논문 2009-46SP-4-20

Common sub-expression sharing과 CORDIC을 이용한 OFDM 시스템의 저면적 파이프라인 FFT 구조

(Low-area Pipeline FFT Structure in OFDM System Using Common Sub-expression Sharing and CORDIC)

최 동 규**, 장 영 범*

(Dong Kyu Choi and Young Beom Jang)

요 약

이 논문에서는 OFDM시스템에서 가장 큰 칩 면적을 차지하고 높은 전력을 요구하는 핵심 연산 블록인 FFT에 대하여 파이프라인 Radix-4 MDC 방식의 저면적 구조를 제안하였다. 나비연산기에서 Twiddle factor 복소 곱셈연산을 수행할 때, 기존의 곱셈기를 사용하지 않고 CSD형 계수의 공통패턴을 공유하여 덧셈의 수를 줄일 수 있는 Common sub-expression sharing 방식과 CORDIC 알고리즘을 사용하여 구현 면적을 감소시켰다. 제안구조는 Verilog-HDL을 통해 모델링하고 Synopsys로 논리합성한 결과 기존구조와 비교하여 복소곱셈부는 48.2%감소효과, 전체 FFT구조는 22.1%의 면적 감소효과를 달성하였다. 따라서 제안된 FFT구조는 다양한 크기의 FFT를 사용하는 OFDM용 시스템에 효율적으로 사용될 수 있는 구조임을 보였다.

Abstract

An efficient pipeline MDC Radix-4 FFT structure is proposed in this paper. Every stages in pipeline FFT structure consists of delay commutator and butterfly. Proposed butterflies in front and rear stages utilize CORDIC and Common Sub-expression Sharing(CSS) techniques, respectively. It is shown that proposed butterfly structure can reduce the number of adders through sharing common patterns of CSD type coefficients. The Verilog-HDL modeling and Synopsys logic synthesis results that the proposed structure show 48.2% cell area reduction in the complex multiplication part and 22.1% cell area reduction in overall 256-point FFT structure comparison with those of the conventional structures. Consequently, the proposed FFT structure can be efficiently used in various OFDM systems.

Keywords: OFDM, FFT, CORDIC, Common sub-expression sharing, twiddle factor

I. 서 론

OFDM(Orthogonal Frequency Division Multiplexing) 방식은 유·무선채널에서 고속 데이터 전송에 적합한 방식으로 IEEE 802.11a, 802.16a/d, DAB/DMB, DVB-T, UWB(Utra Wideband)의 표준 방식으로 채택되어 폭넓게 이용되고 있으며, IEEE 802.15.3a에서 고속 전송방식의 표준으로 MB-OFDM이 고려되는 등 최근 활발히 연구되고 있다. OFDM 전송방식의 기본 개념은 직렬로 입력되는 데이터열을 N개의 병렬 데이터열로 변환하여 각각 분리된 부반송파에 실어 전송함으로써 데이터율을 높인다. 따라서 하나의 반송파를 사용하여 데이터를 순차적으로 전송하는 경우보다 전송되는 심볼의 간격이 길어져 채널의 지연시간 영향과 임펄스 잡음의 영향을 덜 받게 된다. 또한 연속된 심볼간의 간섭을 줄일 수 있어 다중 경로 채널에 대해 강하며 채널 등화의 복잡도를 줄일 수 있고 일반적인 주파수 분할방식에

* 정회원, 상명대학교 공과대학 정보통신공학과
(College of Engineering, Sangmyung University)
** 학생회원, 상명대학교 컴퓨터정보통신공학과
(Graduate School, Sangmyung University)
※ 본 연구는 교육과학기술부와 한국산업기술진흥원의 지역혁신인력양성사업으로 수행된 연구 결과임.
접수일자:2008년12월17일, 수정완료일:2009년6월10일.

비하여 스펙트럼의 효율을 높일 수 있다.[1] OFDM기반 변복조기는 응용 시스템에 따라 64부터 8192-point의 다양한 범위의 IFFT/FFT 프로세서가 사용되며, 이는 OFDM시스템에서 가장 큰 칩 면적을 차지하고 높은 전력소모를 요구하는 핵심 연산 블록이다. 가장 많이 사용되는 파이프라인 Radix-4 FFT 구조는 여러 스테이지로 구성되며 각각의 스테이지는 지연변환기(Delay commutator), 곱셈연산이 사용되는 나비연산기(Butterfly)로 구성된다. Radix-4 나비연산기의 효과적인 구조에 대하여 많은 연구가 진행되었다.[2~6] [2]에서는 DA(Distributed Arithmetic) 방식을 사용하여 기존의 곱셈기를 대체하였으나 하나의 곱셈을 위하여 많은 클록이 요구되는 단점을 보인다. [3~6]에서는 CORDIC 방식을 이용하여 기존 곱셈기를 대체하였다. 이 방식은 앞단의 스테이지에서는 효과적이나 곱셈계수가의 종류가 적은 뒷단의 스테이지에서는 비효율적이다. 이 논문에서는 나비연산기의 저면적 구현을 위하여 앞단의 스테이지에서는 CORDIC 알고리즘을 사용하고 뒷단의 스테이지에서는 Common subexpression sharing(CSS) 기술을 사용하는 효율적인 구조를 제안한다. CSS 기술은 원래 디지털 필터의 곱셈연산을 덧셈기를 사용하여 효과적으로 구현하는 기술이다.[7~9] Ⅱ장에서는 전반적인 OFDM시스템에서의 FFT에 대해 살펴보고 Ⅲ장에서는 CORIDC알고리즘을 이용한 기존의 FFT구조에 대하여 설명한다. Ⅳ장에서는 CORDIC과 CSS방식을 적용한 하이브리드구조를 제안하고 Ⅴ장에서는 제안구조의 시뮬레이션 및 합성결과를 통해 효율성을 입증하고, Ⅵ장에서 결론을 맺는다.

Ⅱ. OFDM시스템에서의 FFT

OFDM시스템에서 변복조는 입력데이터를 부반송파의 수만큼 직병렬변환하고 각각에 대응되는 부반송파로 변조하는데 이는 다음의 DFT에 의해 구현될 수 있다.

$$X(k)=\sum_{n=0}^{N-1}x[n]\,W_N^{nk},\ W_N=e^{-j\frac{2\pi}{N}} \tag{1}$$

부반송파의 수가 많아지면 연산량이 많아지고 그에 따른 하드웨어 설계의 부담이 커지기 때문에 이를 극복하기 위해 FFT알고리즘을 이용하여 구현한다. FFT는 복소연산을 수행하므로 실제 설계상으로는 실수부와 허수부가 나뉘어져 연산된다. FFT를 구현하는 방법으로

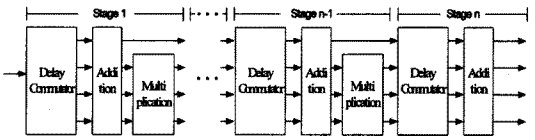


그림 1. *N*-point 파이프라인 FFT 구조
Fig. 1. *N*-point pipeline FFT structure.

는 크게 파이프라인 방식과 메모리기반의 방식으로 나눌 수 있는데, 이 논문은 MDC(multi-path Delay commutator)를 이용한 파이프라인 방식의 Radix-4 DIF 알고리즘을 기반으로 한다. 파이프라인 방식은 기본적으로 그림 1과 같이 각각의 스테이지가 지연변환기(Delay commutator)와 나비연산기(Butterfly)로 구성되며 나비연산기는 다시 Addition 블록과 Multiplication 블록으로 나누어진다.

$$\begin{aligned}
X(4r)&=\sum_{n=0}^{N/4-1}x_0(n)\,W_{N/4}^{nr}\\
X(4r+1)&=\sum_{n=0}^{N/4-1}x_1(n)\,W_{N/4}^{nr}\\
X(4r+2)&=\sum_{n=0}^{N/4-1}x_2(n)\,W_{N/4}^{nr}\\
X(4r+3)&=\sum_{n=0}^{N/4-1}x_3(n)\,W_{N/4}^{nr}
\end{aligned} \tag{2}$$

그림 1은 Radix-4 버터플라이 연산을 수행하는 전체 FFT구조를 나타낸다. Radix-4를 기반으로 하므로 256-point FFT의 경우에 $256=4^4$이므로 그림 1은 4개의 스테이지로 구성된다. Radix-4를 사용하므로 네 개의 패스로 연결되어 있다. 버터플라이 안에는 twiddle factor를 곱하기 위한 복소곱셈기가 포함되고, 각 스테

$$\begin{aligned}
x_0(n)&=x(n)+X(n+\tfrac{N}{4})+x(n+\tfrac{N}{2})\\
&+x(n+\tfrac{3N}{4})\\
x_1(n)&=[x(n)+(-j)x(n+\tfrac{N}{4})\\
&+(-1)x(n+\tfrac{N}{2})+(j)x(n+\tfrac{3N}{4})]\,W_N^n\\
x_2(n)&=[x(n)+(-1)x(n+\tfrac{N}{4})\\
&+(1)x(n+\tfrac{N}{2})+(-1)x(n+\tfrac{3N}{4})]\,W_N^{2n}\\
x_3(n)&=[x(n)+(j)x(n+\tfrac{N}{4})\\
&+(-1)x(n+\tfrac{N}{2})+(-j)x(n+\tfrac{3N}{4})]\,W_N^n
\end{aligned} \tag{3}$$

이지의 나비연산기는 twiddle factor만 다를 뿐, 모두 동일한 연산을 수행하게 되는 규칙성을 갖는다. 또한 마지막스테이지의 twiddle factor는 항상 1이므로 복소곱셈이 필요 없다. 나비연산기의 설계를 위하여 식(1)을 4개의 $N/4$-point로 나누어 나타내면 식(2)와 같다.

N-point의 식(1)이 식(2)와 같이 $N/4$-point로 분해되기 위해서는 식(2)의 $x_0(n), x_1(n), x_2(n), x_3(n)$의 연산이 선행되어야 하며 다음 식(3)과 같다. 식(3)을 연산하는 블록을 나비연산기라고 부른다.

식(3)의 나비연산기 연산은 실제 실수부와 허수부의 복소연산으로 구현되므로 입력, 출력, twiddle factor를 다음과 같이 복소수로 정의하여야 편리하다.

$$\begin{aligned} x(n) &= x_a + jy_a, & x_0(n) &= X_a + jY_a \\ x(n+N/4) &= x_b + jy_b, & x_1(n) &= X_b + jY_b \\ x(n+N/2) &= x_c + jy_c, & x_2(n) &= X_c + jY_c \\ x(n+3N/4) &= x_d + jy_d, & x_3(n) &= X_d + jY_d \end{aligned} \tag{4a}$$

$$\begin{aligned} W_N^n &= e^{-j\frac{2\pi n}{N}} = \cos\theta_b - j\sin\theta_b \\ W_N^{2n} &= e^{-j\frac{2\pi 2n}{N}} = \cos\theta_c - j\sin\theta_c \\ W_N^{3n} &= e^{-j\frac{2\pi 3n}{N}} = \cos\theta_d - j\sin\theta_d \end{aligned} \tag{4b}$$

이와 같이 정의된 복소수를 사용하여 나비연산기의 출력을 다음과 같은 곱셈연산으로 나타낼 수 있다.

$$\begin{aligned} X_b &= x_1\cos\theta_b + x_2\sin\theta_b \\ Y_b &= x_2\cos\theta_b - x_1\sin\theta_b \end{aligned} \tag{5a}$$

$$\begin{aligned} X_c &= x_3\cos\theta_c + x_4\sin\theta_c \\ Y_c &= x_4\cos\theta_c - x_3\sin\theta_c \end{aligned} \tag{5b}$$

$$\begin{aligned} X_d &= x_5\cos\theta_d + x_6\sin\theta_d \\ Y_d &= x_6\cos\theta_d - x_5\cos\theta_d \end{aligned} \tag{5c}$$

식(5)에서 필요한 값들과 X_a와 Y_a는 은 다음과 같은 덧셈연산으로 계산될 수 있다.

$$\begin{aligned} X_a &= x_a + x_b + x_c + x_d \\ Y_a &= y_a + y_b + y_c + y_d \end{aligned} \tag{6a}$$

$$\begin{aligned} x_1 &= x_a + y_b - x_c - y_d \\ x_2 &= y_a - x_b - y_c + x_d \end{aligned} \tag{6b}$$

$$\begin{aligned} x_3 &= x_a - x_b + x_c - x_d \\ x_4 &= y_a - y_b + y_c - y_d \end{aligned} \tag{6c}$$

$$\begin{aligned} x_5 &= x_a - y_b - x_c - y_d \\ x_6 &= y_a + x_b - y_c - x_d \end{aligned} \tag{6d}$$

위의 식에서 보듯이 나비연산은 식(6)의 덧셈연산과 식(5)의 곱셈연산으로 구성된다. 다음 장에서는 곱셈연산의 저면적 구현을 위한 방식을 제안한다.

III. 제안된 나비연산기 구조

1. CORDIC 스테이지 설계

파이프라인 방식의 나비연산기는 앞단의 스테이지일수록 많은 종류의 twiddle factor가 사용된다. twiddle factor는 cosine과 sine 파형의 한주기를 샘플링하여 얻어지는데, point 수가 커질수록 이를 저장하기 위한 ROM의 크기가 매우 커지게 되고, 복소곱셈의 수도 많아지게 되어 전력소모와 면적이 증가한다. 이를 해결하기 위하여 나비연산기의 복소 곱셈연산을 수행하는데 있어, 구현 시 면적을 많이 차지하는 일반곱셈기 대신 CORIDC알고리즘을 이용한 연구가 진행되었다. 이 논문에서는 twiddle factor가 많이 사용되는 앞단의 스테이지에서는 CORDIC 알고리즘을 사용하는 방식을 사용하였다. CORDIC 알고리즘은 간단한 하드웨어로 구현할 수 있는 장점이 있어 통신시스템의 다양한 블록에 이용된다. 나비연산기 곱셈연산은 다음과 같은 행렬의 식으로 나타낼 수 있다.

$$\begin{bmatrix} X \\ Y \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{7}$$

위의 식과 같은 곱셈연산은 그림 2와 같은 CORDIC 구조로 구현할 수 있다.

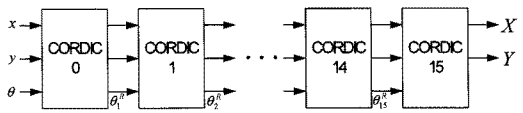


그림 2. CORDIC연산의 블록도
Fig. 2. CORDIC operation block diagram.

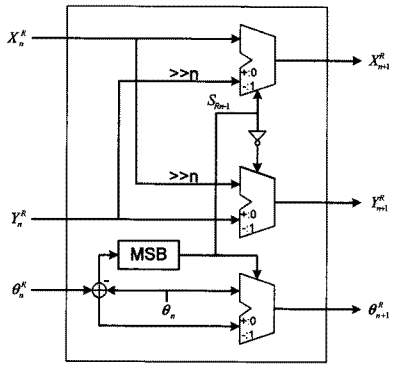


그림 3. CORDIC블록 Cn의 세부구조
Fig. 3. Detail structure of Cn in CORDIC block.

그림 2는 CORDIC의 Rotation 모드를 이용하여 곱셈 연산부를 수행하도록 설계한 블록도이다. 이 논문에서는 결과값을 1 클록에 출력시키기 위해 병렬 CORDIC을 사용하였다. 즉 나비연산기 덧셈부에서 출력되는 출력 값으로 x, y를 초기화하고 θ만큼 16번 반복 회전하여 cosine과 sine값이 곱해진 X와 Y 값을 계산한다. 그림 2의 Cn세부구조는 그림 3과 같다.

x, y 값과 θ를 저장하는 3개의 레지스터와 각각의 덧셈이나 뺄셈을 수행하는 3개의 ALU, 그리고 2개의 쉬프터로 구성된다. 각 입력값들의 가산 혹은 감산연산은 입력받은 θ에 회전된 각도를 빼주어 MSB를 테스트하여 양수이면 가산연산, 음수이면 감산연산을 수행하도록 ALU(Arithmetic Logic Unit)를 제어하도록 설계하였다.

2. CSS 스테이지 설계

기존의 CORDIC을 이용한 파이프라인 FFT구조는 모든 스테이지에 같은 CORDIC 곱셈기를 사용하였다. 그러나 마지막 스테이지에 가까워질수록 사용되는 twiddle factor 계수의 종류는 줄어들게 되어 CORDIC으로 구현하는 것이 비효율적이다. 따라서 이 논문에서는 계수의 종류가 적은 후반부 스테이지 구현에 Common Sub-expression Sharing(CSS) 기술을 사용한다. 원래 CSS 기술은 디지털 필터의 구현에 효과적인 방식이다. CSS 기술을 적용하기 위하여 먼저 사용되는 상수 계수들을 CSD(Canonic Signed Digit) 형으로 나타내어야 한다.

곱셈연산을 덧셈과 쉬프트를 이용하여 구현하는 경우 쉬프트 연산은 하드웨어 구현 비용이 상대적으로 작으므로 덧셈의 수가 비용을 좌우한다. 계수들의 2진수 표현에서는 1과 0의 갯수 중에서 1의 수가 곧 덧셈의 수가 되므로 1의 빈도가 구현 비용을 좌우한다. 따라서 2'complement 형보다 1의 수가 적은 CSD 형의 계수를 사용하면 구현 비용을 줄일 수 있다. 즉, CSD 형의 계수는 (N+1)/2 이상의 non-zero 비트를 갖지 않는 장점이 있으므로 CSD형 계수를 사용하는 것이 효율적이다. 더 나아가 디지털 필터구조에서 사용되는 CSS 방식을 사용하면 덧셈의 수를 더욱 감소시킬 수 있다. 이 절에서는 CSS방식을 twiddle factor의 곱셈연산에 사용하고 sine과 cosine의 대칭성을 이용하여 twiddle factor를 N/4-1만큼만 이용하게 함으로써 구현면적을 더욱 감소시킬 수 있는 효율적인 구조를 제안한다.

제안구조의 설명을 쉽게 하기 위하여 256-point FFT를 예로 들어 설명하기로 한다. 즉, Radix-4를 사용하였으므로 4개의 스테이지로 구성되며 첫 번째 스테이지는 CORDIC 방식을 사용하고 나머지 3개의 스테이지는 CSS 기술을 사용하여 구현하기로 한다. 첫 번째 스테이지에서의 twiddle factor 수는 매우 많기 때문에 CSS형으로 구현하는 것은 비효율적이다. 따라서 제안된 256-point FFT 구조의 첫 번째 스테이지는 CORDIC 알고리즘을 적용하고 두 번째, 세 번째 스테이지에서는 CSS방식을 적용하여 구현한다. 네 번째 스테이지는 곱셈연산이 사용되지 않으므로 제외된다.

먼저 256-point FFT의 두 번째 스테이지에서 사용되는 twiddle factor는 64개가 필요하다. 그러나 twiddle factor는 주기함수이므로 모든 계수를 다 계산할 필요가 없이 절대값이 독립적인 N/4-1만큼의 개수 즉, 15개의 계수만 이용하여 계산한다.

다음 그림 4의 회색부분이 twiddle factor가 이용되는 영역이다. 회색영역을 8로 분할한 값들의 실수부, 허수부로 나타난 16개의 값 중에서 45°되는 부분의 값은

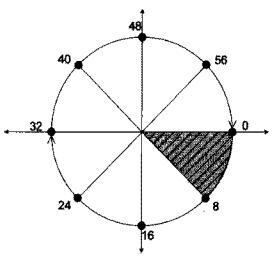


그림 4. 64-point FFT에 사용되는 twiddle factor 영역
Fig. 4. Twiddle factors domain of 64-point FFT.

표 1. Twiddle factor에 대한 CSD형 계수
Table 1. CSD type coefficients of twiddle factors.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0.09801	0	0	0	1	0	N	0	0	1	0	0	1	0	N	0	N
0.19509	0	0	1	0	N	0	0	1	0	0	0	0	N	0	0	0
0.29028	0	0	1	0	0	1	0	1	0	0	1	0	1	0	0	0
0.38268	0	1	0	N	0	0	0	1	0	0	0	0	0	N	0	N
0.47139	0	1	0	0	0	N	0	0	1	0	1	0	1	0	N	0
0.55557	0	1	0	0	1	0	0	N	0	0	1	0	0	N	0	0
0.63439	0	1	0	1	0	0	0	1	0	1	0	N	0	1	0	N
0.70710	1	0	N	0	N	0	1	0	1	0	0	0	0	0	1	0
0.77301	1	0	N	0	0	1	0	N	0	0	0	N	0	0	0	1
0.83147	1	0	N	0	1	0	1	0	1	0	0	N	0	N	0	1
0.88192	1	0	0	N	0	0	0	1	0	0	N	0	0	0	1	0
0.92388	1	0	0	0	N	0	N	0	0	1	0	0	0	0	0	1
0.95694	1	0	0	0	N	0	1	0	1	0	0	0	0	N	0	1
0.98078	1	0	0	0	0	0	N	0	N	0	0	0	1	0	1	0
0.99518	1	0	0	0	0	0	0	0	N	0	N	0	0	0	1	0

표 2. Twiddle factor에 대한 CSD형 계수와 CSS (Common sub-expression) 표현
Table 2. CSD type coefficients and common sub-expression of twiddle factors.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0.098017	0	0	0	1	0	N	0	0	1	0	0	1	0	N	0	N
0.19509	0	0	1	0	N	0	0	1	0	0	0	0	N	0	0	0
0.290285	0	0	1	0	0	1	0	1	0	0	1	0	1	0	0	0
0.382683	0	1	0	N	0	0	0	1	0	0	0	0	0	N	0	N
0.471397	0	1	0	0	0	N	0	0	1	0	1	0	1	0	N	0
0.55557	0	1	0	0	1	0	0	N	0	0	1	0	0	N	0	0
0.634393	0	1	0	1	0	0	0	1	0	1	0	N	0	1	0	N
0.707107	1	0	N	0	N	0	1	0	1	0	0	0	0	0	1	0
0.77301	1	0	N	0	0	1	0	N	0	0	0	N	0	0	0	1
0.83147	1	0	N	0	1	0	1	0	1	0	0	N	0	N	0	1
0.881921	1	0	0	N	0	0	0	1	0	0	N	0	0	0	1	0
0.92388	1	0	0	0	N	0	N	0	0	1	0	0	0	0	0	1
0.95694	1	0	0	0	N	0	1	0	1	0	0	0	0	N	0	1
0.980785	1	0	0	0	0	0	N	0	N	0	0	0	1	0	1	0
0.995185	1	0	0	0	0	0	0	0	N	0	N	0	0	0	1	0

(0.707,0.707)로 같으므로 15개만을 이용한다. 그림 4에서 나타낸 영역의 15개의 twiddle factor를 16비트 정세도의 CSD형으로 나타내면 표 1과 같다.

표 1에서 N은 -1을 나타낸다. 표 1의 0.09801의 곱셈연산에는 6개의 non zero 비트가 있으므로 5개의 덧셈기가 필요하게 된다. 따라서 각각의 계수 구현에 필요한 총 덧셈의 수는 68개가 필요하다. 이는 2's complement 형으로 구현했을 때의 116개보다 CSD 형 구현이 46개의 덧셈이 감소됨을 보였다. CSD 형의 덧셈연산을 더욱 줄이기 위하여 CSS 기술을 다음과 같이 적용한다.

CSS 기술은 표 1에서 공통패턴을 정의하여 정의된 공통패턴을 서로 공유하는 기술이다. 이렇게 공통패턴을 공유함으로써 덧셈의 수를 더욱 감소시킬 수 있다. 표 1에서 관찰되는 공통패턴을 묶음으로 표현하면 표 2와 같다. 표 2에서 보듯이 10N의 패턴이 여러 개 사용되고 있으므로 이 패턴을 공통패턴으로 정의한다.

표 2에서와 같이 공통패턴들을 2중 실선으로 표시하였다. 표 2에서 10N, 101, 1001, 100N의 4개의 공통패턴이 있음을 알 수 있다. 여기서 N001과 N0N의 패턴은 100N과 101의 부호만 바꾸면 같은 패턴이 되므로 공통패턴으로 정의할 필요가 없다.

이와 같은 공통패턴을 식으로 나타내면 다음과 같다.

$$\begin{aligned} x_2 &= x_1 + x_1 \gg 2 \\ x_3 &= x_1 - x_1 \gg 2 \\ x_4 &= x_1 + x_1 \gg 3 \\ x_5 &= x_1 - x_1 \gg 3 \end{aligned} \tag{8}$$

위와 같이 정의된 4개의 공통패턴을 이용하여 t_1부터 t_{15}의 15개의 twiddle factor를 식으로 나타내면 식 (9)와 같다.

$$\begin{aligned} t_1 &= x_3 \gg 3 + x_4 \gg 8 - x_2 \gg 13 \\ t_2 &= x_3 \gg 2 + x_1 \gg 7 - x_1 \gg 12 \\ t_3 &= x_4 \gg 2 + x_4 \gg 7 + x_1 \gg 12 \\ t_4 &= x_3 \gg 1 + x_1 \gg 7 + x_2 \gg 13 \\ t_5 &= x_1 \gg 1 + x_5 \gg 5 + x_1 \gg 10 + x_3 \gg 12 \\ t_6 &= x_1 \gg 1 + x_5 \gg 4 + x_5 \gg 10 \\ t_7 &= x_2 \gg 1 + x_1 \gg 7 + x_3 \gg 9 + x_3 \gg 13 \\ t_8 &= x_3 \gg 1 - x_3 \gg 4 + x_1 \gg 8 + x_1 \gg 14 \\ t_9 &= x_3 + x_3 \gg 5 - x_1 \gg 11 + x_1 \gg 15 \\ t_{10} &= x_3 + x_2 \gg 4 + x_5 \gg 8 - x_3 \gg 13 \\ t_{11} &= x_5 + x_5 \gg 7 + x_1 \gg 14 \\ t_{12} &= x_1 - x_2 \gg 4 + x_1 \gg 9 + x_1 \gg 15 \\ t_{13} &= x_1 - x_3 \gg 4 + x_1 \gg 8 - x_3 \gg 13 \\ t_{14} &= x_1 - x_2 \gg 6 + x_2 \gg 12 \\ t_{15} &= x_1 - x_2 \gg 8 + x_1 \gg 14 \end{aligned} \tag{9}$$

식 (9)와 같이 나타낸 15개의 twiddle factor를 덧셈기와 쉬프트를 사용하여 설계한 CSS구조는 다음 그림 5와 같다.

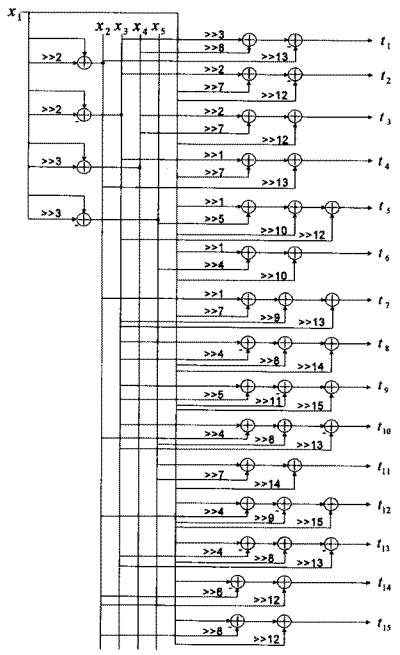


그림 5. 제안된 CSS방식을 사용한 twiddle factor 곱셈 구조

Fig. 5. Proposed twiddle factor multiplication structure using CSS method.

나비연산기의 덧셈부로부터 입력된 x_1은 그림 5의 왼쪽 상단부분에서 보듯이 공통패턴 x_2, x_3, x_4, x_5를 먼저 계산하고 4개의 공통패턴 및 초기입력 값을 이용하여 쉬프트와 덧셈, 뺄셈연산을 통해 각각의 15개의 출력 값들을 계산하도록 설계하였다. 하나의 입력 샘플에 대해 15개의 모든 twiddle factor가 곱해진 출력이 나오고 이것들 중에서 연산되는 순서에 맞도록 선택하여 계산되도록 설계하였다. 그림 5에서 볼 수 있듯이 공통패턴을 공유했으므로 68개를 이용했던 CSD형보다 적은 41개의 덧셈기만으로 twiddle factor의 연산부를 구현하였다. 이와 같이 나비연산기의 곱셈연산부를 CSS 방식을 사용하여 구현면적을 줄일 수 있음을 볼 수 있다.

이와 같이 twiddle factor의 수가 적은 뒷단의 스테이지에서는 CSS 방식이 효율적임을 알 수 있다. 파이프라인 FFT 구조에서는 앞단 스테이지일수록 사용되는 twiddle factor의 종류가 많아지게 되므로 CORDIC 방식을 사용하고, twiddle factor의 종류가 적은 뒷단 스테이지에는 CSS 방식을 적용하여 전체구조의 효율성을 최대화하였다.

Ⅳ. 실험 및 고찰

이 절에서는 제안된 구조의 효율성을 시뮬레이션하기 위하여 256-point FFT를 그림 6과 같이 파이프라인 Radix-4 MDC 방식으로 설계하여 시뮬레이션 하였다.

제안구조 1은 그림 6에서 보듯이 첫 번째 스테이지의 곱셈부는 CORDIC으로 구현하였고 두 번째와 세 번째의 곱셈부는 CSS 방식을 사용하였다.

제안구조 1의 효율성을 비교하기위하여 4개의 스테이지를 모두 CORDIC 방식으로 설계한 구조와 4개의 스테이지를 모두 CSS 방식으로 설계한 구조(제안구조 2)와 비교하였다. 검증을 위한 테스트벡터는 Matlab을 통해 랜덤한 256개의 샘플값을 사용하여 마지막 스테이지의16비트의 출력 샘플용 테스트 벡터를 추출하였다. 각각의 3개의 구조에 대하여 C 프로그램을 사용하여 스테이지 별 샘플 값을 추출하였다. 즉, C 프로그램으로 각각의 구조의 function 검증을 포함한 스테이지 별 테스트 벡터를 추출하였다. 설계된 3개의 FFT 구조는 모두 Verilog-HDL로 모델링하고 Function 시뮬레이션

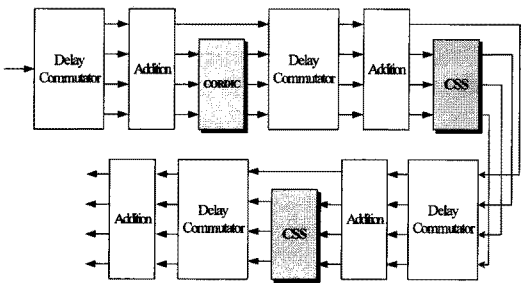


그림 6. 제안된 Radix-4 256-point FFT블록도

Fig. 6. Proposed Radix-4 256-point FFT block diagram.

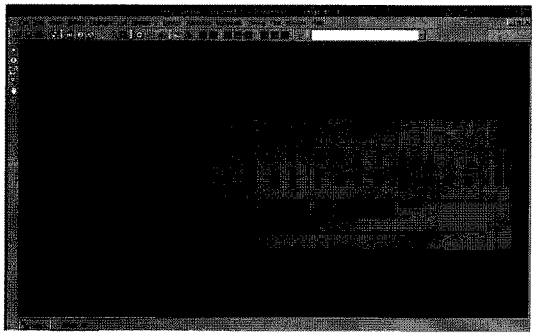


그림 7. Chartered 0.18um 논리합성 결과

Fig. 7. Chartered 0.18um logic synthesis result.

표 3. 각 구조별 복소 곱셈블록의 cell area 비교
Table 3. Cell area comparison of complex multiplier block for each structure.

		CORDIC	제안구조2 (CSS)	제안구조1 (CORDIC +CSS)
Cell Area (μm^2)	1 stage	556443.507	721983.659	556443.507
	2 stage	488165.818	199101.708	199101.708
	3 stage	472561.675	31227.462	31227.462
	Total	1517171	952312.829	786772.677
면적비율		100%	62.7%	51.8%

을 위해 Modelsim을 이용하였다. Chartered 0.18um CMOS 셀 라이브러리를 이용하여 3개의 구조에 대하여 합성하였으며 각 구조 마다 동일한 constraints를 적용하여 진행하였다. Pre-simulation을 위해 Design compiler를 통해 합성한 netlist와 sdc를 Primetime과 Formality의 입력으로 이용하여 timing과 function을 검증하였으며 논리합성결과는 다음 그림 7과 같다.

표 3은 각 구조에 대한 복소 곱셈블록의 cell area 결과를 보여준다.

표 3에서 보듯이 CORDIC 방식만을 이용한 경우는 면적이 1,517,171 μm^2로 합성되었으며, CSS 방식만을 적용한 경우는 952,312 μm^2로 합성되었다. 이 구조들과 비교하여 제안된 CORDIC+CSS 방식은 구현면적이 786,772 μm^2로 합성되었다. 제안된 방식이 CORDIC 방식에 비하여 48.2%의 면적 감소효과를 볼 수 있다.

이제 곱셈블록뿐 아니라 덧셈부를 포함한 나비연산기와 지연변환기를 포함한 전체 FFT 구조에 대한 합성결과를 비교해보기로 한다. 이와 같이 합성한 각 방식별 전체 FFT구조의 합성결과는 표 4와 같다. 표 4에서 나타난 합성결과에서 보듯이 제안구조 1이 기존 구조와 비교하였을 때 22.1%의 구현면적 감소효과가 있음을 볼 수 있었다.

V. 결 론

이 논문에서는 OFDM 시스템의 반도체 구현에서 가장 큰 면적을 차지하고 높은 전력을 요구하는 블록인 FFT에 대하여 파이프라인 Radix-4 MDC방식의 저면적 구조를 제안하였다. 각각의 스테이지에 사용되는 나비연산기의 곱셈블록에 대하여 저면적 구조를 제안하였다. 즉, twiddle factor 계수가 많이 사용되는 앞단의 스테이지에는 CORDIC 방식을 사용하였으며, 상대적으로 적은 twiddle factor 계수들이 사용되는 뒷단의 스테이지에는 CSS 방식을 사용하여 효율적인 저면적 구조를 실현하였다. 256-point FFT에 대하여 chartered 0.18um 공정으로 논리합성하고 cell area를 비교하여 기존 구조와 비교하여 22.1%의 면적감소효과가 나타남을 입증하였다.

참 고 문 헌

[1] 김재석, 조용수, 조중휘, 이동통신용 모뎀의 VLSI 설계, 대영사, 2001.
[2] 장영범, 이원상, 김도한, 김비철, 허은성, "Distributed Arithmetic을 사용한 OFDM용 저전

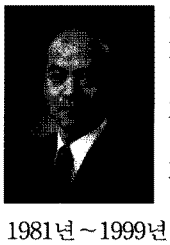
표 4. 각 구조별 전체 FFT블록의 면적비교
Table 4. Cell area comparison of overall FFT blocks for each structure.

	1 stage			2 stage			3 stage			4 stage		Total
	DC256	Adder	multiplier	DC64	Adder	multiplier	DC16	Adder	multiplier	DC4	Adder	
CORDIC	596829.375 (298414.68x2)	28809.95	556443.507	781950.187 (390975.09x2)	28809.95	488165.81	235462.546 (117731.27x2)	28809.95	472561.67	64645.256 (32322.62x2)	28809.95	3311298.16 (100%)
제안구조2 CSS	596829.375 (298414.68x2)	28809.95	721983.659	781950.187 (390975.09x2)	28809.95	199101.708	235462.546 (117731.27x2)	28809.95	31227.46	64645.256 (32322.62x2)	28809.95	2746439.99 (82.94%)
제안구조1 CORDIC +CSS	596829.375 (298414.68x2)	28809.95	556443.507	781950.187 (390975.09x2)	28809.95	199101.708	235462.546 (117731.27x2)	28809.95	31227.46	64645.256 (32322.62x2)	28809.95	2580899.84 (77.94%)

력 Radix-4 FFT 구조", 전자공학회논문지 제43권 SP편 제1호, pp.101-108, 2006년 1월.
[3] R. Sarmiento, V. D. Armas, J. F. Lopez, J. A. Montiel-Nelson, and A. Nunez, "A CORDIC processor for FFT computation and its implementation using gallium arsenide technology", IEEE Trans. on VLSI Systems, vol. 6, No. 1, pp. 18-30, Mar. 1998.
[4] M. Bekooij, J. Huisken, and K. Nowak, "Numerical accuracy of Fast Fourier Transforms with CORDIC arithmetic", Journal of VLSI Signal Processing 25, pp. 187-193, 2000.
[5] 장영범, 최동규, 김도한, "CORDIC을 이용한 OFDM용 저전력 DIF Radix-4 FFT 프로세서", 전자공학회논문지 제45권 SP편 제3호, pp.103-110, 2008년 5월.
[6] 박상윤, 조남익, "CORDIC 알고리즘에 기반한 OFDM 시스템용 8192-Point FFT 프로세서", 한국통신학회논문지, 2002년.
[7] R. I. Hartley, "Subexpression sharing in filters using canonic signed digit multipliers", IEEE Trans. Circuits and Systems-II: Analog and Digital Signal Processing, vol. 43, No.10, pp. 677-688, Oct. 1996.
[8] M. Yagyu, A. Nishihara, and N. Fujii, "Fast FIR digital filter structures using minimal number of adders and its application to filter design," IEICE Trans. Fundamentals of Electronics Communications & Computer Sciences, vol. E79-A No. 8, pp. 1120-1129, Aug. 1996.
[9] Y. Jang, and S. Yang, "Low-power CSD linear phase FIR filter structure using vertical common sub-expression" IEE Electronics Letters, vol. 38, No. 15, pp. 777-779, Jul. 2002.

저 자 소 개

최 동 규(학생회원)
2007년 2월 상명대학교 정보통신공학과 졸업(공학사)
2007년~현재 상명대학교 대학원 컴퓨터정보통신공학과 석사과정
<주관심분야 : 통신신호처리, SoC 설계>



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, 비디오신호처리, SoC 설계>