정적이다. 따라서 i의 값 ±1, ±2, ±3, 각각에 대한 승산부를 설계하여 이를 경로마다 배치함으로써 트위들 팩터 $W_8^{\pm i}$의 승산부를 설계하였다.

복소수 데이터 $(X_{re}+jX_{im})$에 $W_8^{\pm 1}$를 승산하는 과정은 식 (9), (10)과 같이 나타내어 질 수 있다. 식 (9)는 W_8^{+1}의 승산에 해당하며, 식 (10)은 W_8^{-1}의 승산에 해당한다.

$$(X_{re}+jX_{im})\cdot(\cos\frac{\pi}{4}-j\ \sin\frac{\pi}{4}) = \frac{1}{\sqrt{2}}((X_{re}+X_{im})+j(X_{im}-X_{re})) \quad (9)$$

$$(X_{re}+jX_{im})\cdot(\cos\frac{\pi}{4}+j\ \sin\frac{\pi}{4}) = \frac{1}{\sqrt{2}}((X_{re}-X_{im})+j(X_{im}+X_{re})) \quad (10)$$

식 (9)와 (10)을 보면 $W_8^{\pm1}$의 승산부는 입력데이터의 실수항, 허수항의 합과 차를 구하는 부분과 $1/\sqrt{2}$로 스케일링하는 부분으로 구성됨을 알 수 있다. 그림 7은 $W_8^{\pm1}$승산부의 블록도를 보여준다.

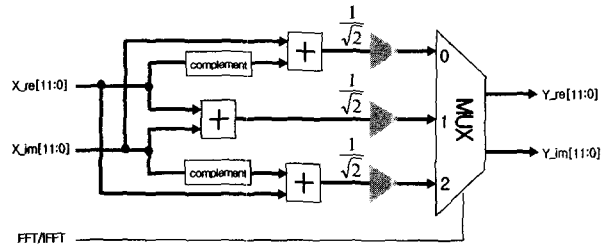


그림 7. $W_8^{\pm1}$ 승산부의 블록도
Fig. 7. Block diagram of the $W_8^{\pm1}$ multiplier.

복소수 데이터 $(X_{re}+jX_{im})$에 $W_8^{\pm2}$를 승산하는 과정은 식 (11), (12)과 같이 나타내어 질 수 있다.

$$(X_{re}+jX_{im})\cdot(\cos\frac{\pi}{2}-j\ \sin\frac{\pi}{2}) = X_{im}+j(-X_{re}) \quad (11)$$

$$(X_{re}+jX_{im})\cdot(\cos\frac{\pi}{2}+j\ \sin\frac{\pi}{2}) = (-X_{im})+jX_{re} \quad (12)$$

트위들팩터 $W_8^{\pm2}$을 승산하는 과정은 식 (11), (12)와 같이 단순히 입력데이터의 부호를 바꾸어 주는 것으로 구현 가능하며, 이의 블록도는 그림 8과 같다.

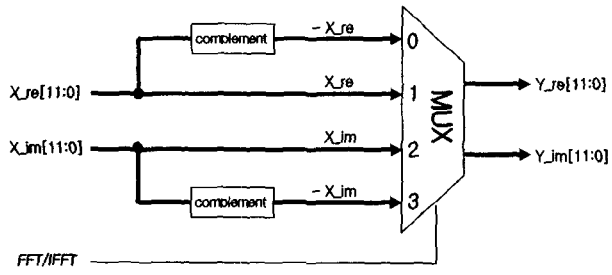


그림 8. $W_8^{\pm2}$ 승산부의 블록도
Fig. 8. Block diagram of the $W_8^{\pm2}$ multiplier.

식 (13), (14)는 복소수 데이터 $(X_{re}+jX_{im})$에 $W_8^{\pm3}$를 승산하는 과정을 각각 제시한다.

$$(X_{re}+jX_{im})\cdot(\cos\frac{3\pi}{4}-j\ \sin\frac{3\pi}{4}) = \frac{1}{\sqrt{2}}((X_{im}-X_{re})+j(-X_{re}-X_{im})) \quad (13)$$

$$(X_{re}+jX_{im})\cdot(\cos\frac{3\pi}{4}+j\ \sin\frac{3\pi}{4}) = \frac{1}{\sqrt{2}}((-X_{re}-X_{im})+j(X_{re}-X_{im})) \quad (14)$$

트위들팩터 $W_8^{\pm1}$을 승산하는 부분과 비슷하게, 입력데이터의 실수항과 허수항의 합과 차를 구하고 이의 보수를 취하는 부분과 스케일링 연산을 수행하는 부분으로 구성된다. 그림 9는 트위들팩터 $W_8^{\pm3}$ 승산부의 블록도를 보여준다.

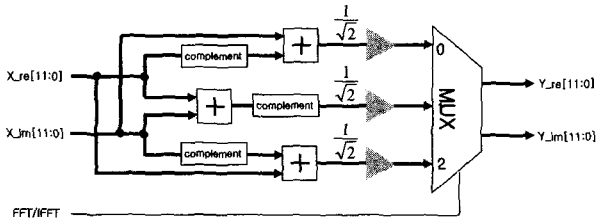


그림 9. $W_8^{\pm3}$ 승산부의 블록도
Fig. 9. Block diagram of the $W_8^{\pm3}$ multiplier.

(2) $W_{64}^{\pm i}$ 승산부

식 (15), (16)은 트위들팩터 $W_{64}^{\pm i}$를 승산하는 과정을 각각 보여준다.

$$(X_{re}+jX_{im})\cdot(\cos\frac{\pi}{32}i-j\ \sin\frac{\pi}{32}i) = (X_{re}\cos\frac{\pi}{32}i+X_{im}\sin\frac{\pi}{32}i) + j(X_{im}\cos\frac{\pi}{32}-X_{re}\sin\frac{\pi}{32}i) \quad (15)$$

$$(X_{re}+jX_{im})\cdot(\cos\frac{\pi}{32}i+j\ \sin\frac{\pi}{32}i) = (X_{re}\cos\frac{\pi}{32}i-X_{im}\sin\frac{\pi}{32}i) + j(X_{im}\cos\frac{\pi}{32}+X_{re}\sin\frac{\pi}{32}i) \quad (16)$$

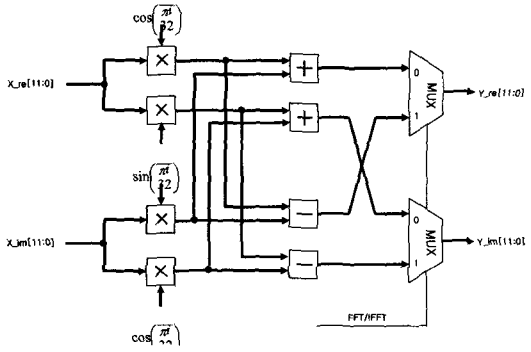


그림 10. $W_{64}^{\pm i}$ 승산부의 블록도
Fig. 10. Block diagram of the $W_{64}^{\pm i}$ multiplier.

수식을 통해, $\cos(\pi i/32)$와 $\sin(\pi i/32)$ 값을 입력 받아 복소수 입력데이터에 승산하고, 이 값들을 가산 및 감

산함으로써 실제 승산부를 구현할 수 있음을 알 수 있다. 트위들팩터 $W_8^{\pm i}$의 승산과 마찬가지로 FFT 연산과 IFFT 연산을 모두 지원해야 하므로 MODE 신호를 입력받아 출력값을 선택해 주는 멀티플렉서가 필요하다. 그림 10은 $W_{64}^{\pm i}$ 승산부의 블록도를 보여준다.

다. 지연교환기

제안된 알고리즘을 이용하여 64-point FFT 프로세서를 MDC 파이프라인 방식으로 구현하기 위해서는 3개의 지연교환기가 필요하다. 이는 프로세서의 구현을 위해 그림 11과 같이 3번의 데이터 정렬이 요구되기 때문이다. 이들 각각의 정렬 방식이 다르기 때문에 각각의 특성에 맞는 별도의 지연교환기가 필요하다.

(1) 지연교환기 1

그림 11에서 볼 수 있듯이 지연교환기 1은 4개의 입력데이터를 시간 지연시킨 후, 이들 데이터를 스위치를 통해 경로를 변경시키고 다시 시간 지연시킴으로써 구현된다. 시간 지연부는 24비트 레지스터를 통해 구현되었으며, 스위치부는 4개의 멀티플렉서를 통해 구현하였다. 그림 12는 스위치부의 블록도를 그림 13은 지연교환기1의 블록도를 각각 보여준다.

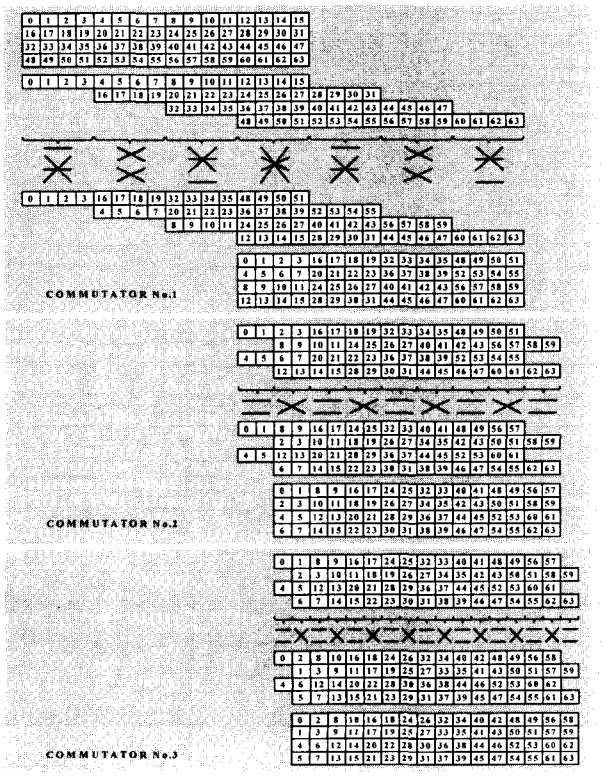


그림 11. 데이터 정렬 패턴.
Fig. 11. Data reordering pattern.

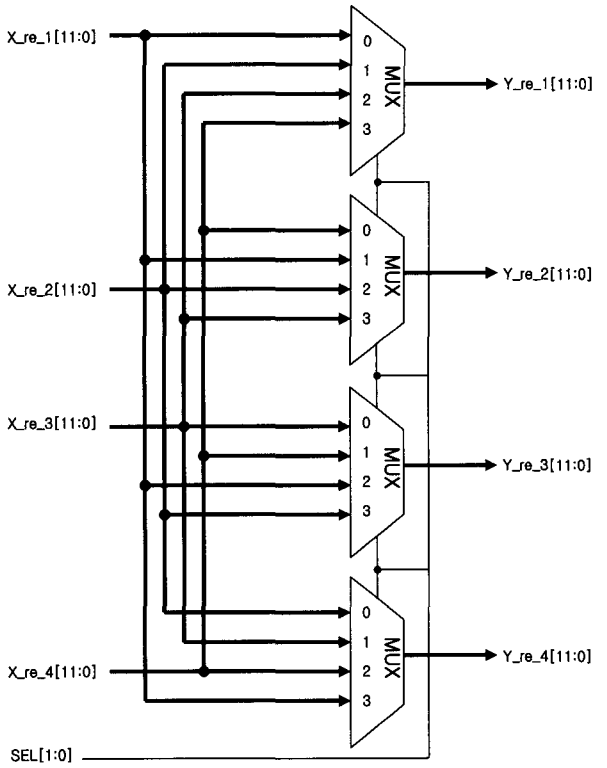


그림 12. 스위치부의 블록도
Fig. 12. Block diagram of the switch part.

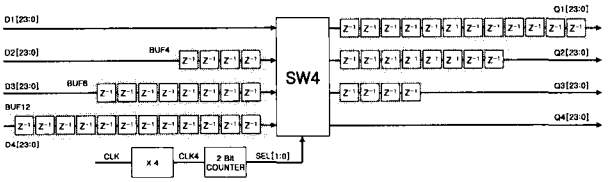


그림 13. 지연교환기1의 블록도
Fig. 13. Block diagram of the delay commutator 1.

(2) 지연교환기 2

지연교환기 2는 지연교환기 1과 다르게 radix-2 기반의 데이터 교환을 수행해야 한다. 따라서 2개의 입력을 갖는 지연교환기를 병렬로 배치함으로써 지연교환기 2를 구현하였다. 그림 14와 15는 지연교환기 2를 위한 2입력 스위치부와 지연교환기 2의 블록도를 각각 보여준다.

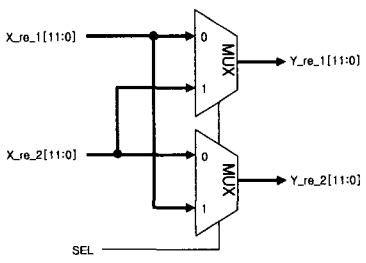


그림 14. 2-입력스위치부의 블록도
Fig. 14. Block diagram of the 2-input switch part.

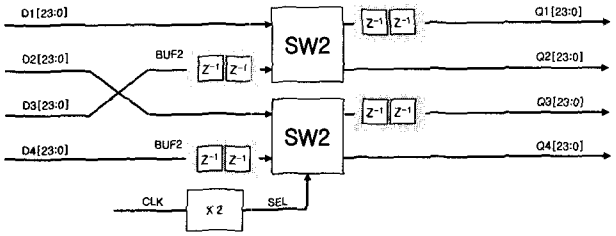


그림 15. 지연교환기2의 블록도
Fig. 15. Block diagram of the delay commutator 2.

(3) 지연교환기 3

지연교환기 2에서처럼 지연교환기 3 역시 radix-2 기반의 데이터 교환을 수행한다. 지연교환기 2가 2클럭 주기만큼의 시간지연을 요구하는 반해, 지연교환기 3은 1클럭 주기만큼의 시간지연을 요구하므로 그림 16과 같은 구조로 설계되었다.

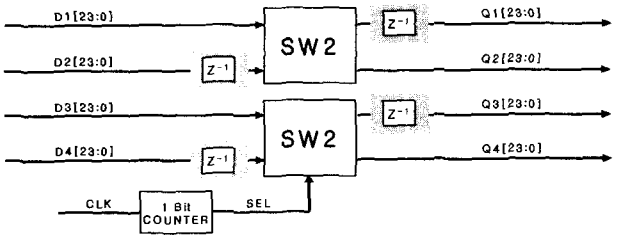


그림 16. 지연교환기3의 블록도
Fig. 16. Block diagram of the delay commutator 3.

V. 시뮬레이션 및 논리합성 결과

1. 시뮬레이션 결과

설계된 FFT 프로세서는 Xilinx FPGA 기반으로 타이밍 검증 되었다. 검증을 위한 테스트 벡터로 실수항은 0.03125의 값을 갖고, 허수항은 0 값을 갖는 사각파 두 심볼을 이용하였다. 각각의 데이터 심볼과 MODE 신호는 3.2us의 주기를 가지고 있고, 첫 번째 데이터가 입력됨과 동시에 50ns의 펄스폭을 갖는 START 신호를 입력하였다. 그림 17은 입력 테스트 벡터를 보여 준다.

그림 18은 FFT 연산에 대한 검증 결과를 보여 준다. START 신호와 함께 3.2us의 주기를 갖는 심볼이 입력되면, 20MHz 클록으로 128클록(6.4us) 지연 후에 첫 번째 출력 심볼과 DONE 신호가 출력됨을 확인할 수 있다. FFT 연산 후의 출력 심볼은 입력 심볼과 마찬가지로, 3.2us의 주기를 갖고 0.8us의 보호구간을 갖는다. 연속적으로 출력되는 두 번째 심볼에 대해서도 첫 번째 심볼과 동일한 결과를 얻음을 확인할 수 있다.

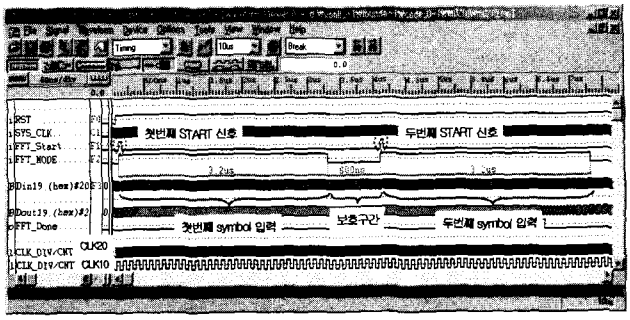


그림 17. 입력 테스트 벡터
Fig. 17. Input test vector.

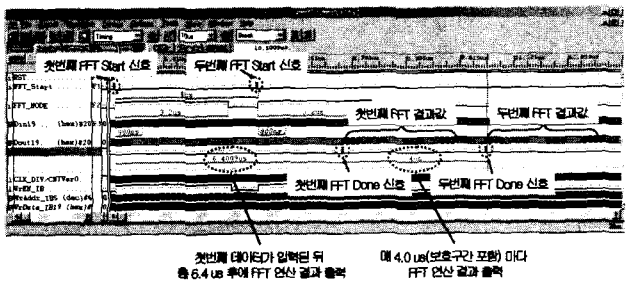


그림 18. 설계된 FFT 프로세서의 시뮬레이션 결과
Fig. 18. Simulation results of the proposed FFT processor.

2. 논리합성 결과

표 2는 제안된 알고리즘을 이용하여 설계된 64-point FFT 프로세서의 논리합성 결과를 보여준다. 시놉시스 캐드 툴을 이용하여 0.6㎛ 공정으로 합성한 결과 총 게이트 수는 39,506개였다.

제안된 알고리즘의 성능을 기존의 radix-4 알고리즘과 비교하기 위해, 같은 MDC 파이프라인 방식으로 FFT 프로세서를 구현한 결과가 표 3에 보여진다.

표 2. 제안된 알고리즘을 이용하여 설계된 FFT 프로세서의 논리합성 결과
Table 2. Logic synthesis results of the proposed FFT processor.

블 록 명	게이트수	블록수	총게이트수
1. Radix-4 BF	1,964	2	3,928
2. Radix-2 BF	459	2	918
3. TF 승산기(W_8)	1,827	2	3,654
4. TF 승산기(W_{64})	19,153	1	19,153
5. 지연교환기 1	9,350	1	9,350
6. 지연교환기 2	1,479	1	1,479
7. 지연교환기 3	1,024	1	1,024
합 계			39,506

표 3. Radix-4 FFT 프로세서의 논리합성 결과
Table 3. Logic synthesis results of the radix-4 FFT processor.

블 록 명	게이트수	블록수	총게이트수
1. Radix-4 BF	1,964	3	5,892
2. TF 승산기(W_{64})	19,153	2	38,306
3. 지연교환기 1	9,350	1	9,350
4. 지연교환기 2	2,820	1	2,820
합 계			56,368

표 2와 3의 논리합성 결과를 통해 제안된 알고리즘과 기존의 radix-4 알고리즘의 성능을 비교해 보면 다음과 같다. 우선, 두 알고리즘 모두 radix-4 버터플라이 연산자를 기반으로 구현되므로 입력 및 출력 데이터 율을 R로 가정하는 경우, $4R$의 처리율을 갖는다. 그러나 면적면에서 제안된 알고리즘의 논리합성 결과가 39,506개임에 비해, radix-4 알고리즘의 결과는 56,368개이므로, 제안된 알고리즘이 기존의 radix-4 알고리즘에 비해 약 30%의 정도 면적의 감소를 가져옴을 알 수 있다. 표 4는 성능 비교 결과를 정리하여 제시해 준다.

표 4. 제안된 알고리즘과 기존의 radix-4 알고리즘의 성능 비교
Table 4. Performance comparison of the proposed and radix-4 FFT algorithm.

알 고 리 즘	처리율	면적(논리합성 결과)
Radix-4 Algorithm	$4R$	39,506
Proposed Algorithm	$4R$	56,368

VI. 결 론

본 논문에서는 고속 멀티미디어 통신 시스템을 위한 효율적인 FFT 알고리즘이 제안되었고, 이의 하드웨어 구현 및 검증 결과가 제시되었다. 제안된 알고리즘은 radix-4 버터플라이 연산자를 기반으로 구현되므로 기존의 radix-2^3 알고리즘에 비해 2배의 처리율을 가지며, 또한 radix-2^3 알고리즘의 비단순 승산기의 수를 줄이는 특성을 그대로 이용하므로, 같은 처리율을 갖는 기존의 radix-4 알고리즘에 비해 저면적의 구현이 가능하다. 처리율의 비교를 위해, radix-2 및 radix-4 알고리즘을 MDC 파이프라인 방식에 적용하여 비교 분석하였고, 이를 통해 제안된 알고리즘이 기존의 radix-2^3 알고리즘에 비해 2배의 처리율을 가짐을 확인하였다. 또한 면적의 비교를 위해, 제안된 알고리즘과 기존의 radix-4 알고리즘을 하드웨어로 구현하였고, 0.6㎛ 공정으로 논리 합성한 결과, 제안된 알고리즘은 총 39,506개의 논리 게이트로 구성되고, 기존의 radix-4 알고리즘은 총 56,368개의 논리 게이트로 구성됨을 확인하였다. 이를 통해, 제안된 알고리즘을 이용하는 경우, 기존의 알고리즘에 비해 면적 면에서 약 30% 정도의 이득을 얻을 수 있음을 알 수 있었다. 이처럼 제안된 알고리즘은 고속 동작이 가능하며, 동시에 저면적 구현이 가능하므로 무선 LAN 시스템, DAB 및 DVB 시스템, ADSL/VDSL 시스템 등 고속 멀티미디어 통신 시스템에 적합한 알고리즘이라 할 수 있다.

참 고 문 헌

[1] E. F. Casa and C. Leung, "OFDM for Data Communication over Mobile Radio Channels", IEEE Trans. on Comm., Vol. 39, No. 5, pp. 783-793, May 1991.

[2] J. A. C. Bingham, "Multi-carrier Modulation for Data Transmission : An Idea Whose Time Has Come", IEEE Comm. Magazine., Vol. 28, No. 5, pp. 5-14, May 1990.

[3] S. Bertazzoni, G. C. Cardarilli, M. Iannuccelli, M. Salmeri, A. Salsano, and O. Simonelli, "16-Point High Speed (I)FFT For OFDM Modulation", ISCAS98, Vol. 5, pp. 210-212, 1998.

[4] Shousheng He and Mats Torkelson, "Designing Pipeline FFT Processor for OFDM (de)Modulation", ISSSE'98, Vol. 2, pp. 945-950, 1998.

[5] Bevan M. Baas, "A 9.5㎽ 330㎲ 1024-point FFT Processor", IEEE Custom Integrated Circuits Conference, pp. 127-130, 1998.

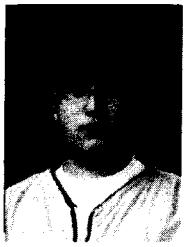
[6] E. H. Wold and A. M. Despain, "Pipeline and Parallel Pipeline FFT Processors for VLSI Implementation", IEEE Trans. on Comput., C-33(5), pp. 414-426, May 1984.

[7] H. Stones, "Parallel Processing with the Perfect Shuffle", IEEE Trans. on Comput., pp. 156-161, Feb. 1971.

[8] Neil Weste and David J. Skellern, "VLSI for OFDM", IEEE Communication Magazine, Vol. 36, No. 10, pp. 127-131, Oct. 1998.

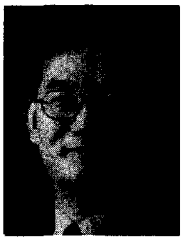
[9] Earl E. Swartzlander, "VLSI Signal Processing Systems", Kluwer Academic Publishers, pp. 125-133, 1986.

저 자 소 개



정 윤 호(정회원)

1998년 연세대학교 전자공학과 학사 졸업.
2000년 연세대학교 전자공학과 석사 졸업.
2000년~현재 연세대학교 전기 전자공학과 박사과정.
<주관심분야 : 유무선 통신시스템 알고리즘 개발, 통신시스템용 SoC 설계>



김 재 석(정회원)

1977년 연세대학교 전자공학과 학사 졸업.
1979년 한국과학기술원 전기전자공학과 석사 졸업.
1988년 Rensselaer Polytechnic Institutes NY NSA(Ph.D)
1993년~1995년 한국전자통신연구원 책임연구원
1996년~현재 연세대학교 전기전자공학과 교수
<주관심분야 : 유무선 통신시스템, 영상처리시스템, 통신 및 영상처리시스템용 SoC 설계>