논문 2008-45SP-3-12

CORDIC을 이용한 OFDM용 저전력 DIF Radix-4 FFT 프로세서

(A Low-power DIF Radix-4 FFT Processor for OFDM Systems Using CORDIC Algorithm)

장 영 범*, 최 동 규**, 김 도 한**

(Young Beom Jang, Dong Kyu Choi, and Do Han Kim)

요 약

이 논문에서는 8K/2K-Point FFT Radix-4 알고리즘을 CORDIC 연산을 이용하여 효율적으로 나비연산 구조를 설계할 수 있음을 보였다. 즉 CORDIC 연산을 사용하여 cosine 과 sine 값을 저장하지 않고 4개의 복소 곱셈연산을 효과적으로 수행할 수 있음을 보였다. 제안된 CORDIC 나비연산기 구조를 Verilog HDL 코딩으로 구현한 결과, 기존의 승산기를 사용한 나비연산기 구조와 비교하여 36.9%의 cell area 감소 효과를 보였다. 또한 전체 8K/2K-point Radix-4 FFT 구조의 Verilog-HDL 코딩을 기존의 승산기를 사용한 구조의 코딩과 비교한 결과, 11.6%의 cell area 감소효과를 볼 수 있었다. 따라서 제안된 FFT 구조는 DMB용 OFDM 모뎀과 같은 큰 크기의 FFT에 효율적으로 사용될 수 있는 구조임을 보였다.

Abstract

In this paper, an efficient butterfly structure for 8K/2K-Point Radix-4 FFT algorithm using CORDIC(coordinate rotation digital computer) is proposed. It is shown that CORDIC can be efficiently used in twiddle factor calculation of the Radix-4 FFT algorithm. The Verilog-HDL coding results for the proposed CORDIC butterfly structure show 36.9% cell area reduction comparison with those of the conventional multiplier butterfly structure. Furthermore, the 8K/2K-point Radix-4 pipeline structure using the proposed butterfly and delay commutators is compared with other conventional structures. Implementation coding results show 11.6% cell area reduction. Due to its efficient processing scheme, the proposed FFT structure can be widely used in large size of FFT like OFDM Modem.

Keywords: Fast Fourier Transform, Radix-4, CORDIC, Twiddle factor

Ⅰ. 서 론

최근에 OFDM(Orthogonal Frequency Division Multiplexing)의 상용화 속도가 점차 빨라지면서 OFDM 단말기용 MODEM SoC(System on a Chip)의 저전력 구현에 대한 연구가 활발히 진행되고 있다. OFDM용 MODEM SoC는 크게 FFT(Fast Fourier Transform) 블록, 동기화 블록, 비터비 블록, 변복조 블록, 등화기 블록 등으로 구성된다. OFDM 전송방식은 직렬로 입력되는 데이터 열을 병렬 데이터 열로 변환한 후에 부반송파에 실어 전송하는 방식이며, 이와 같은 병렬화와 부반송파를 곱하는 동작은 IFFT와 FFT로 구현이 가능하다. 일반적으로 OFDM 시스템에서는 매우 큰 point의 FFT를 필요로 하므로, FFT 블록의 구현 비용과 전력 소모를 줄이는 것이 OFDM용 SoC의 매우 중요한 핵심 요소라고 할 수 있다. 예를 들면 지상파 DMB용 OFDM에서는 246 μs 동안에 2048 point FFT를 수행하여야 하므로 높은 처리율을 갖는 FFT가 요구된다.

* 정회원, 상명대학교 공과대학 정보통신공학과
(College of Engineering, Sangmyung University)
** 학생회원, 상명대학교 대학원 컴퓨터정보통신공학과
(Graduate School, Sangmyung University)
※ 본 연구보고서는 정보통신부 출연금으로 MIC/IITA/ETRI, SoC산업진흥센터에서 수행한 IT SoC 핵심설계인력양성사업의 연구결과입니다.
접수일자:2007년8월29일, 수정완료일:2008년4월15일.

FFT의 복소곱셈연산의 수를 감소시키는 방법은 DIT (Decimation In Time), DIF(Decimation In Frequency), Cooley-Tukey 알고리즘[1] 등이 있는데, 어느 알고리즘이나 많은 곱셈연산을 필요로 한다. 이와 같은 FFT용 곱셈연산을 Booth 알고리즘을 이용하는 방식이 제안되었다.[2] 또한 FFT는 cosine과 sine 값의 곱셈이 주된 연산이므로 CORDIC(COordinate Rotation DIgital Computer)[3] 알고리즘을 사용하는 구조가 제안되었다. 그 가운데서도 Cooley-Tukey 알고리즘에 기반을 둔 CORDIC FFT 프로세서 구조가 제안되었다.[4~5] 이 방식은 나비연산기의 곱셈연산에는 Booth 곱셈기를 사용하였고, 트위들 곱셈연산에는 CORDIC 곱셈기를 사용한 구조이다. 그러나 이 구조는 여러 가지 방식의 곱셈기를 사용하고 있으며, Regular 구조를 갖고 있지 않다. 그밖에 Radix-2와 Radix-4용 나비연산기에 CORDIC 알고리즘을 사용한 다양한 FFT 구조들이 제안되었다.[6~7] 이 논문은 Radix-4 DIF 알고리즘과 CORDIC 알고리즘 기반의 8K/2K-point FFT 구조를 제안한다. Ⅱ장에서는 Radix-4 DIF 고속 알고리즘에 대하여 살펴보고, Ⅲ장에서는 CORDIC 알고리듬을 적용한 FFT 전체구조를 제안한다. Ⅳ장에서는 제안된 구조의 구현 및 검증을 통해 전력과 구현비용의 감소가 이루어진 실험 결과를 제시하였다.

Ⅱ. 8K/2K-point Radix-4 DIF 알고리즘

N-Point의 DFT는 다음과 같이 표현된다.

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

이 논문에서 우리는 Radix-4 DIF에 기반한 FFT 구조를 제안한다. 먼저 식 (1)을 4개의 $N/4$-point로 나누어 표현하면 다음과 같다.

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

식 (2)에서 보듯이 N-Point의 DFT는 4개의 $N/4$-Point DFT로 분해될 수 있게 된다. 여기에서 $x_0(n),x_1(n),x_2(n),x_3(n)$는 다음과 같다.

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

4개의 $N/4$-Point DFT로 분해되기 위해서는 먼저 식 (3)의 덧셈연산과 복소 곱셈 연산이 선행되어야 함을 알 수 있다. 식 (3)을 수행하는 블록을 나비연산기라 부른다. 우리는 이 논문에서 8K/2K-point FFT에 대한 building block을 제안하게 된다. 따라서 제안된 Radix-4 DIF 알고리즘을 사용한 8K/2K-point FFT의 building block은 그림 1과 같이 7개의 스테이지로 구성한다.

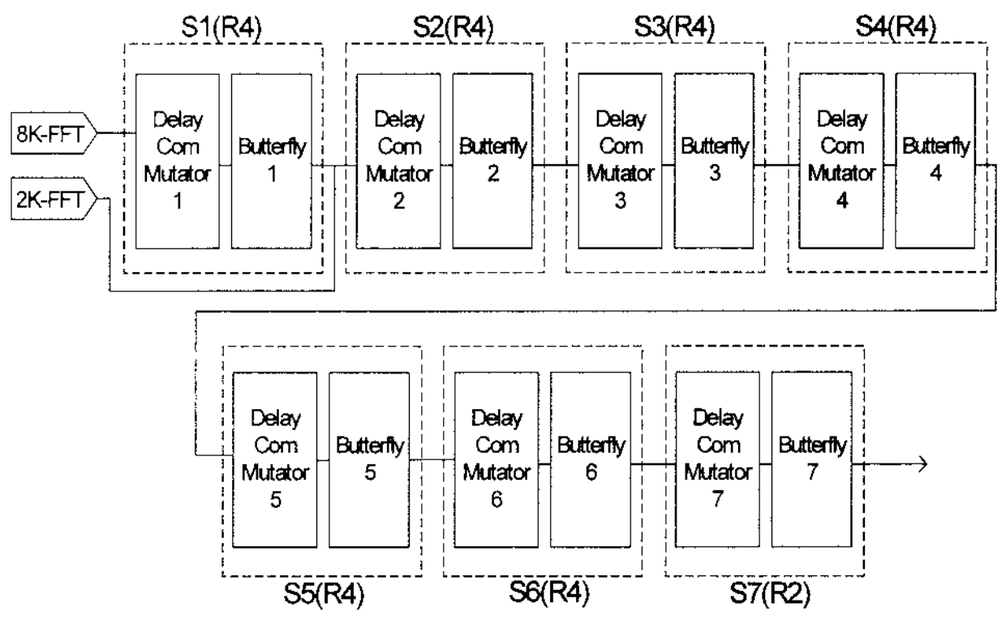


그림 1. 8K/2K-point Radix-4 DIF FFT의 building block
Fig. 1. Building block for 8K/2K-point Radix-4 DIF FFT.

그림 1에서 보듯이, 8K/2K-point이므로 S_1부터 S_6의 스테이지는 Radix-4를 사용하였고, S_7는 Radix-2를 사용하였다. 8K-point FFT를 수행할 때에는 7개의 스테이지를 모두 사용하고, 2K-point FFT에는 S_1 스테이지를 사용하지 않는다. 그림 1에서 보듯이 각각의 스테이지는 지연변환기(Delay Commutator, DC)와 나비연산기(Butterfly, BF)로 구성한다.

Ⅲ. CORDIC을 사용한 나비연산기 구조

3.1 제안된 나비연산기 블록도

식 (3)의 나비연산의 식에서 보듯이 twiddle factor W_N^0, W_N^n, W_N^{2n}, W_N^{3n}와 입력 값, 출력 값이 모두 실수부와 허수부로 구성된 복소수이며, 실제 연산도 실수부분과 허수부분을 따로 계산하여야 한다. 따라서 입출력 값들을 다음과 같이 실수부와 허수부의 값으로 표현하는 것이 편리하다.

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

Radix-4 나비연산의 각각 4개의 입력과 4개의 출력을 모두 복소수로 나타내었으므로 twiddle factor, W^n, W^{2n}, W^{3n}도 모두 직각형 복소수로 나타내어야 계산이 가능하다. 따라서 twiddle factor 역시 실수부와 허수부로 다음과 같이 나타내기로 한다.

$$\begin{aligned} W_N^n &= e^{-j\frac{2\pi n}{N}} = \cos\theta_b - j\sin\theta_b \\ W_N^{2n} &= e^{-j\frac{2\pi 2n}{N}} = \cos\theta_c - j\sin\theta_c \\ W_N^{3n} &= e^{-j\frac{2\pi 3n}{N}} = \cos\theta_d - j\sin\theta_d \end{aligned} \tag{5}$$

실수부와 허수부로 표현된 입출력신호를 사용하여 나비연산기를 나타내면 그림 2와 같다.

이제 가산 연산과 twiddle factor가 연산되어진 후 출력되는 각각의 출력 결과 값을 직각형 복소수로 나타내보도록 한다. 먼저 그림 2에서 첫 번째 나비연산기 출력은 다음 식과 같이 나타낼 수 있다.

$$\begin{aligned} & X_a + jY_a \\ &= (x_a + jy_a) + (x_b + jy_b) + (x_c + jy_c) + (x_d + jy_d) \\ &= (x_a + x_b + x_c + x_d) + j(y_a + y_b + y_c + y_d) \end{aligned} \tag{6}$$

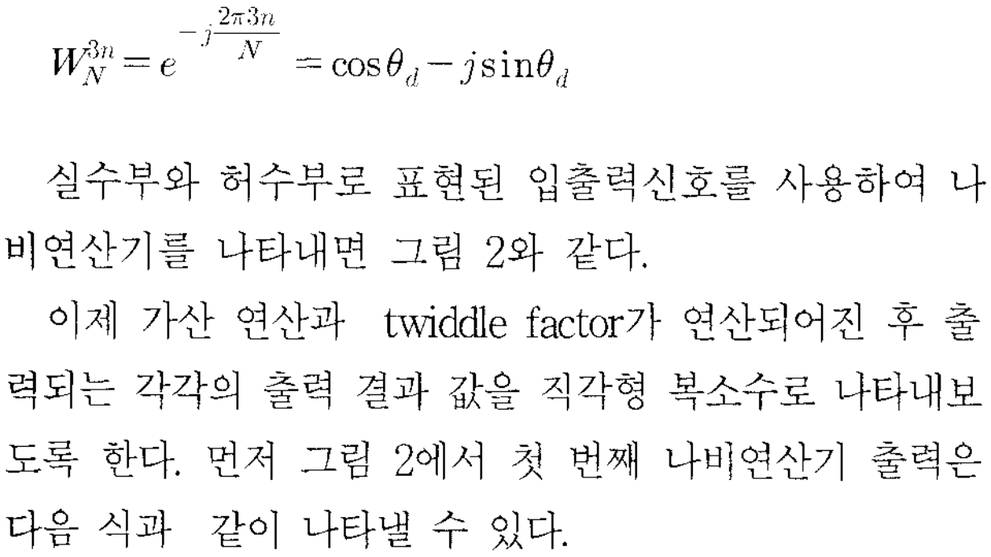


그림 2. 복소수로 나타낸 Radix-4 나비연산기 구조
Fig. 2. Complex Radix-4 butterfly structure.

따라서 X_a와 Y_a는 다음 식과 같이 나타낼 수 있다.

$$\begin{aligned} X_a &= x_a + x_b + x_c + x_d \\ Y_a &= y_a + y_b + y_c + y_d \end{aligned} \tag{7}$$

식 (7)에서 보듯이 나비연산의 첫 출력 값에는 twiddle factor가 1이므로 고려되지 않는다. 그러나 그림 2의 나비연산기의 두 번째 출력 값을 구하는 과정에는 twiddle factor 복소수를 곱하는 연산이 포함된다. 따라서 나비연산기의 두 번째 출력 값은 다음과 같이 나타낼 수 있다.

$$\begin{aligned} & X_b + jY_b \\ &= [(x_a + jy_a) - j(x_b + jy_b) - (x_c + jy_c) + j(x_d + jy_d)] \\ &\quad (\cos\theta_b - j\sin\theta_b) \\ &= [(x_a + y_b - x_c - y_d) + j(y_a - x_b - y_c + x_d)] \\ &\quad (\cos\theta_b - j\sin\theta_b) \end{aligned} \tag{8}$$

식 (8)로부터 X_b와 Y_b는 다음과 같이 나타낼 수 있다.

$$\begin{aligned} X_b &= x_1\cos\theta_b + x_2\sin\theta_b \\ Y_b &= x_2\cos\theta_b - x_1\sin\theta_b \\ x_1 &= x_a + y_b - x_c - y_d \\ x_2 &= y_a - x_b - y_c + x_d \end{aligned} \tag{9}$$

같은 방법으로 세 번째와 네 번째 나비연산기의 출력 값은 다음과 같이 나타낼 수 있다.

$$\begin{aligned} X_c &= x_3\cos\theta_c + x_4\sin\theta_c \\ Y_c &= x_4\cos\theta_c - x_3\sin\theta_c \\ x_3 &= x_a - x_b + x_c - x_d \\ x_4 &= y_a - y_b + y_c - y_d \end{aligned} \tag{10}$$

$$\begin{aligned} X_d &= x_5\cos\theta_d + x_6\sin\theta_d \\ Y_d &= x_6\cos\theta_d - x_5\sin\theta_d \\ x_5 &= x_a - y_b - x_c + y_d \\ x_6 &= y_a + x_b - y_c - x_d \end{aligned} \tag{11}$$

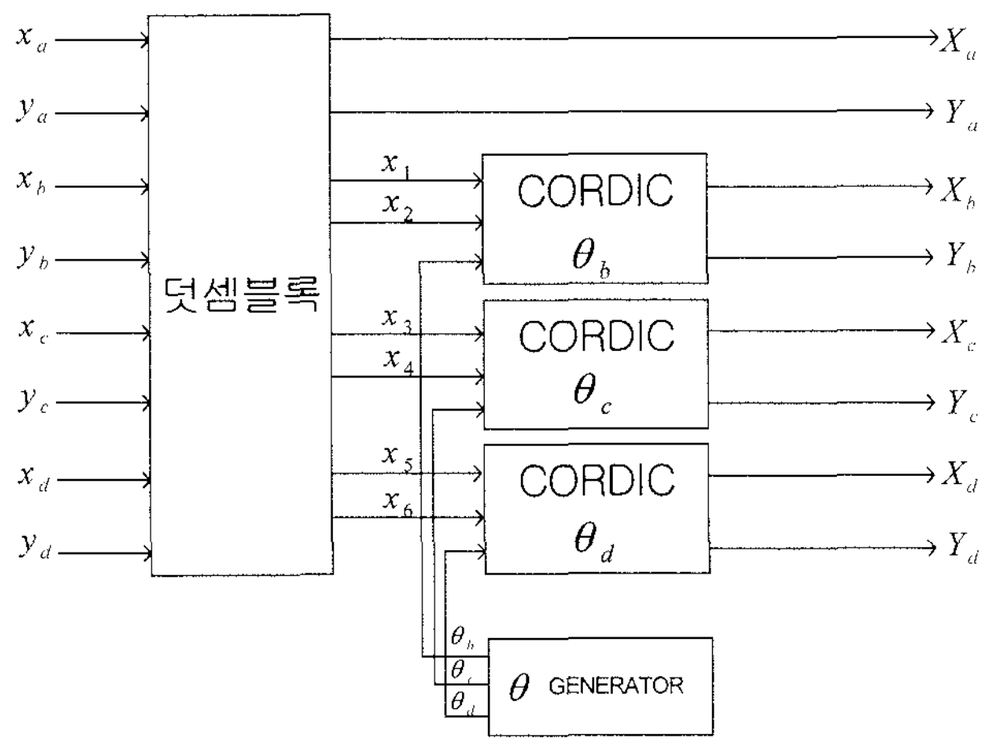


그림 3. 제안된 나비연산기 블록도
Fig. 3. Proposed butterfly block diagram.

지금까지 유도된 식 (7), (9), (10), (11)로 표현되는 나비연산기의 제안 구조는 그림 3과 같다.

3.2 덧셈블록의 세부구조

그림 3의 덧셈블록에서는 식 (7), (9), (10), (11)의 다음의 덧셈연산을 수행한다.

$$X_a = x_a + x_b + x_c + x_d, \quad Y_a = y_a + y_b + y_c + y_d$$
$$x_1 = x_a + y_b - x_c - y_d, \quad x_2 = y_a - x_b - y_c + x_d$$
$$x_3 = x_a - x_b + x_c - x_d, \quad x_4 = y_a - y_b + y_c - y_d$$
$$x_5 = x_a - y_b - x_c + y_d, \quad x_6 = y_a + x_b - y_c - x_d$$

즉 식 (7)의 X_a와 Y_a를 계산하며, 식 (9), (10), (11)의 $x_1, x_2, x_3, x_4, x_5, x_6$을 계산하도록 설계하였으며, 그 덧셈블록의 세부 구조는 그림 4와 같다.

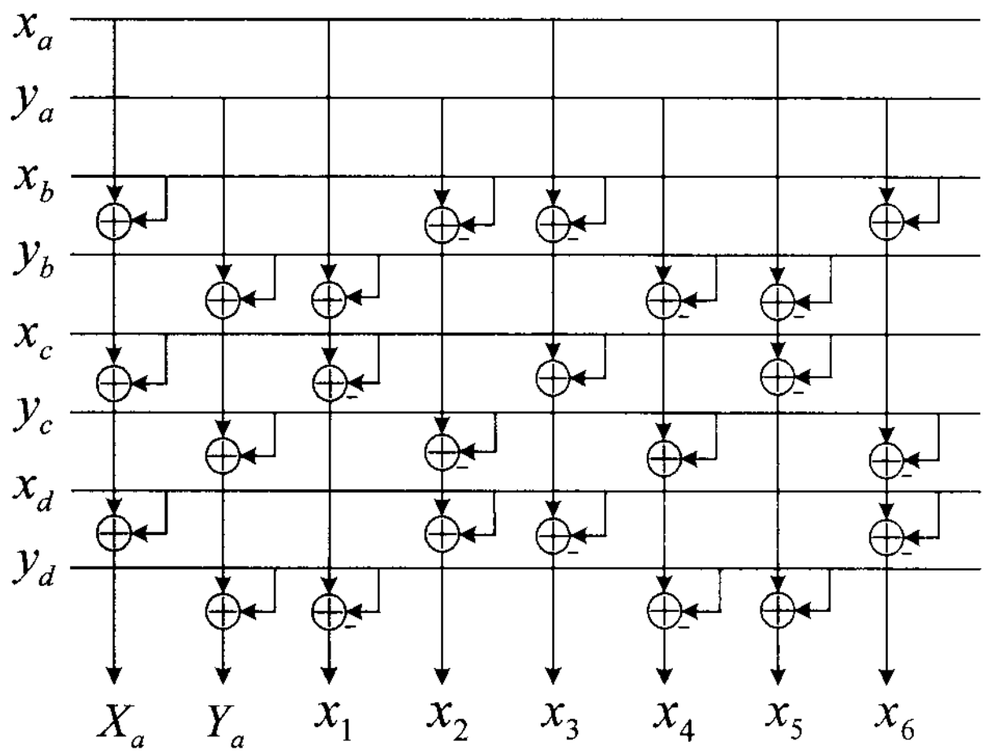


그림 4. 제안된 나비연산용 덧셈블록 구조
Fig. 4. Proposed adder block structure for butterfly.

그림 4의 덧셈블록에서 출력되는 신호 중에서 X_a와 Y_a는 최종 출력이 되며, $x_1, x_2, x_3, x_4, x_5, x_6$는 CORDIC 블록으로 입력되어 다음 계산에 사용된다.

3.3 CORDIC 알고리즘

제안된 회로의 설계를 다루기 위하여 CORDIC 알고리즘에 대하여 소개한다. 전장에서 정리한 Radix-4 식의 cosine값과 sine값을 곱셈 연산하는 twiddle factor 부분에 CORDIC 알고리즘을 적용하여 구현비용감소의 효과를 목적으로 한다. 임의의 벡터 (R,β)의 회전을 고려해보자. 기본 벡터를 X와 Y를 요소로 하는 직각 좌표로 가정하고 이 기본벡터를 양각 θ만큼 회전시키면 벡터 X′와 Y′는 그림 5와 같다. 그림 5에서 보듯이 회전된 벡터

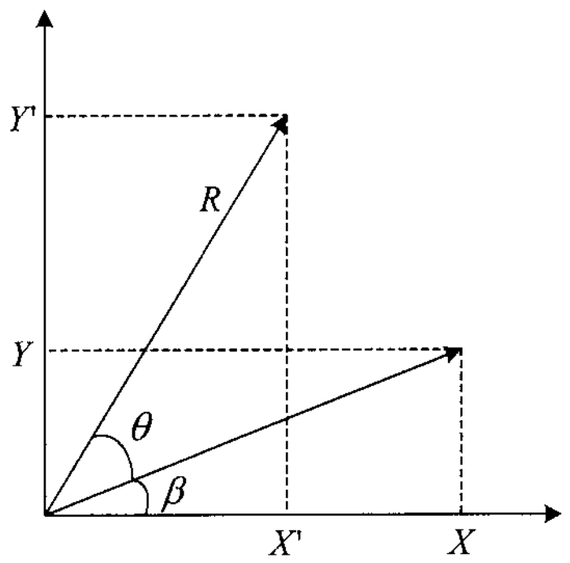


그림 5 회전각 θ를 갖는 벡터 (R, β)
Fig. 5. Vector(R, β) rotated through angle θ.

는 원래의 벡터와 다음과 같은 관계이다.

$$X' = X\cos\theta - Y\sin\theta \qquad Y' = Y\cos\theta + X\sin\theta \tag{12}$$

식 (12)의 양변을 $\cos\theta$로 나누어 다시 표현하면 다음과 같이 나타낼 수 있다.

$$\frac{X'}{\cos\theta} = X - Y\tan\theta \qquad \frac{Y'}{\cos\theta} = Y + X\tan\theta \tag{13}$$

CORDIC 알고리즘은 위의 결과를 한 번에 얻는 것이 아니라 여러 번의 단계를 통하여 결과를 얻는다. 이는 쉬프트와 덧셈연산만을 사용하여야 하기 때문이다. 각 단계에서 벡터를 회전각 크기 $\alpha_i = \tan^{-1}(2^{-i})$만큼 시계방향 혹은 반시계방향으로 회전 시킨다. 각 α_i의 +혹은 -방향은 $(\theta - \sum\alpha_j)$에 따라 선택되어진다. 이때 $(\theta - \sum\alpha_j) > 0$이면 +방향이고 $(\theta - \sum\alpha_j) < 0$이면 -방향이 된다.

θ_i는 단계 i에서 회전된 벡터가 가지고 있는 각의 크기를 총 합한 것이다. 즉, $\theta_i = \pm\alpha_1 \pm\alpha_2 \pm\alpha_3 \pm\cdots\pm\alpha_i$이다. θ_i의 최대 크기가 100°이하 이므로 $|\theta| > 90°$인 각은 우선 90°이하 크기로 조정된다면 특정 각에 대해 접근은 $\pm\alpha_i$각만큼 반복적인 회전과정을 통해 이루어질 수 있다.

$$X_{i+1} = X_i \pm Y_i\tan\alpha_i = X_i \pm Y_i \times 2^{-i}$$
$$Y_{i+1} = Y_i \mp X_i\tan\alpha_i = Y_i \mp X_i \times 2^{-i} \tag{14}$$

여기에서 $i = 0,1,2,3\cdots$이고 X_0와 Y_0는 기본벡터의 X와 Y값이다. 식으로부터 유출한 식에서의 X_{i+1}과 Y_{i+1}은 (X_i, Y_i)벡터보다 $1/\cos\alpha_i$의 비율로 큰 값이 된다. 단계별 $1/\cos\alpha_i$을 보정하기 위해 임의의 상수

K_n를 계산하여 정의할 수 있다.

$$K_n = \prod_{i=0}^{n} K_i = \frac{1}{\cos\alpha_0} \cdot \frac{1}{\cos\alpha_1} \cdot \cdots \frac{1}{\cos\alpha_n} \quad (15)$$
$$= 1.646759954$$

위의 식에서 얻어진K_n의 값은 $n=10$일 때, 즉 10회 반복 계산한 값이다. 다음 절에서는 CORDIC 알고리즘을 적용한 Radix-4 FFT 나비연산기를 제안한다.

3.4 CORDIC 블록의 세부구조

CORDIC 블록에서는 식 (9), (10), (11) 곱셈연산을 수행하도록 설계하였다. 덧셈블록으로부터 들어오는 $x_1, x_2, x_3, x_4, x_5, x_6$를 받아 식(9),(10),(11)의 $\cos\theta, \sin\theta$ 곱셈연산을 수행한다. 즉, θ GENERATOR로부터 만들어지는 각을 이용하여 다음의 곱셈연산을 수행한다.

$$\begin{bmatrix} X_b \\ Y_b \end{bmatrix} = \begin{bmatrix} \cos\theta_b & \sin\theta_b \\ -\sin\theta_b & \cos\theta_b \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \quad (16a)$$

$$\begin{bmatrix} X_c \\ Y_c \end{bmatrix} = \begin{bmatrix} \cos\theta_c & \sin\theta_c \\ -\sin\theta_c & \cos\theta_c \end{bmatrix} \begin{bmatrix} x_3 \\ x_4 \end{bmatrix} \quad (16b)$$

$$\begin{bmatrix} X_d \\ Y_d \end{bmatrix} = \begin{bmatrix} \cos\theta_d & \sin\theta_d \\ -\sin\theta_d & \cos\theta_d \end{bmatrix} \begin{bmatrix} x_5 \\ x_6 \end{bmatrix} \quad (16c)$$

이 절에서 제안하는 CORDIC 구조를 쉽게 설명하기 위하여 식 (16a)를 계산하는 회로를 예로 들어 구조를 유도하도록 한다. 이 예제에서 $x_1 = 0.4$, $x_2 = 0.9$, $\theta_b = 32.47^\circ$ 로 설정하면 다음과 같이 X_b와 Y_b를 얻을 수 있다.

$$\begin{aligned} &P_0 = 0.4,\ Q_0 = 0.9, \\ &P_1 = P_0 \pm 2^0 Q_0 = 0.4 + 1 \times 0.9 = 1.3, \\ &Q_1 = Q_0 \mp 2^0 P_0 = 0.9 - 1 \times 0.4 = 0.5, \\ &P_2 = P_1 \pm 2^{-1} Q_1 = 1.3 - 0.5 \times 0.5 = 1.05, \\ &Q_2 = Q_1 \mp 2^{-1} P_1 = 0.5 + 0.5 \times 1.3 = 1.15, \\ &P_3 = P_2 \pm 2^{-2} Q_2 = 1.05 + 0.25 \times 1.15 = 1.3375, \\ &Q_3 = Q_2 \mp 2^{-2} P_2 = 1.15 - 0.25 \times 1.05 = 0.8875 \end{aligned} \quad (17)$$

P_3, Q_3의 값을 최종 벡터의 절대 값인 1.6298로 나누면 다음과 같이 목표한 결과를 얻을 수 있다.

$$\begin{aligned} 1.3375/1.6298 &= 0.82085 \\ &= 0.4\cos 32.47^\circ + 0.9\sin 32.47^\circ \\ 0.8875/1.6298 &= 0.54455 \\ &= 0.9\cos 32.47^\circ - 0.4\sin 32.47^\circ \end{aligned} \quad (18)$$

표 1. 단계별 CORDIC 블록의 값
Table 1. Values of CORDIC block for steps.

	단계별 P와 Q의 값	
	P	Q
초기 값	0.4	0.9
1단계	1.3	0.5
2단계	1.05	1.15
3단계	1.3375	0.8875
devided by 1.6298	0.82085	0.54455
결과	0.4cos32.47 +0.9sin32.47	0.9cos32.47 -0.4sin32.47

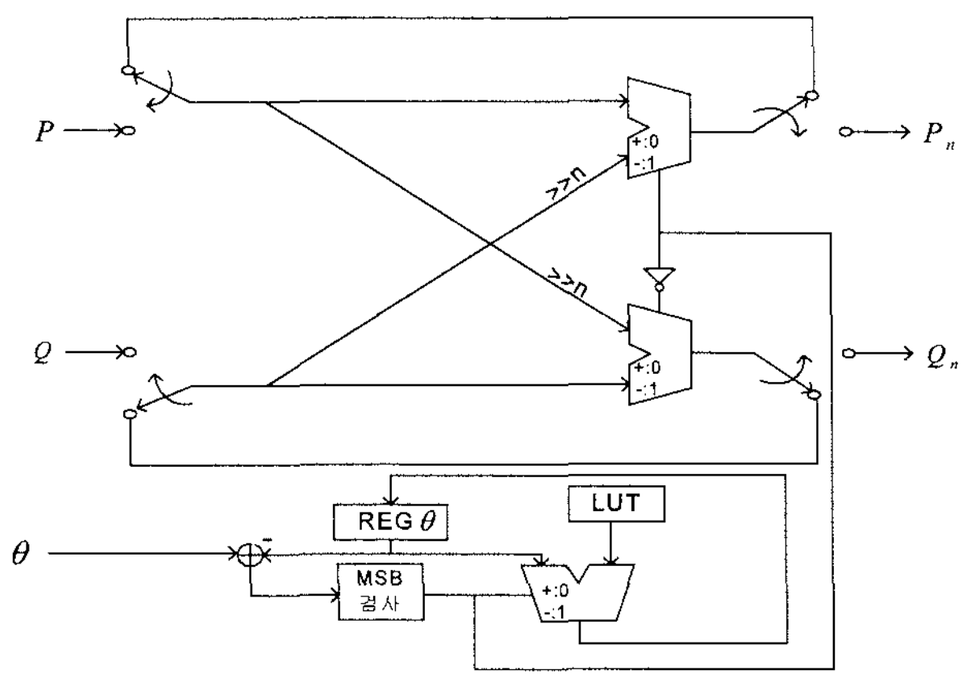


그림 6. 제안된 CORDIC 블록의 세부구조
Fig. 6. Proposed CORDIC block structure.

위의 단계별 결과를 표로 나타내면 표 1과 같다. 제안된 CORDIC 프로세서의 세부구조는 그림 6과 같다. 각각의 덧셈블록으로부터 들어오는 입력 값으로 P와 Q의 레지스터를 초기화하고, θ의 각도만큼 회전하여 $\cos\theta$, $\sin\theta$ 값과 x_1, x_2를 곱하여 더한 값들을 계산해 낸다. 그림 6의 >>n에서 쉬프트되는 횟수는 반복되는 횟수에 따라 n = 0, 1, 2 ,3 , …,17로 증가하도록 조절한다.

3.5 θ GENERATOR 블록의 세부구조

제안 구조는 CORDIC 프로세서를 사용하여 곱셈연산을 수행하므로 cosine이나 sine의 값을 ROM에 저장할 필요가 없다. 또한 θ의 값도 일정한 규칙에 따라 필요하므로 ROM에 저장할 필요 없이 그림 7과 같이 θ GENERATOR를 통해 $\theta_b, \theta_c, \theta_d$를 만들어서 사용할 수 있다. 이 절에서는 S_1 블록의 θ GENERATOR를 다음과 같이 설계한다. 기존의 N-point의 FFT에서는 $N/2$ 만큼의 cosine과 sine의 값을 보관하는 ROM을 필요로 한다. 그러나 제안된 FFT 구조의 S_1 블록의 θ GENERATOR에서는 θ를 저장하지 않고 생성하여 사

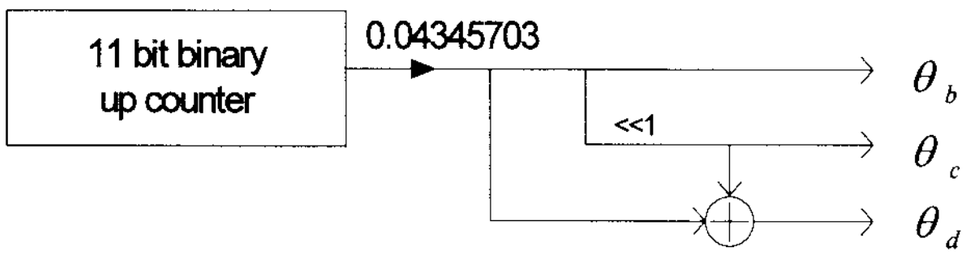


그림 7. 제안된 θ GENERATOR블록의 세부구조
Fig. 7. Proposed θ GENERATOR structure.

표 2. S_1 블록의 θ GENERATOR의 출력값
Table 2. θ GENERATOR output values in S_1 block.

Up-count	θ_b	θ_c	θ_d
0	0	0	0
1	0.04345703	0.08691406	0.13037109
2	0.08691406	0.17382812	0.26074218
3	0.13037109	0.26074218	0.39111328
⋮	⋮	⋮	⋮
⋮	⋮	⋮	⋮
2045	88.86962890	177.73925781	266.60888671
2046	88.91308593	177.82617187	266.73925781
2047	88.95654296	177.91308593	266.86962890

용하므로 부가적인 하드웨어를 줄일 수 있다. 제안된 θ GENERATOR의 세부구조는 그림 7과 같다.

위의 그림의 θ GENERATOR 구조는 S_1 블록의 twiddle factor에서 필요로 하는 각의 값을 W^n, W^{2n}, W^{3n}의 식에 따라 설계한다. 즉, 11 bit binary up counter에서는 0부터 2047까지 카운트하므로 θ_b는 0부터 89.95651296까지 0.04345703씩 증가하게 된다. 또한 θ_c와 θ_d는 각각 $2\theta_b$, $\theta_b+\theta_c$와 같다. 여기에서 생성된 θ_b,θ_c,θ_d는 CORDIC 블록으로 들어가 CORDIC 연산에 이용된다. θ GENERATOR에 의해 생성된 θ값은 표 2와 같다.

IV. 실험 및 고찰

설계된 FFT프로세서는 Xilinx ISE 7.1i로 Verilog코딩을 하여 FPGA기반으로 타이밍을 검증하였다. 검증을 위한 테스트벡터로 Matlab을 통해 8192개의 랜덤 입력값을 산출하였으며, 각각의 입력값을 16 비트 2진수로 변환하여 입력하였으며 16 비트는 부호 1 비트, 정수 4 비트, 실수 11비트로 구성하였다. 먼저 7개의 버터플라이 구조에 대한 코딩결과는 표 3과 같다. 표 3의 Synopsis 합성 결과에서 보듯이 제안된 CORDIC 프로세서를 이용한 버터플라이구조는 399,531개의 게이트로 구성되고, 기존의 승산기를 이용한 구조는 623,719개의 게이트로 구성됨을 보였다.

표 3. 버터플라이블록의 Gate count 비교
Table 3. Gate count for Butterfly Block.

	기존구조	제안구조
덧셈블록	17,269 (2,467x7stage)	17,269 (2,467x7stage)
곱셈블록	606,450 (34,025x3x6stage)	341,874 (18,933x3x6stage)
θ	ROM(131,072bit)	40,388
계	623,719	399,531

표 3에서 보듯이 제안된 CORDIC 프로세서를 사용한 버터플라이 구조가 기존의 승산기를 사용한 구조에 비하여 구현면적이 36.9% 감소하는 효과를 나타내었다.

표 4는 8K-point FFT 전체에 대한 제안 구조와 기존 구조의 Synopsis 논리합성 결과이다. 이 실험에서는 지연변환기도 모두 포함시킨 전체 FFT 구조에 대한 코딩 결과이다. 기존 구조는 승산기를 이용하여 구현한 구조이며, 제안 구조는 제안한 CORDIC 프로세서를 이용하여 구현한 구조의 논리합성 결과이다. 기존 구조는 버터플라이연산에 승산기를 3개 사용하여 구현하였으며, 8K-point FFT에서는 twiddle factor를 저장하기 위한 ROM을 필요로 한다. ROM의 크기는 $N/2$로 정의 되며, 내부 정세도는 16 비트이고, cosine값과 sine값을 가지는 2-word ROM이므로 131,072 비트로 나타난다. 제안 구조에서는 CORDIC 알고리즘 연산의 반복 횟수를 17번으로 결정하였으며, θ GENERATOR를 이용하여 ROM의 사용을 없애고 하드웨어의 효율을 높였다. 기존 구조와 제안 구조 모두 지연변환기는 MDC(Multi-path Delay Commutator)구조를 사용하여 설계하였다. 구현 결과 제안 구조는 기존 구조와 비교하였을 때 11.6%의 구현 면적 감소 효과를 보였다. 또한 기존의 구조에서 필요로 하는 ROM을 사용하지 않으므로 하드웨어비용을 더욱 줄일 수 있다.

표 4. 기존구조와 제안구조의 Gate count 비교
Table 4. Gate count for conventional and proposed structure.

	기존 구조		제안 구조	
	Gate Count	ROM	Gate Count	ROM
1st stage	276,467	131,072bit (16bit)	243,873	-
2nd stage	342,147		306,735	
3rd stage	341,585		277,333	
4th stage	312,414		274,044	
5th stage	300,275		261,469	
6th stage	293,848		252,094	
7th stage	98,558	-	98,558	
계	1,938,294	131,072 bit	1,714,106	-

V. 결 론

이 논문에서는 8K/2K-point Radix-4 알고리즘의 저전력 파이프라인 구조를 제안하였다. 파이프라인 FFT 구조는 버터플라이블록과 지연변환기로 구성되는데, 이 논문은 버터플라이블록을 덧셈블록과 CORDIC 블록으로 지연변환기는 MDC구조로 사용하는 저전력 구조를 제안하였다. 또한 기존 승산기 구조에서 twiddle factor값을 저장하기 위해 필요로 하던 ROM의 사용을 없애기 위하여 counter와 shifter를 이용한 θ GENERATOR를 제안하여 하드웨어의 비용을 줄일 수 있음을 보였다. 면적의 비교를 위해 제안된 CORDIC 프로세서 알고리즘과 기존의 승산기를 이용한 알고리즘을 하드웨어로 구현하였고, Synopsys논리 합성한 결과, 제안된 구조를 이용하는 경우 기존 구조보다 약 11.6%의 면적 감소 효과가 나타남을 알 수 있었다. 그리고 θ GENERATOR를 통해 기존의 구조에서 필요로 했던 ROM을 사용하지 않음으로써 하드웨어의 비용을 더욱 줄일 수 있음을 보였다.

참 고 문 헌

[1] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series", *Mathematics of Computation*, vol. 19, No. 90, pp. 297-301, Apr. 1965.

[2] E Bidet, D. Castelain, C. Joanblanq, and P. Senn, "A fast single-chip implementation of 8192 complex point FFT", *IEEE Journal of Solid-State Circuits*, vol. 30, No. 3, pp. 300-305, Mar. 1995.

[3] J. Volder, "The CORDIC trigonometric computing technique", *IRE Trans. on Electronic Computers*, vol. EC-8, No. 3, pp. 330-334, Sept. 1959.

[4] 박상윤, 조남익, "CORDIC 알고리즘에 기반한 OFDM 시스템용 8192-Point FFT 프로세서", 한국통신학회논문지, 2002년

[5] 박상윤, 조남익, "CORDIC 알고리즘에 기반한 DVB-T용 2K/4K/8K-Point FFT 프로세서", 제14회 신호처리합동학술대회, pp. 261-264, 2001년

[6] 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.

[7] 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.

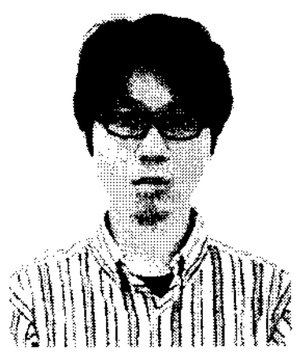
저 자 소 개

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, 비디오신호처리, SoC 설계>

김 도 한(학생회원)

2006년 2월 상명대학교 정보통신공학과 졸업(공학사)
2006년~2008년 2월 상명대학교 대학원 컴퓨터정보통신공학과 졸업(공학석사)
<주관심분야 : 통신신호처리, SoC 설계>



최 동 규(학생회원)

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