논문 2005-42SP-2-15

저 전력 및 면적 효율적인 알고리즘 기반 고속 퓨리어 변환 프로세서

(Fast Fourier Transform Processor based on Low-power and Area-efficient Algorithm)

오 정 열*, 임 명 섭*

(Jung-yeol Oh and Myoung-seob Lim)

요 약

본 논문에서는 OFDM 시스템에 적용하기 위한 새로운 Radix-24 FFT 알고리즘을 제안하고 이 알고리즘을 기반으로 하는 효율적인 파이프라인 FFT 프로세서 구조를 제안한다. Radix-24 알고리즘 기반의 파이프라인 FFT 구조는 Radix-22 알고리즘 구조와 같은 개수의 곱셈기를 가지고 있으나, 전체 프로그래머블 복소 곱셈기의 절반에 해당하는 곱셈기를 본 논문에서 제안한 CSD(Canonic Signed Digit) 상수 복소 곱셈기로 대체하여 곱셈기의 복잡도를 30%이상 줄이는 효과가 있다. 0.35um CMOS 삼성공정의 합성 시뮬레이션을 통해 제안한 CSD 상수 복소 곱셈기는 기존의 프로그래머블 복소 곱셈기에 비교하여 60%이상 면적효율을 갖는 것으로 분석되었다. 이러한 FFT 구조는 면적과 전력 면에서 높은 효율을 필요로 하는 무선 OFDM 응용분야에 핵심 블록인 큰 포인트 크기를 갖는 FFT 프로세서 설계에 효과적으로 적용될 것이다.

Abstract

This paper proposes a new radix-2^4 FFT algorithm and an efficient pipeline architecture based on this new algorithm for OFDM systems. The pipeline architecture based on the new algorithm has the same number of multipliers as that of the radix-2^2 algorithm. However, the multiplier complexity could be reduced by more than 30% by replacing one half of the programmable complex multipliers by the newly proposed CSD constant complex multipliers. From synthesis simulations of a standard 0.35um CMOS Samsung process, a proposed CSD constant complex multiplier achieved more than 60% area efficiency when compared with the conventional programmable complex multiplier. This promoted efficiency can be used for the design of a long length FFT processor in wireless OFDM applications which needs more power and area efficiency.

Keywords : FFT, Radix-2^4, Radix-2^2, CSD, 복소 곱셈기

I. 서 론

FFT(Fast Fourier Transform)과 IFFT (Inverse FFT)는 OFDM (Orthogonal Frequency Division Multiplexing) 변복조 시스템의 핵심블록이다. 최근 무선 OFDM 응용분야에서는 큰 포인트 크기뿐만 아니라, 고성능 및 저 전력 FFT프로세서에 대한 요구가 증가하고 있다. 기존의 FFT의 구조는 크게 단일 메모리 구조와 이중 메모리 구조 그리고, 파이프라인 구조로 나눌 수 있다. 단일 메모리 구조는 적은 하드웨어를 가진다는 장점이 있으나, 연속처리가 불가능 하고, 따라서 속도가 느린 단점이 있다. 이중 메모리 구조는 복잡한 메모리 어드레싱 없이 연속처리가 용이하다는 장점이 있으나 역시 고속연산에는 부적합하다. 파이프라인 구조는 고성능 연산을 처리할 수 있어 고속데이터 전송방식

* 정회원, 전북대학교
(Chonbuk National University)
※ 본 연구는 정보통신부 대학 IT연구센터 육성.지원사업의 연구결과로 수행되었습니다.
※ This work was supported(in part) by the Ministry of Information & Communications, Korea, under the Information Technology Research Center (ITRC) Support Program.
접수일자: 2004년9월21일, 수정완료일: 2004년11월10일

에 적합하다. 반면에 파이프라인 구조는 비교적 큰 하드웨어 면적을 요구하므로 비용이 크고, 전력소모가 큰 단점이 있다.[1] 광대역 OFDM 무선 전송방식에 파이프라인 FFT 탕식을 적용하기 위해서는 하드웨어 면적을 최소화 하여 면적과 전력 면에서 효율적인 알고리즘을 적용해야 한다.

다양한 FFT 알고리즘 중에서, Cooley-Turkey 알고리즘이 가장 널리 쓰인다. 그 이유는 DFT의 복소 계산량인 $O(N^2)$을 $O(N\log_2 N)$으로 줄일 수 있기 때문이다. 또한 알고리즘의 규칙적인 특성으로 VLSI 구현이 용이한 장점이 있다. 이에 더 계산량을 줄이기 위한 방안으로, Radix-4, Split-radix, radix-2^2, radix-2/4/8, 고차 Radix 버전 등이 제안되었다. 일반적으로 이러한 알고리즘의 공통적인 원리는 길이 $N(=2^n)$ FFT를 홀수항과 짝수항으로 반복적으로 분리하고, FFT 연산의 대칭성을 이용하여 효과적으로 곱셈량의 횟수를 줄이는데 있다.

최근 30년간 OFDM 기술발전과 더불어 여러 파이프라인 FFT 구조가 제안되었다. 크게 Radix-2구조와 Radix-4 구조 그리고 Split-radix 기반 파이프라인 FFT 구조가 있다. Split-radix는 구조의 비대칭적인 취약점 때문에 파이프라인 구조에 적용하기에 용이하지 않고, Radix-2와 Radix-4의 MDC (Multi-path Delay Commutator), SDC (Single-path Delay Commutator) 및 SDF (Single-path Delay Feedback) 구조로 나눌 수 있다. 각 파이프라인 FFT 구조는 서로 다른 장점을 가지고 있으나 이를 요약하여 보면 다음 표 1과 같다.[2] 복소 곱셈기와 복소 덧셈기의 관점에서 보면 R4SDC 구조가 가장 효율적이나 메모리가 R2^2SDF 구조보다 두 배 크고 제어가 복잡하여, 종합적인 관점에서 보면 R2^2SDF 구조가 가장 효율적인 구조라고 할 수 있다.

R2^2SDF 구조의 기반인 Radix-2^2과 Radix-2^3 FFT알고리즘은 1998년 He와 Torkeson이 제안하였다.[3] 이 알고리즘들은 기존의 Radix-2 알고리즘의 구조를 개량한 것으로 Radix-4 구조의 곱셈량과 Radix-2 구조의 간단한 연산구조를 복합 수용하여 복잡한 곱셈을 효과적으로 줄이고, 제어를 쉽게 만든 구조이다.

반면에, 파이프라인 FFT 프로세서 구현에 있어서 복소 곱셈기의 설계는 여전히 중요한 관건이다. 왜냐하면 하나의 복소 곱셈기는 4개의 실수 곱셈기와 2개의 실수 덧셈기를 가지고 있어, 큰 면적뿐만 아니라 총 전력소모의 50~80%까지 소모하기 때문이다.[4]

본 논문에서는 복소 곱셈기의 면적과 전력소모를 줄

표 1. 파이프라인 FFT구조의 복잡도 비교
Table 1. Hardware requirement comparison of the pipeline FFT architectures.

	복소곱셈기	복소덧셈기	메모리	제어
R2MDC	$2\log_4 N$-1	$4\log_4 N$	$3N/2$-2	simple
R2SDF	$2\log_4 N$-1	$4\log_4 N$	N-1	simple
R4SDF	$\log_4 N$-1	$8\log_4 N$	N-1	medium
R4MDC	$3(\log_4 N$-1)	$8\log_4 N$	$5N/2$-4	simple
R4SDC	$\log_4 N$-1	$3\log_4 N$	$2N$-2	complex
R2^2SDF	$\log_4 N$-1	$4\log_4 N$	N-1	simple

이는 방안으로 Radix-2^4 알고리즘을 제안하고, R2^2SDF 구조를 개량하여, 전체 프로그래머블 복소 곱셈기의 절반을 새로운 CSD(Canonic Signed Digit) 상수 곱셈기로 대체하는 R2^4SDF 구조를 제안한다.

본 논문의 구성은 Ⅱ장에서 Radix-2^4 알고리즘을 전개하며, Ⅲ장에서는 제안한 알고리즘을 기반으로 하는 R2^4SDF 파이프라인 구조를 제안하고 이를 R2^2SDF 구조와 비교한다. Ⅳ장에서는 제안하는 CSD 상수 복소 곱셈기와 프로그래머블 복소 곱셈기를 설계하여 이를 합성하여 그 면적을 비교 평가하고, 마지막으로 Ⅴ장에서 결론을 맺는다.

Ⅱ. Radix-2^n 알고리즘

N-point DFT(Discrete Fourier Transform) 는 다음과 같이 주어진다.

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

여기서, $W_N=e^{-j(2\pi/N)}$ 이며, 아래첨자 n은 시간 인덱스 그리고 k는 주파수 인덱스이다. 이를 이산 퓨리어 변환의 회전인자로서, 이를 연속적인 분해법(Cascade decomposition)을 통해 회전인자의 배치특성을 향상시키는 Radix-2^n 알고리즘을 전개할 수 있다.

1. Radix-2^2 알고리즘

Radix-2^2 알고리즘은 Radix-2 알고리즘을 발전시켜, 시간과 주파수 인덱스 n과 k를 3차원으로 분리하고 첫 두개의 스테이지를 동시에 전개하게 된다.[3]

$$n=\frac{N}{2}n_1+\frac{N}{4}n_2+n_3\left\{0\le n_1,n_2\le 1, 0\le n_3\le \frac{N}{4}-1\right\}$$
$$k=k_1+2k_2+4k_3 \quad \left\{0\le k_1,k_2\le 1, 0\le k_3\le \frac{N}{4}-1\right\} \quad (2)$$

수식(1)에 대입하여 n_1에 관하여 전개하면 수식(3)과 같다.

$$X(k_1+2k_2+4k_3) = \sum_{n_1=0}^{\frac{N}{4}-1}\sum_{n_2=0}^{1}\left\{B_{\frac{N}{2}}^{k_1}\left(\frac{N}{4}n_2+n_3\right)W_N^{\left(\frac{N}{4}n_2+n_3\right)}\right\}W_N^{\left(\frac{N}{4}n_2+n_3\right)(k_1+2k_2+4k_3)} \quad (3)$$

첫 번째 스테이지 버터플라이 구조는 수식(4)와 같다.

$$B_{\frac{N}{2}}^{k_1}\left(\frac{N}{4}n_2+n_3\right)=x\left(\frac{N}{4}n_2+n_3\right)+(-1)^{k_1}x\left(\frac{N}{4}n_2+n_3+\frac{N}{2}\right) \quad (4)$$

수식(3)의 회전계수는 수식(5)와 같이 전개된다.

$$W_N^{\left(\frac{N}{4}n_2+n_3\right)(k_1+2k_2+4k_3)}=(-j)^{n_2(k_1+2k_2)}W_N^{n_3(k_1+2k_2)}W_{\frac{N}{4}}^{n_3k_3} \quad (5)$$

수식(5)를 수식(3)에 대입하고, n_2에 관하여 전개하면 수식(6)과 같다.

$$X(k_1+2k_2+4k_3)=\sum_{n_3=0}^{\frac{N}{4}-1}\left[H_{\frac{N}{4}}^{k_1k_2}(n_3)W_N^{n_3(k_1+2k_2)}\right]W_N^{4n_3k_3} \quad (6)$$

두 번째 스테이지 버터플라이 구조는 수식(7)과 같다.

$$H_{\frac{N}{4}}^{k_1k_2}(n_3)=\underbrace{\overbrace{B_{\frac{N}{2}}^{k_1}(n_3)}^{\text{BF I}}+(-j)^{k_1+2k_2}\overbrace{B_{\frac{N}{2}}^{k_1}\left(n_3+\frac{N}{4}\right)}^{\text{BF I}}}_{\text{BF II}} \quad (7)$$

이와 같이 두 번째 스테이지 까지 전개한 후, 남은 N/4 포인트 길이의 DFT 연산은 위와 같은 방식으로 반복적인 절차에 의해 전개되어 진다. 수식(7)의 $(-j)$ 연산은 곱셈기 대신 입력신호의 실수부와 허수부를 교환하고 허수부의 부호를 바꾸어 준다. Radix-2^2 알고리즘의 장점은 두 스테이지가 지난 후 에 프로그래머블 곱셈기가 나타나게 되므로, Radix-2의 간단한 버터플라이 연산 구조를 가지면서, 복소 곱셈기는 매 두개의 스테이지후에 배치한다는 것이다. 따라서 파이프라인 구조에 적용하였을 때, Radix-4구조의 곱셈기의 개수만을 필요로 한다. 그림 1은 N이 32일 때 Radix-2^2 신호흐름도의 예를 보여준다. 첫 스테이지후에는 다이아몬드(◇) 기호로 표기된 (-j)연산 블록이 있고, 두 번째 스테이지 후에는 삼각형(▷) 기호로 표시되어있는 프로그래머블 복소 곱셈기를 요구하는 여러 종류의 회전인자블록이 있다.

2. Radix-2^3 알고리즘

Radix-2^3 알고리즘을 Radix-2^2 구조를 발전시켜 세 개의 스테이지까지 동시에 분리한다. 이를 위해 4차원 인덱스 맵을 사용한다.[3]

$$n=\left\langle\frac{N}{2}n_1+\frac{N}{4}n_2+\frac{N}{8}n_3+n_4\right\rangle_N$$
$$k=\left\langle k_1+2k_2+4k_3+8k_4\right\rangle_N \quad (8)$$

수식(1)에 대입한 후, n_1,n_2,n_3를 차례로 전개하면 수식(9)와 같이 전개되고, 수식(10) 같은 세 번째 스테이지 버터플라이 구조를 얻을 수 있다.

$$X(k_1+2k_2+4k_3+8k_4)=\sum_{n_4=0}^{\frac{N}{8}-1}\left[T_{\frac{N}{8}}^{k_1k_2k_3}(n_4)W_N^{n_4(k_1+2k_2+4k_3)}\right]W_{N/8}^{n_4k_4} \quad (9)$$

$$T_{\frac{N}{8}}^{k_1k_2k_3}(n_4)=\underbrace{\overbrace{H_{\frac{N}{4}}^{k_1k_2}(n_4)}^{\text{BF II}}+(-1)^{k_3}W_8^{(k_1+2k_2)}\overbrace{H_{\frac{N}{4}}^{k_1k_2}\left(n_4+\frac{N}{8}\right)}^{\text{BF II}}}_{\text{BF III}} \quad (10)$$

수식(10)처럼 세 번째 스테이지 버터플라이 구조는 두 번째 스테이지 버터플라이 간에 W_8^n의 회전인자 곱셈을 요구한다. 그림 2는 N이 32일 때 Radix-2^3 신호흐름도의 예를 보여준다. 첫 스테이지후에는 다이아몬드(◇) 기호로 표기된 (-j)연산 블록이 있고, 두 번째 스테이지 후의 사각형(□) 기호는 W_8^n의 회전인자 곱셈을 위한 상수 복소 곱셈기이고, 세 번째 스테이지 후에 삼각형(▷) 기호로 표시되어있는 프로그래머블 복소 곱셈기를 요구하는 여러 종류의 회전인자블록이 있다.

2. Radix-2^4 알고리즘

Radix-2^4 알고리즘은 n과 k를 5 차 선형 인덱스 맵으로 수식(11)와 같이 분리한다.

$$n=\left\langle\frac{N}{2}n_1+\frac{N}{4}n_2+\frac{N}{8}n_3+\frac{N}{16}n_4+n_5\right\rangle_N$$
$$k=\left\langle k_1+2k_2+4k_3+8k_4+16k_5\right\rangle_N \quad (11)$$

Radix-2^4 알고리즘을 전개하기 위해 처음 네 번째 스테이지까지 동시에 분리한다. 수식(11)를 수식(1)에 대입하여 후, n_1,n_2,n_3,n_4를 차례로 전개하면 수식 (12)과 같은 네 번째 스테이지 버터플라이 구조를 얻을 수 있다.

$$X(k_1+2k_2+4k_3+8k_4+16k_5)=\sum_{n_5=0}^{\frac{N}{16}-1}\left[G_{\frac{N}{16}}^{k_1,k_2,k_3,k_4}(n_5)W_N^{n_5(k_1+2k_2+4k_3+8k_4)}\right]W_{N/16}^{n_5k_5} \quad (12)$$

$$G_{\frac{N}{16}}^{k_1,k_2,k_3,k_4}(n_5)=\overbrace{\left\{H_{\frac{N}{4}}^{k_1,k_2}(n_5)+(-1)^{k_3}W_{16}^{2(k_1+2k_2)}H_{\frac{N}{4}}^{k_1,k_2}(n_5+\frac{N}{8})\right\}}^{\text{BF III}}+(-1)^{k_4}(-j)^{k_3}\overbrace{\left\{W_{16}^{(k_1+2k_2)}H_{\frac{N}{4}}^{k_1,k_2}\left(n_5+\frac{N}{16}\right)+(-1)^{k_3}W_{16}^{3(k_1+2k_2)}H_{\frac{N}{4}}^{k_1,k_2}(n_5+\frac{3N}{16})\right\}}^{\text{BF III}}\quad(15)$$

$$\text{BF IV}$$

회전인자는 다음 수식(13)과 같이 분리되었다.

$$W_N^{nk}=\left\{(-1)^{n_1k_1}(-j)^{n_2(k_1+2k_2)}W_8^{n_3(k_1+2k_2+4k_3)}\cdot W_{16}^{n_4(k_1+2k_2+4k_3+8k_4)}\right\}W_N^{n_5(k_1+2k_2+4k_3+8k_4)}W_{N/16}^{n_5k_5}\quad(13)$$

네 번째 스테이지 버터플라이 연산은 아래의 수식(14)와 같다.

$$G_{\frac{N}{16}}^{k_1,k_2,k_3,k_4}(n_5)=T_{\frac{N}{8}}^{k_1,k_2,k_3}(n_5)+W_{16}^{(k_1+2k_2+4k_3+8k_4)}T_{\frac{N}{8}}^{k_1,k_2,k_3}\left(n_5+\frac{N}{16}\right)\quad(14)$$

네 번째 스테이지의 버터플라이 구조는 세 번째 스테이지 버터플라이 간에 W_{16}^n의 회전인자 곱셈을 요구하게 된다. 그러나 이러한 회전인자 중 (-j) 연산을 제외한 곱셈을 세 번째 스테이지 버터플라이 구조에 포함시킬 수 있다. 이러한 수정된 네 번째 스테이지 버터플라이 구조는 이전 페이지 하단의 수식(15)와 같다. 수식(15)의 회전인자 W_{16}^n은 W_{16}^0, W_{16}^1, W_{16}^2, W_{16}^3의 4개의 복소수 계수로 정의된다. 본 논문에서는 회전인자 W_{16}^n를 위해 프로그래머블 복소 곱셈기 대신 새로운 CSD 상수 복소 곱셈기를 설계한다. 상수 곱셈기는 고정된 계수의 "1"에 해당하는 부분곱을 생성하여 시프트와 덧셈기만을 가지고 설계하는 방법이다. W_{16}^0은 "1"에 해당되기 때문에 입력데이터를 어떤 위상의 회전 없이 단순히 곱셈기를 통과하는 것을 의미한다.

W_{16}^1는 $\cos\pi/8-j\sin\pi/8$에 해당하는 복소수로서 하나의 상수 복소 곱셈기로 설계할 수 있다. W_{16}^2 또한 $\cos\pi/4-j\sin\pi/4$에 해당하는 복소수이다. 반면에, 이 회전계수는 삼각함수공식 수식 (16)을 적용하면 회전계수 W_{16}^1을 이용하여 W_{16}^2값을 구할 수 있다.

$$\cos\frac{\pi}{4}=\sin\frac{\pi}{4}=2\sin\frac{\pi}{8}\cos\frac{\pi}{8}\quad(16)$$

W_{16}^3은 $\sin\pi/8-j\cos\pi/8$에 해당하는 복소수이다. 이것은 단지 회전인자 W_{16}^1의 실수부와 허수부가 자리바꿈을 한 것이기 때문에, 곱셈기 내부의 덧셈과 뺄셈연산을 바꾸어주면, 회전인자 W_{16}^1을 연산하는 방법과 동일하게 구할 수 있다. 따라서 회전인자 W_{16}^n을 위한 복소곱셈은 약간의 제어로직이 추가된 하나의 상수 복소 곱셈기로 설계할 수 있다. 또한 상수 곱셈기에 CSD (Canonic Signed Digit)의 표현방식을 적용하면 최소한의 1의 개수로 계수를 표현해 주기 때문에 면적과 전력소모 면에서 일반 상수 곱셈기보다 33%의 효율이 있다.[5][6] Radix-2^4 알고리즘에서 프로그래머블 복소 곱셈기는 네 번째 스테이지 이후 수식 (12)의 분해된 회전인자 $W_N^{n_5(k_1+2k_2+4k_3+8k_4)}$를 적용하기 위해 사용한다. 위 절차를 반복하여 전체 N 사이즈에 대한 Radix-2^4 FFT 흐름제어를 완성할 수 있다.

그림 3은 N이 32일 때 Radix-2^4 신호흐름도의 예를 보여준다. 여기에서 다이아몬드(◇) 기호는 -j에 의한 간소한 곱셈을 표현하고, 두 번째 스테이지의 사각형(□) 기호는 제안된 CSD 상수 복소 곱셈기로 구현할 수 있는 곱셈연산을 표현한 것이다. 네 번째 스테이지의 삼각형(▷) 기호는 프로그래머블 복소 곱셈기를 필요로 하는 곱셈연산을 표현한 것이다.

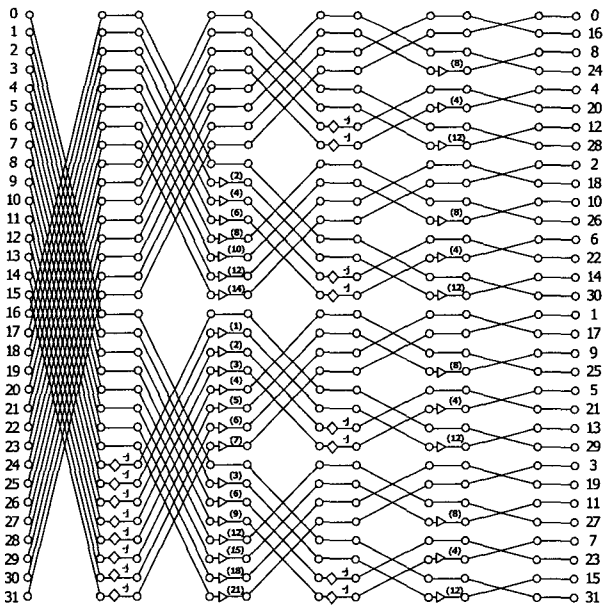


그림 1. Radix-2^2 32-point FFT 신호흐름도
{(n)은 W_{32}^n을 의미한다}

Fig. 1. Radix-2^2 32-point FFT signal flow.
{(n) defines W_{32}^n}

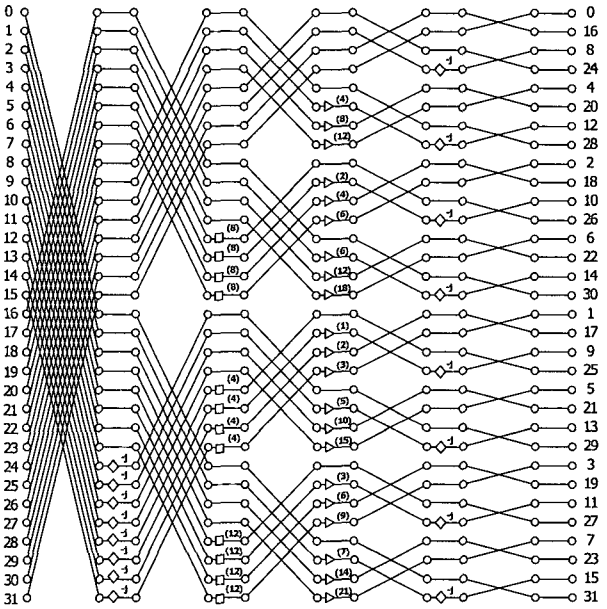


그림 2. Radix-2^3 32-point FFT 신호흐름도
{(n)은 W_{32}^n을 의미한다}

Fig. 2. Radix-2^3 32-point FFT signal flow.
{(n) defines W_{32}^n}

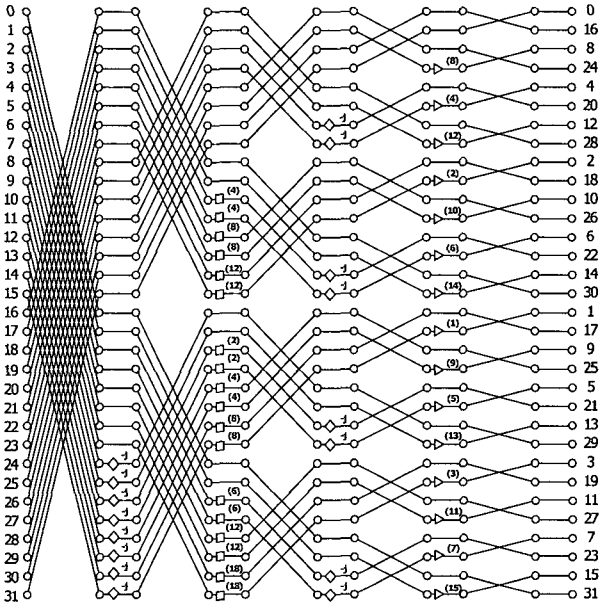


그림 3. Radix-2^4 32-point FFT 신호흐름도
{(n)은 W_{32}^n을 의미한다}

Fig. 3. Radix-2^4 32-point FFT signal flow.
{(n) defines W_{32}^n}

표 2는 회전인자 W_{16}^1의 곱셈연산을 위한 계수를 12 비트로 2의 보수와 CSD 방식으로 표현한 것이다. 입력 신호가 $x+jy$로 표현될 때, 회전인자 W_{16}^n 연산을 위한 제안된 CSD 상수 복소 곱셈기의 구조와 절차는 표 3과 그림 4에 보인다.

표 2. 계수의 이진 표현값 ($\bar{1}$은 -1을 의미한다)

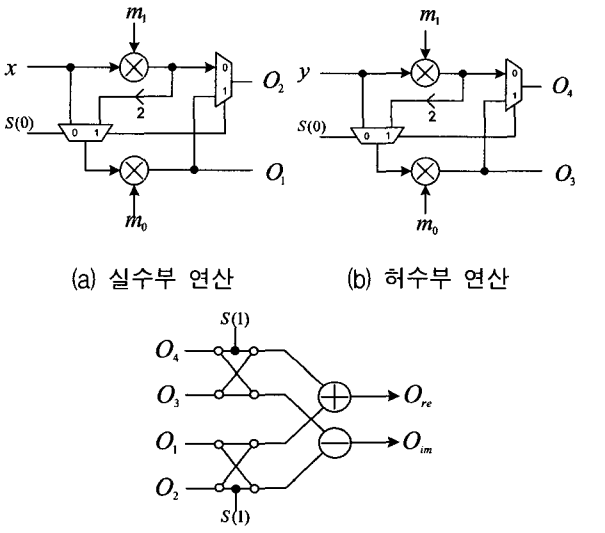
Table 2. Binary representation of the coefficients.

계수		십진수	2의 보수	CSD
m_0	$\cos(\pi/8)$	0.9239	011101100100	$1000\bar{1}0\bar{1}00100$
m_1	$\sin(\pi/8)$	0.3827	001100001111	$010\bar{1}0001000\bar{1}$

표 3. CSD 상수 복소 곱셈기 연산 절차

Table 3. CSD constant complex multiplication procedure.

s(1)	s(0)	O_1	O_2	O_3	O_4	Outputs
0	0	xm_0	xm_1	ym_0	ym_1	$(O_1+O_4)+j(O_3-O_2)$
0	1	$2xm_0m_1$	$2xm_0m_1$	$2ym_0m_1$	$2ym_0m_1$	$(O_1+O_4)+j(O_3-O_2)$
1	0	xm_0	xm_1	ym_0	ym_1	$(O_2+O_3)+j(O_4-O_1)$
1	1	bypass mode				$x+jy$



(a) 실수부 연산 (b) 허수부 연산

(c) 최종 덧셈연산부

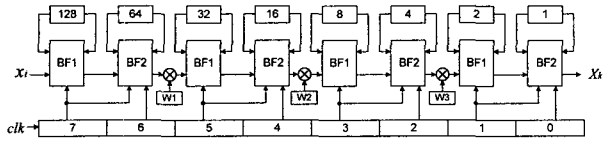
그림 4. 제안된 CSD 상수 복소 곱셈기

Fig. 4. The proposed CSD constant complex multiplier.

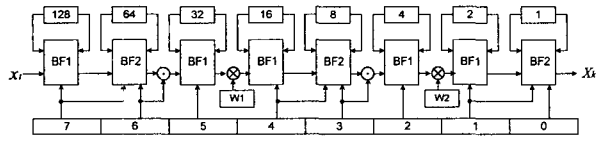
III. Radix-2^n 파이프라인 FFT 구조

그림 5는 256-point R2^{n}SDF(Radix-2^n Single-path Delay Feedback) FFT구조를 서로 비교한 그림이다. 각 스테이지에 사용되는 버터플라이 구조는 BF1과 BF2로 표현하였다. BF2은 BF1구조에 (-j)연산을 위한 스위치 블록을 포함한 것이다.[3] 그림 5에서 ⊗는 프로그래머블 복소 곱셈기를 의미하고 ⊙는 CSD 상수 복소 곱셈기를 의미한다. 그림 5(a)는 Radix-2^2 알고리즘의 기반을 둔 R2^{2}SDF구조로 두개의 스테이지 후에 프로그래머블 복소 곱셈기가 위치해 있으며, 회전인자 저장을 위한 메모리를 가지고 있다. 그림 5(b)는 R2^{3}SDF 구조로 총 두개의 프로그래머블 복소 곱셈기와 두개의 상수 곱

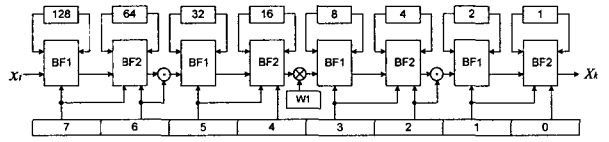
셈기를 사용한다. 이전의 연구에서 상수 복소 곱셈기의 크기는 프로그래머블 복소 곱셈기의 크기에 비하여 약 0.4배로 줄일 수 있었다.[2][5] 이것을 감안하면 R2^{3}SDF가 R2^{2}SDF보다 약간의 면적효율은 있겠으나, 전체적으로 곱셈이 네 번 수행되기 때문에 양자화 오류가 급증하여 정밀도 관점에서 성능이 저하된다. 이에 반하여 그림 5(c)는 제안된 R2^{4}SDF 파이프라인 FFT 구조이다. 두 개의 프로그래머블 복소곱셈기 대신 두 개의 제안된 CSD 상수 복소 곱셈기가 사용된 것을 제외하면, 이 구조의 데이터 흐름은 R2^{2}SDF 구조와 유사하다. 그러나 전체 복소 곱셈기의 면적을 40%가량 줄일 수 있다. 표 4는 여러 종류의 FFT 길이에 따른 R2^{2}SDF 구조와 R2^{4}SDF 구조간의 복소 곱셈기의 복잡도를 비교한 것이다. R2^{4}SDF의 곱셈 복잡도는 R2^{2}SDF 구조보다 30% 이상 감소함을 알 수 있다. 복소 곱셈기의 비교를 위해 프로그래머블 복소 곱셈기의 곱셈량을 1로 간주하고 제안한 CSD 상수 복소 곱셈기 0.4로 곱셈량을 계산하였다.



(a) Radix-2^2 SDF 구조



(b) Radix-2^3 SDF 구조



(c) 제안된 Radix-2^4 SDF 구조

그림 5. Radix-2^n 256-point 파이프라인 SDF FFT 구조
Fig. 5. Radix-2^n 256-point Pipeline SDF FFT Structures.

표 4. FFT 길이에 따른 곱셈기 복잡도 비교
Table 4. Multiplier complexity comparison as FFT length.

	곱셈 복잡도	N=64	128	512	2^{10}	2^{12}
R2^{2}SDF	$\log_4 N-1$	2	3	4	5	6
R2^{4}SDF	$<0.7\log_4 N-1$	1.4	1.8	2.8	3.2	4.2
개선정도(%)		**30**	**40**	**30**	**36**	**30**

IV. 실 험

비교를 위해, 실수부와 허수부 각각 12비트 워드길이를 갖는 두 종류의 복소 곱셈기를 설계하였다. 하나는 프로그래머블 복소 곱셈기의 예제로 Modified Booth 알고리즘 기반 복소 곱셈기이고[8], 다른 하나는 본 논문에서 제안된 CSD 상수 복소 곱셈기이다. FFT 설계에서는 출력결과 비트수의 증가를 막기 위해 절사(Truncation)를 통해 비트수를 고정시키고, 최종 덧셈 연산시 결과의 오버플로우를 피하기 위해 한 비트를 증가시키는 방식을 취한다.[4] 본 논문에서 설계한 곱셈기들은 이렇게 비트수를 고정했을 때 발생하는 양자화 오류를 최소화하는 구조로서 기존의 곱셈기보다 50%에 가까운 면적효율을 갖는 저 오차 고정길이 곱셈기로 설계하였다.[7][8] 프로그래머블 복소 곱셈기는 회전계수 저장공간을 위한 ROM을 필요로 한다. 따라서 N/2개의 회전계수를 저장하는 공간을 위한 메모리가 추가적으로 늘어나게 된다. 그러나 제안한 CSD 상수 복소 곱셈기는 회전계수를 위한 저장 공간을 필요로 하지 않는 장점이 있다.

구현된 CSD 고정길이 상수 복소 곱셈기는 입력신호(X)가 입력되었을 시 다음의 계수에 따른다. m_0과 m_1는 표 2에서 설명한 계수이다.

$$P_0 = X \times m_0 = x_{11}.x_{10}x_9x_8x_7x_6x_5x_4x_3x_2x_1x_0 \times 1000\bar{1}0\bar{1}00100 \quad (17)$$

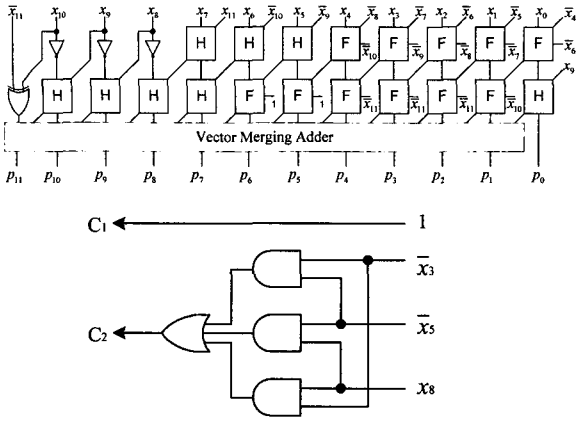
$$P_1 = X \times m_1 = x_{11}.x_{10}x_9x_8x_7x_6x_5x_4x_3x_2x_1x_0 \times 010\bar{1}0001000\bar{1} \quad (18)$$

P_0과 P_1의 고정길이 연산 후 양자화 오류를 최소화하기 위해 에러 벡터를 더하여 이를 보상한다. 에러보상 바이어스 회로(σ_P)는 각각 수식 (19)과 수식 (20)와 같다.[7]

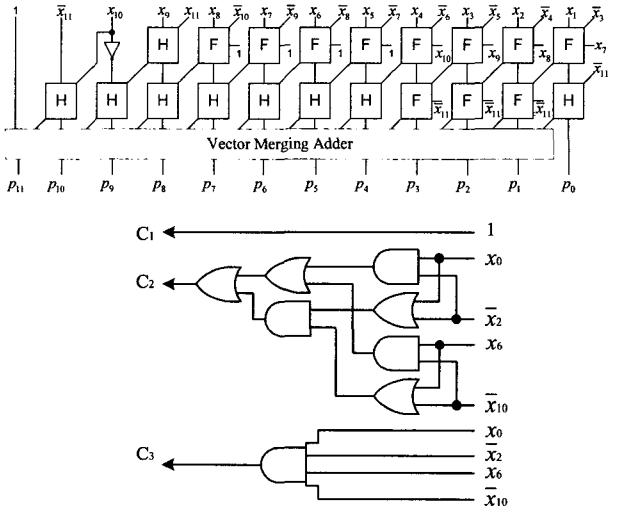
$$\sigma_{P_0} = 1 + C_1$$
$$C_1 = \bar{x}_3 x_8 + \bar{x}_5 x_8 + \bar{x}_3 \bar{x}_5 \quad (19)$$

$$\sigma_{P_1} = 1 + C_1 + C_2$$
$$C_1 = x_6\bar{x}_{10} + x_0\bar{x}_2 + (\bar{x}_{10} + x_6)(x_0 + \bar{x}_2)$$
$$C_2 = x_0\bar{x}_2x_6\bar{x}_{10} \quad (20)$$

구현된 CSD 상수 복소 곱셈기와 에러보상 바이어스 회로는 그림 6에 보인다.



(a) 최적화된 P_0와 σ_{P_0}의 구조 (F:전가산기, H:반가산기)



(b) 최적화된 P_1와 σ_{P_1}의 구조 (F:전가산기, H:반가산기)

그림 6. 제안된 CSD 고정길이 상수 곱셈기와 에러 보상 바이어스

Fig. 6. The proposed CSD fixed-width constant multiplier and error compensation circuits.

표 5. 프로그래머블 복소 곱셈기와 제안된 곱셈기 비교

Table 5. Comparison with the programmable multiplier.

형식	MB. 복소 곱셈기 (real multiplier × 4)		제안된 CSD 상수 복소 곱셈기
ROM	0 bits	768 bits	0 bits
효율	2820	5042	1352

삼성 0.35um CMOS 표준 공정을 통해 합성 시뮬레이션을 하고 서로 비교하였다. 회전인자를 위한 ROM을 가지고 있지 않은 프로그래머블 복소 곱셈기의 게이트 개수는 2820이고, ROM을 가진 프로그래머블 복소 곱셈기는 5042였다. 반면에 제안된 CSD 상수 복소 곱셈기는 1352로 합성되었다. 이 결과는 표 5와 같이 면적과 전력 면에서 52%에서 73%의 효율을 나타낸다.

V. 결 론

본 논문은 새로운 Radix-2^4 알고리즘을 제안하고 이를 기반으로 하는 파이프라인 구조인 R2^4SDF를 제안하였다. R2^4SDF 구조는 기존의 R2^2SDF구조의 프로그래머블 복소 곱셈기의 절반을 새롭게 제안하는 CSD 상수 복소 곱셈기로 대체할 수 있었다. 제안된 CSD 상수 복소 곱셈기는 전력소모와 면적 관점에서 기존의 프로그래머블 복소 곱셈기보다 60%이상 효율이 있었다. 따라서 R2^2SDF 구조와 비교한다면, 새로운 R2^4SDF FFT 구조는 곱셈기의 절반을 제안한 CSD 상수 복소 곱셈기로 대체하여 전체 복소 곱셈기의 복잡도를 30%이상 줄여 면적과 전력 면에서 효율을 얻을 수 있었다. 제안된 FFT 구조는 고성능이며, 전력과 면적 면에서 효율성을 요구하는 OFDM 무선응용 분야의 큰 포인트 크기를 갖는 FFT 프로세서 설계에 유용할 것이다.

참 고 문 헌

[1] 조병각,손병수, 선우명훈 "OFDM 시스템을 위한 고속 FFT 프로세서," 전자공학회논문지, 제39권 TC편, 제12호, 513-519, 2002년 12월

[2] W. C. Yey, C. W. Jen, "High-Speed and Low-Power Split-Radix FFT," IEEE Trans. Sig. Proc., Vol. 51, No. 3, pp. 864-674, 2003.

[3] S. He and M. Torkelson, "Designing pipeline FFT processor for OFDM (de)Modulation," in Proc. IEEE URSI Int. Symp. Signals, Syst., Electron., pp. 257-262, 1998.

[4] J. Melander, "Design of SIC FFT Architectures, Linköping Studies in Science and Technology," Thesis No. 618, Linköping University, Sweden, 1997.

[5] J. Y. Oh, J. S. Cha, S. K Kim, M. S. Lim, "Imple-mentation of Orthogonal Frequency Division Multiplexing using radix-N Pipeline Fast Fourier Transform (FFT) Processor," Jpn. J. Appl. Phys. vol. 42, part 1, No. 4B, pp. 1-6, 2003.

[6] K. K. Parhi, *VLSI digital signal processing systems*, John Wiley & Sons, Inc., USA, pp. 505-511, 1999.

[7] S. M. Kim, J. G. Chung, K. K. Parhi, "Design of low error CSD fixed-width multiplier," IEEE Int. Symp. Cir., Syst., Vol.1, pp. I-69 - I-72, 2002.

[8] K. J. Cho, K. C. Lee, J. G. Chung, K. K. Parhi, "Design of Low-Error Fixed-Width Modified Booth Multiplier," IEEE Trans. VLSI Syst., Vol. 12, No. 5, 2004.

저 자 소 개

오 정 열(정회원)

1997년 동신대학교 정보통신 공학과 학사 졸업.
1999년 전북대학교 정보통신 공학과 석사 졸업
2001년 전북대학교 컴퓨터공학과 박사 수료.

현재 (주)에이트리 선임연구원

<주관심분야 : 휴대인터넷, OFDM, 신호처리, FFT 설계>

임 명 섭(정회원)

1980년 연세대학교 전자공학과 학사 졸업.
1982년 연세대학교 전자공학과 석사 졸업.
1990년 연세대학교 전자공학과 박사 졸업.

1984년 1월~1985년 9월 대우통신 종합연구소 연구원.
1985년 9월~1996년 10월 한국전자통신연구소 책임 연구원.
1996년 10월~현재 전북대학교 전자정보공학부 부교수

<주관심분야 : OFDM, UWB, 차세대 이동통신, 신호처리, 차량용 정보전자기기 >