Journal of the Korea Academia-Industrial cooperation Society
Vol. 16, No. 8 pp. 5579-5585, 2015

http://dx.doi.org/10.5762/KAIS.2015.16.8.5579
ISSN 1975-4701 / eISSN 2288-4688

# 새로운 DIT Radix-8 FFT 나비연산기 구조

장영범[1*]
[1]상명대학교 정보통신공학과

# New DIT Radix-8 FFT Butterfly Structure

**Young-Beom Jang**[1*]
[1]**Department of Information and Communication Engineering, Sangmyung University**

**요 약** FFT(Fast Fourier Transform)는 DIT(Decimation -In-Time)와 DIF(Decimation-In-Frequency) 방식이 주로 사용되고 있다. DIF 방식은 Radix-2/4/8 등의 다양한 구조와 그 구현 방법이 개발되어 사용되고 있는데 반하여 DIT 방식은 순차적인 출력을 낼 수 있는 장점이 있음에도 불구하고 다양한 구조와 그 구현방법이 연구되지 못하였다. 이 논문에서는 순차적인 출력을 낼 수 있는 DIT Radix-8 FFT용 나비연산기 구조를 제안한다. 또한 기존에 주로 사용되어 온 Radix-2나 Radix-4 구조는 스테이지 수가 많아 연산지연시간이 길어지는 단점이 있다. 제안구조는 Radix-8의 알고리즘을 사용하였으므로 연산지연이 상대적으로 짧으며, 특히 큰 point의 FFT 구조의 경우에 스테이지의 수가 작아지는 장점을 갖는다. 제안구조의 나비연산기를 사용하여 4096-point FFT를 설계할 경우에, 4096개의 출력이 순서대로 출력되는 장점뿐 아니라 4개의 스테이지로 구성되므로 Radix-2를 사용하는 12 스테이지보다 연산지연이 짧은 장점을 갖는다. 따라서 제안 구조는 순차적인 출력과 짧은 연산지연을 요구하는 OFDM용 반도체 칩의 FFT 블록에 사용될 수 있다.

**Abstract** In FFT(Fast Fourier Transform) implementation, DIT(Decimation-In-Time) and DIF (Decimation-In-Frequency) methods are mostly used. Among them, various DIF structures such as Radix-2/4/8 algorithm have been developed. Compared to the DIF, the DIT structures have not been investigated even though they have a big advantage producing a sequential output. In this paper, a butterfly structure for DIT Radix-8 algorithm is proposed. The proposed structure has smaller latency time because of Radix-8 algorithm in addition to the advantage of the sequential output. In case of 4096-point FFT implementation, the proposed structure has only 4 stages which shows the smaller latency time compared to the 12 stages of Radix-2 algorithm. The proposed butterfly can be used in FFT block required the sequential output and smaller latency time.

**Keywords :** Butterfly, DIT, FFT, Radix-8, SoC

## 1. 서론

FFT(Fast Fourier Transform)는 DIT (Decimation-In-Time)와 DIF(Decimation-In -Frequency) 방식이 주로 사용되고 있다. 그 중에서 DIF 알고리즘은 Radix-2/4/8 등의 알고리즘이 개발되었고 다양한 구현방법이 제안되었다[1-7]. 특히 DIF Radix-2 알고리즘에는 Radix-2, Radix-$2^2$, Radix-$2^3$, Radix-$2^4$ 등의 알고리즘의 구조가 연구되었고, DIF Radix-4 알고리즘도 Radix-4, Radix-$4^2$ 등의 알고리즘의 구조가 연구되었다. 이와 더불어 DIF Radix-8의 구조도 연구되었다. 이와 비교하여 DIT는 순차적인 출력을 낼 수 있는 장점이 있음에도 다양한 알고리즘이 개발되지 못하였다[8-10]. 이 논문에서는 DIT Radix-8 FFT에 사용되는 나비연산기 구조를 제안한다. DIT Radix-8 알고리즘은 순차적인 출력을 낼 수 있는 장점과 Radix-2나 Radix-4 알고리즘과 비교하여 큰 포

본 논문은 상명대학교 2014년도 교내연구비에 의하여 수행되었음.
*Corresponding Author : Young-Beom Jang(Sangmyung University)
Tel: +82-41-550-5353 email: ybjang@smu.ac.kr
Received May 29, 2015 Revised August 4, 2015
Accepted August 6, 2015 Published August 31, 2015

인트의 FFT에서도 스테이지 수가 작아서 지연시간이 상대적으로 적은 장점이 있다. 이 논문의 2장에서는 DIT Radix-8 알고리즘을 소개하고, 3장에서는 DIT Radix-8 알고리즘의 구현을 위한 나비연산기 구조를 제안한다. 4장에서 function simulation을 수행하고 5장에서 결론을 맺는다.

## 2. DIT Radix-8 FFT 알고리즘

*N*-point DFT(Discrete Fourier Transform)의 정의는 다음과 같다.

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

여기서 $W_N$은 $e^{-j2\pi/N}$이다. DIT Radix-8의 알고리즘을 유도하기 위해서 먼저 다음과 같이 분해한다.

$$\begin{aligned}X(k)=&\sum_{n=0,8,\cdots}x(n)\,W_N^{nk}+\sum_{n=1,9,\cdots}x(n)\,W_N^{nk}\\&+\cdots+\sum_{n=7,15,\cdots}x(n)\,W_N^{nk}\end{aligned} \tag{2}$$

이 식은 다시 다음과 같이 $N_1=N/8$의 8개의 합으로 분해할 수 있다.

$$\begin{aligned}X(k)=&\sum_{r=0}^{N_1-1}x(8r)\,W_N^{8rk}+\sum_{r=0}^{N_1-1}x(8r+1)\,W_N^{(8r+1)k}\\&+\cdots+\sum_{r=0}^{N_1-1}x(8r+7)\,W_N^{(8r+7)k}\end{aligned} \tag{3}$$

식 (3)에서 $x(8r)$ 등의 신호를 다음과 같이 새로 정의한다.

$$\begin{aligned}a(r)&=x(8r)\\b(r)&=x(8r+1)\\&\vdots\\h(r)&=x(8r+7)\end{aligned} \tag{4}$$

이 정의와 $W_N^8=W_{N_1}$을 사용하여 식 (3)을 간략화하면 다음과 같이 나타낼 수 있다.

$$\begin{aligned}X(k)=&\sum_{r=0}^{N_1-1}a(r)\,W_{N_1}^{rk}+W_N^{k}\sum_{r=0}^{N_1-1}b(r)\,W_{N_1}^{rk}\\&+\cdots+W_N^{7k}\sum_{r=0}^{N_1-1}h(r)\,W_{N_1}^{rk}\end{aligned} \tag{5}$$

식 (5)에서 $a(r)$, $b(r),\cdots,h(r)$의 $N_1$-point DFT를 각각 다음과 같이 정의한다.

$$\begin{aligned}A(k)&=\sum_{r=0}^{N_1-1}a(r)\,W_{N_1}^{rk}\\B(k)&=\sum_{r=0}^{N_1-1}b(r)\,W_{N_1}^{rk}\\&\vdots\\H(k)&=\sum_{r=0}^{N_1-1}h(r)\,W_{N_1}^{\ rk}\end{aligned} \tag{6}$$

이 정의를 사용하여 $X(k)$를 다음과 같이 나타낼 수 있다.

$$X(k)=A(k)+W_N^kB(k)+\cdots+W_N^{7k}H(k) \tag{7}$$

그런데 식 (7)은 오직 인덱스 $k$가 0부터 $N_1-1$까지 유효하므로 $k$가 $N_1$부터 $2N_1-1$까지 유효한 식을 다시 유도해야한다. 식 (7)을 사용하여 $k$가 $N_1$부터 $2N_1-1$인 경우를 나타내면 다음과 같다.

$$\begin{aligned}X(k+N_1)=&A(k+N_1)+W_N^{(k+N_1)}B(k+N_1)\\&+\cdots+W_N^{7(k+N_1)}H(k+N_1)\end{aligned} \tag{8}$$

이 식에서 $A(k+N_1)$ 등은 다음과 같이 나타낼 수 있다.

$$\begin{aligned}A(k+N_1)&=\sum_{r=0}^{N_1-1}a(r)\,W_{N_1}^{r(k+N_1)}=A(k)\\B(k+N_1)&=\sum_{r=0}^{N_1-1}b(r)\,W_{N_1}^{r(k+N_1)}=B(k)\\&\vdots\\H(k+N_1)&=\sum_{r=0}^{N_1-1}h(r)\,W_{N_1}^{r(k+N_1)}=H(k)\end{aligned} \tag{9}$$

또한 식 (8)에서 $W_N^{N_1}$ 등은 다음과 같이 나타낼 수 있다.

$$\begin{aligned}W_N^{N_1}&=m(1-j)\\W_N^{2N_1}&=-j\\W_N^{3N_1}&=m(-1-j)\\W_N^{4N_1}&=-1\\W_N^{5N_1}&=m(-1+j)\\W_N^{6N_1}&=j\\W_N^{7N_1}&=m(1+j)\end{aligned} \tag{10}$$

여기에서 $m$은 0.7071이다. 식 (9)와 (10)을 사용하여 식 (8)을 다음과 같이 쓸 수 있다.

$$\begin{aligned}X(k+N_1)=&A(k)+m(1-j)\,W_N^kB(k)-jW_N^{2k}C(k)\\&+\cdots+m(1+j)\,W_N^{7k}H(k)\end{aligned} \tag{11}$$

$X(k+N_1)$을 유도한 것과 같은 방법으로 다음 6개의 식을 유도할 수 있다.

$$\begin{aligned}
X(k+2N_1) &= A(k) - jW_N^k B(k) - W_N^{2k} C(k) \\
&\quad + \cdots + jW_N^{7k} D(k) \\
X(k+3N_1) &= A(k) + m(-1-j)W_N^k B(k) + jW_N^{2k} C(k) \\
&\quad + \cdots + m(-1+j)W_N^{7k} H(k) \\
X(k+4N_1) &= A(k) - W_N^k B(k) + W_N^{2k} C(k) \\
&\quad + \cdots - W_N^{7k} D(k) \\
X(k+5N_1) &= A(k) + m(-1+j)W_N^k B(k) - jW_N^{2k} C(k) \\
&\quad + \cdots + m(-1-j)W_N^{7k} H(k) \\
X(k+6N_1) &= A(k) + jW_N^k B(k) - W_N^{2k} C(k) \\
&\quad + \cdots - jW_N^{7k} D(k) \\
X(k+7N_1) &= A(k) + m(1+j)W_N^k B(k) + jW_N^{2k} C(k) \\
&\quad + \cdots + m(1-j)W_N^{7k} H(k)
\end{aligned} \tag{12}$$

지금까지 유도한 식 (7), (11), (12)를 사용하여 나비연산기를 나타내면 다음 그림과 같다.

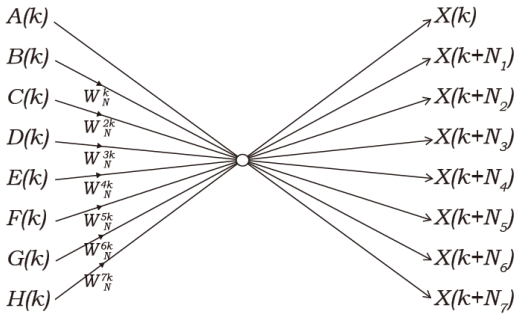



Fig. 1. DIT Radix-8 butterfly

다음 절에서는 반도체 구현이 가능한 나비연산기 구조를 제안한다.

## 3. 제안된 DIT Radix-8 나비연산기 구조

이 절에서는 1절의 DIT Radix-8의 반도체 구현을 위한 나비연산기 구조를 제안한다. 반도체 칩을 사용하여 FFT 회로를 구현하기 위해서는 모든 입출력을 실수와 허수로 나타내어 연산한다. 따라서 나비연산기를 그림 2와 같이 정의한다.

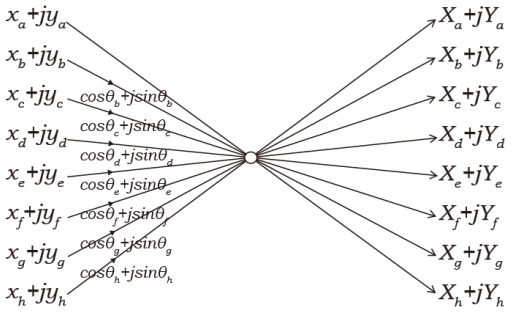



Fig. 2. DIT Radix-8 butterfly with rectangular form of complex number

그림 2에서 보듯이, 구현을 위한 나비연산기 구조에서는 입력과 출력을 다음과 같이 모두 직각형 복소수로 정의한다.

$$\begin{aligned}
A(k) &= x_a + jy_a, \;\; X(k) = X_a + jY_a \\
B(k) &= x_b + jy_b, \;\; X(k+N_1) = X_b + jY_b \\
&\vdots \qquad\qquad \vdots \\
H(k) &= x_h + jy_h, \;\; X(k+7N_1) = X_h + jY_h
\end{aligned} \tag{13}$$

그리고 트위들 인자도 다음과 같이 직각형 복소수로 다시 정의한다.

$$\begin{aligned}
W_N^k &= \cos\theta_b - j\sin\theta_b \\
W_N^{2k} &= \cos\theta_c - j\sin\theta_c \\
&\vdots \\
W_N^{7k} &= \cos\theta_h - j\sin\theta_h
\end{aligned} \tag{14}$$

이제부터 그림 2의 나비연산기에서 출력을 구하는 하드웨어설 설계해보기로 한다. 먼저 그림 2에서 입력과 트위들 인자의 곱셈 결과를 각각 프라임을 붙여서 정의한다.

$$\begin{aligned}
x'_b + jy'_b &= (\cos\theta_b - j\sin\theta_b)(x_b + jy_b) \\
x'_c + jy'_c &= (\cos\theta_c - j\sin\theta_c)(x_c + jy_c) \\
&\vdots \\
x'_h + jy'_h &= (\cos\theta_h - j\sin\theta_h)(x_h + jy_h)
\end{aligned} \tag{15}$$

식 (15)에서 실수부와 허수부를 정리하면 각각 다음과 같다.

$$\begin{aligned}
x'_b &= x_b\cos\theta_b + y_b\sin\theta_b, \;\; y'_b = y_b\cos\theta_b - x_b\sin\theta_b \\
x'_c &= x_c\cos\theta_c + y_c\sin\theta_c, \;\; y'_c = y_c\cos\theta_c - x_c\sin\theta_c \\
x'_d &= x_d\cos\theta_d + y_d\sin\theta_d, \;\; y'_d = y_d\cos\theta_d - x_d\sin\theta_d \\
x'_e &= x_e\cos\theta_e + y_e\sin\theta_e, \;\; y'_e = y_e\cos\theta_e - x_e\sin\theta_e \\
x'_f &= x_f\cos\theta_f + y_f\sin\theta_f, \;\; y'_f = y_f\cos\theta_f - x_f\sin\theta_f \\
x'_g &= x_g\cos\theta_g + y_g\sin\theta_g, \;\; y'_g = y_g\cos\theta_g - x_g\sin\theta_g \\
x'_h &= x_h\cos\theta_h + y_h\sin\theta_h, \;\; y'_h = y_h\cos\theta_h - x_h\sin\theta_h
\end{aligned} \tag{16}$$

식 (16)의 프라임 신호를 이용하여 그림 2의 8개의 나비연산기 출력을 다음과 같이 차례로 구하여야 한다.

(i) 첫 번째 출력

$$\begin{aligned}&X_a+jY_a=(x_a+jy_a)+(x'_b+jy'_b)\\&+(x'_c+jy'_c)+(x'_d+jy'_d)+(x'_e+jy'_e)\\&+(x'_f+jy'_f)+(x'_g+jy'_g)+(x'_h+jy'_h)\\&X_a=x_a+x'_b+x'_c+x'_d+x'_e+x'_f+x'_g+x'_h\\&Y_a=y_a+y'_b+y'_c+y'_d+y'_e+y'_f+y'_g+y'_h\end{aligned}\quad(17)$$

(ii) 두 번째 출력

Table 1. Second output of the butterfly

| multiplier | multiplicand | $X_b$(real) | $Y_b$(imaginary) |
|---|---|---|---|
| 1 | $x_a+jy_a$ | $x_a$ | $y_a$ |
| $m(1-j)$ | $x'_b+jy'_b$ | $m(x'_b+y'_b)$ | $m(y'_b-x'_b)$ |
| $-j$ | $x'_c+jy'_c$ | $y'_c$ | $-x'_c$ |
| $m(-1-j)$ | $x'_d+jy'_d$ | $m(y'_d-x'_d)$ | $m(-x'_d-y'_d)$ |
| -1 | $x'_e+jy'_e$ | $-x'_e$ | $-y'_e$ |
| $m(-1+j)$ | $x'_f+jy'_f$ | $m(-x'_f-y'_f)$ | $m(x'_f-y'_f)$ |
| $j$ | $x'_g+jy'_g$ | $-y'_g$ | $x'_g$ |
| $m(1+j)$ | $x'_h+jy'_h$ | $m(x'_h-y'_h)$ | $m(x'_h+y'_h)$ |

$$\begin{aligned}X_b&=(x_a+y'_c-x'_e-y'_g)\\&+m(x'_b+y'_b+y'_d-x'_d-x'_f-y'_f+x'_h-y'_h)\\Y_b&=(y_a-x'_c-y'_e+x'_g)\\&+m(y'_b-x'_b-x'_d-y'_d+x'_f-y'_f+x'_h+y'_h)\end{aligned}\quad(18)$$

(iii) 세 번째 출력

$$\begin{aligned}&X_c+jY_c=(x_a+jy_a)-j(x'_b+jy'_b)\\&-(x'_c+jy'_c)+j(x'_d+jy'_d)+(x'_e+jy'_e)\\&-j(x'_f+jy'_f)-(x'_g+jy'_g)+j(x'_h+jy'_h)\\&X_c=x_a+y'_b-x'_c-y'_d+x'_e+y'_f-x'_g-y'_h\\&Y_c=y_a-x'_b-y'_c+x'_d+y'_e-x'_f-y'_g+x'_h\end{aligned}\quad(19)$$

(iv) 네 번째 출력

Table 2. Fourth output of the butterfly

| multiplier | multiplicand | $X_d$(real) | $Y_d$(imaginary) |
|---|---|---|---|
| 1 | $x_a+jy_a$ | $x_a$ | $y_a$ |
| $m(-1-j)$ | $x'_b+jy'_b$ | $m(-x'_b+y'_b)$ | $m(-y'_b-x'_b)$ |
| $j$ | $x'_c+jy'_c$ | $-y'_c$ | $x'_c$ |
| $m(1-j)$ | $x'_d+jy'_d$ | $m(x'_d+y'_d)$ | $m(-x'_d+y'_d)$ |
| -1 | $x'_e+jy'_e$ | $-x'_e$ | $-y'_e$ |
| $m(1+j)$ | $x'_f+jy'_f$ | $m(x'_f-y'_f)$ | $m(x'_f+y'_f)$ |
| $-j$ | $x'_g+jy'_g$ | $y'_g$ | $-x'_g$ |
| $m(-1+j)$ | $x'_h+jy'_h$ | $m(-x'_h-y'_h)$ | $m(x'_h-y'_h)$ |

$$\begin{aligned}X_d&=(x_a-y'_c-x'_e+y'_g)\\&+m(y'_b-x'_b+x'_d+y'_d+x'_f-y'_f-x'_h-y'_h)\\Y_d&=(y_a+x'_c-y'_e-x'_g)\\&+m(-x'_b-y'_b-x'_d+y'_d+x'_f+y'_f+x'_h-y'_h)\end{aligned}\quad(20)$$

(v) 다섯 번째 출력

$$\begin{aligned}&X_e+jY_e=(x_a+jy_a)-(x'_b+jy'_b)\\&+(x'_c+jy'_c)-(x'_d+jy'_d)+(x'_e+jy'_e)\\&-(x'_f+jy'_f)+(x'_g+jy'_g)-(x'_h+jy'_h)\\&X_e=x_a-x'_b+x'_c-x'_d+x'_e-x'_f+x'_g-x'_h\\&Y_e=y_a-y'_b+y'_c-y'_d+y'_e-y'_f+y'_g-y'_h\end{aligned}\quad(21)$$

(vi) 여섯 번째 출력

Table 3. Sixth output of the butterfly

| multiplier | multiplicand | $X_f$(real) | $Y_f$(imaginary) |
|---|---|---|---|
| 1 | $x_a+jy_a$ | $x_a$ | $y_a$ |
| $m(-1+j)$ | $x'_b+jy'_b$ | $m(-x'_b-y'_b)$ | $m(x'_b-y'_b)$ |
| $-j$ | $x'_c+jy'_c$ | $y'_c$ | $-x'_c$ |
| $m(1+j)$ | $x'_d+jy'_d$ | $m(x'_d-y'_d)$ | $m(x'_d+y'_d)$ |
| -1 | $x'_e+jy'_e$ | $-x'_e$ | $-y'_e$ |
| $m(1-j)$ | $x'_f+jy'_f$ | $m(x'_f+y'_f)$ | $m(-x'_f+y'_f)$ |
| $j$ | $x'_g+jy'_g$ | $-y'_g$ | $x'_g$ |
| $m(-1-j)$ | $x'_h+jy'_h$ | $m(-x'_h+y'_h)$ | $m(-x'_h-y'_h)$ |

$$\begin{aligned}X_f&=(x_a+y'_c-x'_e-y'_g)\\&+m(-x'_b-y'_b+x'_d-y'_d+x'_f+y'_f-x'_h+y'_h)\\Y_f&=(y_a-x'_c-y'_e+x'_g)\\&+m(x'_b-y'_b+x'_d+y'_d-x'_f+y'_f-x'_h-y'_h)\end{aligned}\quad(22)$$

(vii) 일곱 번째 출력

$$\begin{aligned}&X_g+jY_g=(x_a+jy_a)+j(x'_b+jy'_b)\\&-(x'_c+jy'_c)-j(x'_d+jy'_d)+(x'_e+jy'_e)\\&+j(x'_f+jy'_f)-(x'_g+jy'_g)-j(x'_h+jy'_h)\\&X_g=x_a-y'_b-x'_c+y'_d+x'_e-y'_f-x'_g+y'_h\\&Y_g=y_a+x'_b-y'_c-x'_d+y'_e+x'_f-y'_g-x'_h\end{aligned}\quad(23)$$

(viii) 여덟 번째 출력

Table 4. Eighth output of the butterfly

| multiplier | multiplicand | $X_h$ (real) | $Y_h$ (imaginary) |
|---|---|---|---|
| 1 | $x_a+jy_a$ | $x_a$ | $y_a$ |
| $m(1+j)$ | $x'_b+jy'_b$ | $m(x'_b-y'_b)$ | $m(x'_b+y'_b)$ |
| $j$ | $x'_c+jy'_c$ | $-y'_c$ | $x'_c$ |
| $m(-1+j)$ | $x'_d+jy'_d$ | $m(-x'_d-y'_d)$ | $m(x'_d-y'_d)$ |
| -1 | $x'_e+jy'_e$ | $-x'_e$ | $-y'_e$ |
| $m(-1-j)$ | $x'_f+jy'_f$ | $m(-x'_f+y'_f)$ | $m(-x'_f-y'_f)$ |
| $-j$ | $x'_g+jy'_g$ | $y'_g$ | $-x'_g$ |
| $m(1-j)$ | $x'_h+jy'_h$ | $m(x'_h+y'_h)$ | $m(-x'_h+y'_h)$ |

$$X_h = (x_a - y'_c - x'_e + y'_g) + m(x'_b - y'_b - x'_d - y'_d - x'_f + y'_f + x'_h + y'_h) \quad (24)$$
$$Y_f = (y_a + x'_c - y'_e - x'_g) + m(x'_b + y'_b + x'_d - y'_d - x'_f - y'_f - x'_h + y'_h)$$

지금까지 유도한 곱셈과 덧셈의 식을 이용하여 나비연산기를 설계하면 그림 3과 같다. 그림 3에서 mult 블록들은 식 (16)을 연산하고 Adder 블록은 식 (17) ~ (24)를 연산한다.

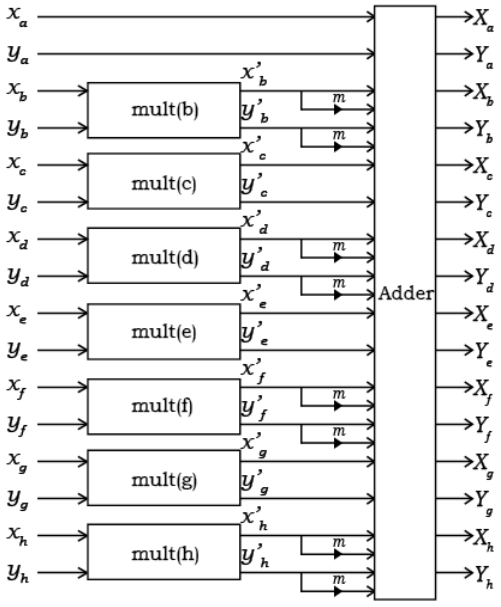
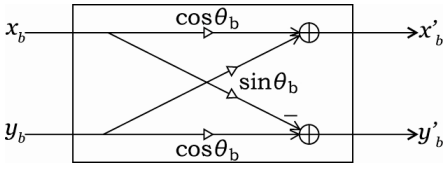
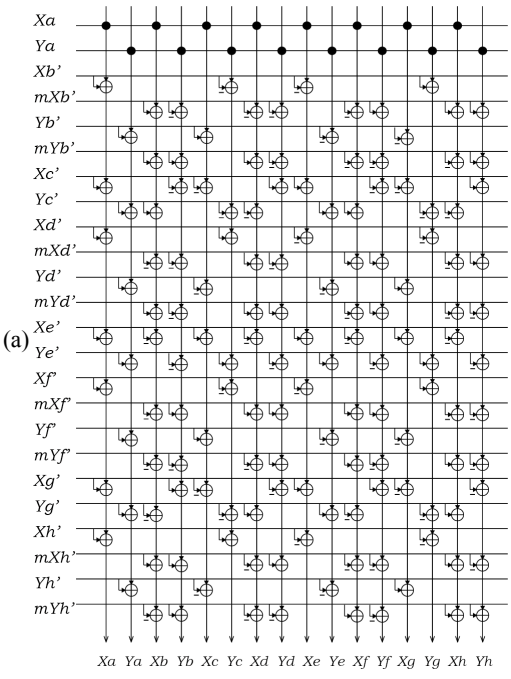



Fig. 3. Proposed DIT Radix-8 butterfly structure

제안된 그림 3의 구조에서 곱셈기 블록(mult)과 덧셈기 블록(Adder)의 내부 구조는 그림 4와 같다.




(a)




(a)

(b)

Fig. 4. (a) mult block (b) adder block of Fig. 3

## 4. Function simulation

이 장에서는 제안된 나비연산기 구조에 대하여 정상동작을 확인하기 위해 64-point FFT 구조를 사용하여 function simulation을 수행하였다. Function simulation에서는 그림 5의 64-point DIT Radix-8 FFT의 SFG를 사용하였다. 그림 5에서 보듯이 Radix-8의 알고리즘을 사용하였으므로 64-point FFT를 2개의 스테이지를 사용하여 구현하였으며, 첫 번째 스테이지에서는 복소 곱셈기가 사용되지 않는다. 두 번째 스테이지에서는 복소 곱셈기를 $(n)$으로 표기하였으며 $(n) = W_{64}^{n} = e^{-j2\pi n/64}$로 정의하여 64개가 사용된다. 실제로 $(0)$은 곱셈기가 아니므로 49개의 복소 곱셈기가 사용된다. 또한 그림 5의 나비연산기에서는 그림 3과 그림 4의 구조를 사용하였다. 사용된 입력 벡터 $x[0] \sim x[63]$은 표 5의 왼쪽과 같은 $-1.0 \sim 1.0$ 사이의 64개의 실수 값을 사용하였다.

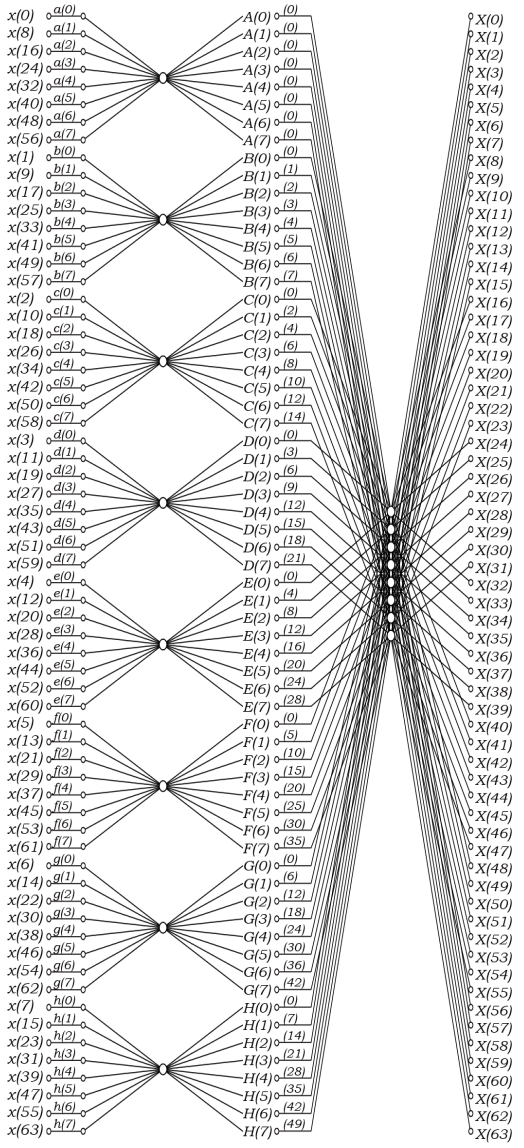


Fig. 5. SFG of the 64-point DIT Radix-8 FFT

Table 5. Input vectors and output for function simulation

| | |
|---|---|
| x[0]= 0.4605 | X(0)= -2.3592 + 0.0000i |
| x[8]= 0.1887 | X(1)= 0.1146 + 3.2110i |
| x[16]= 0.1970 | X(2)= 1.0585 + 1.6702i |
| x[24]= 0.0617 | X(3)= 3.5875 - 6.3863i |
| x[32]=-0.7887 | X(4)= 2.4889 + 3.4334i |
| x[40]=-0.1198 | X(5)= -6.9437 - 4.6805i |
| x[48]=-0.5186 | X(6)= -5.5614 + 0.5575i |
| x[56]= 0.3357 | X(7)= 1.7772 + 2.2365i |
| | |
| x[1]= -0.3122 | X(8)= -2.4585 - 4.7305i |
| x[9]= -0.9550 | X(9)= 1.0650 + 2.8321i |
| x[17]=-0.0582 | X(10)= 3.5596 + 1.1961i |
| x[25]= 0.3089 | X(11)= 6.4955 + 3.6918i |
| x[33]= 0.2219 | X(12)=-2.7702 + 0.6047i |
| x[41]= 0.0543 | X(13)= 3.8098 + 1.9517i |
| x[49]= 0.3522 | X(14)= 0.7528 - 6.0029i |
| x[57]= 0.6888 | X(15)= 2.9978 + 0.5043i |
| | |
| x[2]= 0.1681 | X(16)= 1.6813 + 0.0899i |
| x[10]=-0.1495 | X(17)= 1.3509 + 0.6740i |
| x[18]= 0.3919 | X(18)=-2.1556 + 0.9694i |
| x[26]=-0.1848 | X(19)= 0.2590 + 0.9829i |
| x[34]= 0.5576 | X(20)=-4.0221 - 0.9514i |
| x[42]=-0.0852 | X(21)=-2.5312 - 0.5286i |
| x[50]=-0.4219 | X(22)=-3.4193 + 3.4465i |
| x[58]=-0.3111 | X(23)=-1.6570 + 0.4295i |
| | |
| x[3]= -07845 | X(24)= 0.4753 - 2.7567i |
| x[11]=-0.3746 | X(25)= 0.6678 - 3.5513i |
| x[19]= 0.3998 | X(26)= 3.2590 - 1.8220i |
| x[27]= 0.6400 | X(27)= 1.7467 + 1.4740i |
| x[35]=-0.1531 | X(28)=-0.1609 + 3.9376i |
| x[43]= 0.7507 | X(29)= 0.2749 - 2.5098i |
| x[51]= 0.3436 | X(30)= 2.4537 + 1.3847i |
| x[59]= 0.5610 | X(31)= 6.9724 + 5.9154i |
| | |
| x[4]= 0.8126 | X(32)= 1.4950 + 0.0000i |
| x[12]=-0.6770 | X(33)= 6.9724 - 5.9154i |
| x[20]= 0.2771 | X(34)= 2.4537 - 1.3847i |
| x[28]= 0.4367 | X(35)= 0.2749 + 2.5098i |
| x[36]=-0.8184 | X(36)=-0.1609 - 3.9376i |
| x[44]= 0.0361 | X(37)= 1.7467 - 1.4740i |
| x[52]= 0.3903 | X(38)= 3.2590 + 1.8220i |
| x[60]= 0.3507 | X(39)= 0.6678 + 3.5513i |
| | |
| x[5]= 0.7593 | X(40)= 0.4753 + 2.7567i |
| x[13]=-0.6425 | X(41)=-1.6570 - 0.4295i |
| x[21]=-0.9328 | X(42)=-3.4193 - 3.4465i |
| x[29]= 0.9373 | X(43)=-2.5312 + 0.5286i |
| x[37]=-0.4671 | X(44)=-4.0221 + 0.9514i |
| x[45]= 0.8872 | X(45)= 0.2590 - 0.9829i |
| x[53]=-0.8640 | X(46)=-2.1556 - 0.9694i |
| x[61]=-0.9866 | X(47)= 1.3509 - 0.6740i |
| | |
| x[6]= 0.6355 | X(48)= 1.6813 - 0.0899i |
| x[14]=-0.1542 | X(49)= 2.9978 - 0.5043i |
| x[22]=-0.8624 | X(50)= 0.7528 + 6.0029i |
| x[30]= 0.0627 | X(51)= 3.8098 - 1.9517i |
| x[38]=-0.6927 | X(52)=-2.7702 - 0.6047i |
| x[46]= 0.2754 | X(53)= 6.4955 - 3.6918i |
| x[54]=-0.4904 | X(54)= 3.5596 - 1.1961i |
| x[62]= 0.2043 | X(55)= 1.0650 - 2.8321i |
| | |
| x[7]= -0.4785 | X(56)=-2.4585 + 4.7305i |
| x[15]=-0.8115 | X(57)= 1.7772 - 2.2365i |
| x[23]=-0.3608 | X(58)=-5.5614 - 0.5575i |
| x[31]=-0.3497 | X(59)=-6.9437 + 4.6805i |
| x[39]=-0.4380 | X(60)= 2.4889 - 3.4334i |
| x[47]= 0.9154 | X(61)= 3.5875 + 6.3863i |
| x[55]=-0.5519 | X(62)= 1.0585 - 1.6702i |
| x[63]=-0.2265 | X(63)= 0.1146 - 3.2110i |

MatLab을 사용하여 위의 입력 벡터를 그림 5의 구조에 입력시켜서 표 5의 오른쪽과 같은 FFT 출력 $X(0) \sim X(63)$의 출력을 얻었다. 이 출력 벡터가 FFT 함수를 사용하여 얻은 출력 테스트 벡터와 일치함을 확인하였다. 따라서 그림 3과 4의 제안된 나비연산기 구조가 FFT 변환을 올바르게 수행함을 확인하였다.

## 5. 결론

이 논문에서는 DIT Radix-8 FFT 알고리즘의 구현을 위한 새로운 나비연산기 구조를 제안하고, 그 구조의 동작을 확인하였다. DIF 구조와 비교하여 DIT 구조의 장점은 FFT 출력이 순차적으로 출력되는 것과, Radix-8 알고리즘의 장점은 스테이지 수가 적아서 지연시간이 적다는 것이다. OFDM 통신용 SoC에 사용되는 FFT 블록은 연산후 병렬 출력이 순차적인 직렬 출력으로 변환되어야 하므로 순차적인 출력이 나오는 DIT 구조가 효용성이 높다. 따라서 제안된 DIT Radix-8 FFT 구조는 순차적인 FFT 출력을 필요로 하고, 연산지연시간이 짧아야하는 고속 OFDM 통신용 SoC에 사용될 수 있다.

## References


[1] 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.
DOI: http://dx.doi.org/10.1109/92.661241

[2] M. Bekooij, J. Huisken, and K. Nowak, “Numerical accuracy of Fast Fourier Transforms with CORDIC arithmetic", Journal of VLSI Signal Processing, vol. 25, No. 2, pp. 187-193, Jun. 2000.
DOI: http://dx.doi.org/10.1023/A:1008179225059

[3] J. Lee and H. Lee, "A high-Speed 2-Parallel Radix-24 FFT/IFFT processor for MB-OFDM UWB Systems", IEICE Trans. on Fundamentals, vol. E91-A, No. 4, pp. 1206- 1211, April, 2008.
DOI: http://dx.doi.org/10.1093/ietfec/e91-a.4.1206

[4] H. J. Kim and Y. B. Jang, "Low-area FFT processor structure using radix-42 algorithm", Journal of IEEK, vol. 49-SD, No. 3, pp. 8-14, Mar. 2012.

[5] In-Gul Jang and Jin-Gyun Chung, "Low-power FFT design for NC-OFDM in cognitive radio systems", Journal of IEEK, vol. 48-TC, No. 6, pp. 28-33, Jun. 2011.

[6] Eun Ji Kim and Myung Hoon Sunwoo, "High speed 8-parallel FFT/IFFT processor using efficient pipeline architecture and scheduling scheme", Journal of KICS, vol. 36, No. 3, pp. 175-182, Mar. 2011.
DOI: http://dx.doi.org/10.7840/kics.2011.36c.3.175

[7] Y. B. Jang, E. S. Hur, J. S. Park and D. K. Hong, "High-speed Radix-8 FFT Structure for OFDM", Journal of IEEK, vol. 44-SP, No. 5, pp. 84-93, May, 2007.

[8] Young Beom Jang and Sang Woo Lee, "Low-power butterfly structure for DIT Radix-4 FFT implementation", Journal of KICS, vol. 38A, No. 12, pp. 1145-1147, Dec. 2013.
DOI: http://dx.doi.org/10.7840/kics.2013.38a.12.1145

[9] Young Beom Jang and Sang Woo Lee, "A New DIT Radix-4 FFT Structure and Implementation", Journal of the Korea Academia-Industrial cooperation Society, vol. 16, No. 1, pp. 683-690, Jan. 2015.
DOI: http://dx.doi.org/10.5762/KAIS.2015.16.1.683

[10] K. Rao, D. N. Kim, and J. J. Hwang, “Fast Fourier Transform - Algorithms and Applications” Springer, 2011.



**장 영 범(Young-Beom Jang)** **[정회원]**

- 1981년 2월 : 연세대학교 전기공학과 (공학사)
- 1990년 1월 : Polytechnic University 전기공학과 (MS)
- 1994년 1월 : Polytechnic University 전기공학과 (Ph.D.)
- 1983년 7월 ~ 1999년 12월 : 삼성전자 수석연구원
- 2002년 9월 ~ 현재 : 상명대학교 정보통신공학과 교수

<관심분야>
통신신호처리, 반도체설계, 오디오/비디오신호처리