논문 2007-44SP-5-10

OFDM용 고속 Radix-8 FFT 구조

(High-speed Radix-8 FFT Structure for OFDM)

장 영 범*, 허 은 성**, 박 진 수**, 홍 대 기*

(Young-Beom Jang, Eun-Sung Hur, Jin-Su Prak, and Dae-Ki Hong)

요 약

이 논문에서는 고속 FFT 구현을 위한 Radix-8 구조를 제안한다. 제안 FFT 구조의 핵심은 Radix-8 DIF(Decimation In Frequency) 나비연산기 구조이다. Radix-8 알고리즘은 고속처리는 가능하나 구현면적이 증가하는 단점이 있는데, 제안 구조는 곱셈연산을 DA(Distributed Arithmetic) 방식을 사용하여 구현함으로써 구현 면적이 증가하는 것을 줄일 수 있었다. 64-point FFT에 대하여 기존의 Radix-4 나비연산기와 제안된 Radix-8 나비연산기를 각각 사용하여 구현한 결과 구현면적이 49.2%가 증가하였다. 즉, Throughput을 2배로 증가시키기 위하여 하드웨어는 49.2%만 증가함을 Verilog-HDL 코딩을 통하여 확인하였다. 또한 기존 구조와 제안 구조가 같은 Throughput을 얻는 경우에는 전력소모가 25.4%가 감소하게 된다. 따라서 제안된 나비연산기를 사용하는 FFT 구조는 고속/저전력 FFT를 필요로하는 OFDM용 통신단말기에 사용될 수 있다.

Abstract

In this paper, a Radix-8 structure for high-speed FFT is propose. Main block of the proposed FFT structure is Radix-8 DIF(Decimation In Frequency) butterfly. Even throughput of the Radix-8 FFT is twice than that of the Radix-4 FFT, implementation area of the Radix-8 is larger than that of Radix-4 FFT. But, implementation area of the proposed Radix-8 FFT was reduced by using DA(Distributed Arithmetic) for multiplication. For comparison, the 64-point FFT was implemented using conventional Radix-4 butterfly and proposed Radix-8 butterfly, respectively. The Verilog-HDL coding results for the proposed FFT structure show 49.2% cell area increment comparison with those of the conventional Radix-4 FFT structure. Namely, to speed up twice, 49.2% of area cost is required. In case of same throughput, power consumption of the proposed structure is reduced by 25.4%. Due to its efficient processing scheme, the proposed FFT structure can be used in large size of FFT like OFDM Modem.

Keywords: Fast Fourier Transform, Radix-8, Distributed Arithmetic, Twiddle factor

I. 서 론

OFDM(OrthogonalFrequencyDivisionMultiplexing) 방식을 사용하는 통신 단말기의 상용화 속도가 빨라짐에 따라 MODEM SoC(System on a Chip)의 고속/저전력 구현에 대한 연구가 활발히 진행되고 있다. OFDM용 MODEM SoC는 크게 동기화 블록, FFT(Fast Fourier Transform) 블록, 등화기 블록, 비터비 블록, 변복조 블록 등으로 구성된다. OFDM 전송방식은 직렬로 입력되는 데이터 열을 병렬 데이터 열로 변환한 후에 부반송파에 실어 전송하는 방식이다. 이와 같은 병렬화와 부반송파를 곱하는 동작은 IFFT와 FFT로 구현이 가능한데, OFDM 방식을 사용하는 DMB 표준에서는 2048 point의 큰 크기의 FFT를 필요로 하므로 FFT 블록의 구현 비용과 전력소모를 줄이는 것이 핵심사항이라고 할 수 있다. 지상파 DMB용 OFDM에서는 246 μs 동안에 2048 point FFT를 수행하여야 하므로 높은 처리율을 갖는 FFT가 요구된다. FFT 구조로는 단일버터플라이연산자 구조[1], 파이프라인 구조[2], 병렬구조[3] 등의 여러 구조가 제안되었다. 파이프라인 구조에서 각

* 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:2007년4월12일, 수정완료일:2007년8월24일

스테이지마다 나비연산기가 사용되는데, Radix-2와 Radix-4의 나비연산기가 주로 사용된다. Radix-2 DIF(Decimation-In-Frequency) 나비연산기는 입력신호를 짝수와 홀수 번째로 나누어 처리하므로 구조는 간단하나 속도가 느린 단점이 있다.[4] 또한, Radix-4 DIF (Decimation-In-Frequency) 나비연산기는 입력신호를 4의 배수끼리 나누어 처리하므로 Radix-2의 구조와 비교하여 처리속도는 빠르나 1, j, -1, $-j$ 등과 트위들 factor가 결합하여 구조가 복잡해진다.[5]-[6] 본 논문에서는 OFDM의 고속 통신단말기에 사용할 수 있는 고속의 Radix-8 DIF 나비연산기 구조를 제안한다. Radix-8 FFT 구조는 한번에 8개의 입력을 처리하므로 처리속도는 향상되나 구현 구조가 매우 복잡해진다. 그러나 제안된 Radix-8 구조는 DA(Distributed Arithmetic) 방식을 사용하여 기존의 곱셈기를 사용하는 구조보다 구현 면적을 줄일 수 있었다.[7]

본문 Ⅱ장에서는 Radix-8 알고리즘에 대하여 살피고, Ⅲ장에서 DA 연산을 적용한 저 전력 Radix-8 FFT 구조를 제안하였다. Ⅳ장에서는 제안된 구조를 Verilog HDL 코딩으로 구현하여 처리속도와 게이트 수를 기존 구조들과 비교하여 제안 구조의 효율성을 분석하였다.

Ⅱ. Radix-8 FFT 알고리즘

2.1 Radix-8 DIF의 FFT 알고리즘

N-Point의 DFT는 다음과 같이 표현된다.

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

식 (1)을 Radix-8 DIF(Decimation-In-Frequency) 알고리즘을 사용하기 위하여 8의 배수끼리 분리하면 다음과 같다.

$$\begin{aligned}X(8r)=\sum_{n=0}^{N/8-1}&[x(n)+x(n+\tfrac{N}{8})+x(n+\tfrac{2N}{8})\\&+x(n+\tfrac{3N}{8})+x(n+\tfrac{4N}{8})+x(n+\tfrac{5N}{8})\\&+x(n+\tfrac{6N}{8})+x(n+\tfrac{7N}{8})]\,W_{N/8}^{nr}\end{aligned}$$

$$\begin{aligned}X(8r+1)=\sum_{n=0}^{N/8-1}\Big\{&[x(n)+(k-jk)x(n+\tfrac{N}{8})\\&-jx(n+\tfrac{2N}{8})+(-k-jk)x(n+\tfrac{3N}{8})\\&-x(n+\tfrac{4N}{8})+(-k+jk)x(n+\tfrac{5N}{8})\\&+jx(n+\tfrac{6N}{8})+(k+jk)x(n+\tfrac{7N}{8})]\,W_N^{n}\Big\}W_{N/8}^{nr}\end{aligned}$$

$$\begin{aligned}X(8r+2)=\sum_{n=0}^{N/8-1}\Big\{&[x(n)-jx(n+\tfrac{N}{8})-x(n+\tfrac{2N}{8})\\&+jx(n+\tfrac{3N}{8})+x(n+\tfrac{4N}{8})-jx(n+\tfrac{5N}{8})\\&-x(n+\tfrac{6N}{8})+jx(n+\tfrac{7N}{8})]\,W_N^{2n}\Big\}W_{N/8}^{nr}\end{aligned}$$

$$\begin{aligned}X(8r+3)=\sum_{n=0}^{N/8-1}\Big\{&[x(n)+(-k-jk)x(n+\tfrac{N}{8})+jx(n+\tfrac{2N}{8})\\&+(k-jk)x(n+\tfrac{3N}{8})-x(n+\tfrac{4N}{8})\\&+(k+jk)x(n+\tfrac{5N}{8})-jx(n+\tfrac{6N}{8})\\&+(-k+jk)x(n+\tfrac{7N}{8})]\,W_N^{3n}\Big\}W_{N/8}^{nr}\end{aligned}$$

$$\begin{aligned}X(8r+4)=\sum_{n=0}^{N/8-1}\Big\{&[x(n)-x(n+\tfrac{N}{8})+x(n+\tfrac{2N}{8})\\&-x(n+\tfrac{3N}{8})+x(n+\tfrac{4N}{8})-x(n+\tfrac{5N}{8})\\&+x(n+\tfrac{6N}{8})-x(n+\tfrac{7N}{8})]\,W_N^{4n}\Big\}W_{N/8}^{nr}\end{aligned} \tag{2}$$

$$\begin{aligned}X(8r+5)=\sum_{n=0}^{N/8-1}\Big\{&[x(n)+(-k+jk)x(n+\tfrac{N}{8})-jx(n+\tfrac{2N}{8})\\&+(k+jk)x(n+\tfrac{3N}{8})-x(n+\tfrac{4N}{8})\\&+(k-jk)x(n+\tfrac{5N}{8})+jx(n+\tfrac{6N}{8})\\&+(-k-jk)x(n+\tfrac{7N}{8})\,W_N^{6n}\Big\}W_{N/8}^{nr}\end{aligned}$$

$$\begin{aligned}X(8r+6)=\sum_{n=0}^{N/8-1}\Big\{&[x(n)+jx(n+\tfrac{N}{8})-x(n+\tfrac{2N}{8})\\&-jx(n+\tfrac{3N}{8})+x(n+\tfrac{4N}{8})+jx(n+\tfrac{5N}{8})\\&-x(n+\tfrac{6N}{8})-jx(n+\tfrac{7N}{8})]\,W_N^{6n}\Big\}W_{N/8}^{nr}\end{aligned}$$

$$\begin{aligned}X(8r+7)=\sum_{n=0}^{N/8-1}\Big\{&[x(n)+(k+jk)x(n+\tfrac{N}{8})+(j)x(n+\tfrac{2N}{8})\\&+(-k+jk)x(n+\tfrac{3N}{8})+(-1)x(n+\tfrac{4N}{8})\\&+(-k-jk)x(n+\tfrac{5N}{8})+(-j)x(n+\tfrac{6N}{8})\\&+(k-jk)x(n+\tfrac{7N}{8})]\,W_N^{7n}\Big\}W_{N/8}^{nr}\end{aligned}$$

위의 식 (2) 에서 $k=0.7071068$이다. 이 식을 살펴보면 N-point의 DFT는 Radix-8 DIF에 의하여 8개의 다른 $N/8$-point DFT로 분해될 수 있음을 알 수 있다. 또한 각각의 다른 $N/8$-Point DFT로 나누어 수행하기 위해서는 먼저 다음의 식들이 계산되어야 한다.

$$\begin{aligned}x_0(n)=&[x(n)+x(n+\tfrac{N}{8})+x(n+\tfrac{2N}{8})\\&+x(n+\tfrac{3N}{8})+x(n+\tfrac{4N}{8})+x(n+\tfrac{5N}{8})\\&+x(n+\tfrac{6N}{8})+x(n+\tfrac{7N}{8})]\end{aligned}$$

$$
\begin{aligned}
x_1(n) &= [x(n)+(k-jk)x(n+\frac{N}{8})-jx(n+\frac{2N}{8}) \\
&\quad +(-k-jk)x(n+\frac{3N}{8})-x(n+\frac{4N}{8}) \\
&\quad +(-k+jk)x(n+\frac{5N}{8})+jx(n+\frac{6N}{8}) \\
&\quad +(k+jk)x(n+\frac{7N}{8})]\,W_N^n \\
x_2(n) &= [x(n)-jx(n+\frac{N}{8})-x(n+\frac{2N}{8}) \\
&\quad +jx(n+\frac{3N}{8})+x(n+\frac{4N}{8})-jx(n+\frac{5N}{8}) \\
&\quad -x(n+\frac{6N}{8})+jx(n+\frac{7N}{8})]\,W_N^{2n} \\
x_3(n) &= [x(n)+(-k-jk)x(n+\frac{N}{8})+jx(n+\frac{2N}{8}) \\
&\quad +(k-jk)x(n+\frac{3N}{8})-x(n+\frac{4N}{8}) \\
&\quad +(k+jk)x(n+\frac{5N}{8})-jx(n+\frac{6N}{8}) \\
&\quad +(-k+jk)x(n+\frac{7N}{8})]\,W_N^{3n} \\
x_4(n) &= [x(n)-x(n+\frac{N}{8})+x(n+\frac{2N}{8}) \\
&\quad -x(n+\frac{3N}{8})+x(n+\frac{4N}{8})-x(n+\frac{5N}{8}) \\
&\quad +x(n+\frac{6N}{8})-x(n+\frac{7N}{8})]\,W_N^{4n} \\
x_5(n) &= [x(n)+(-k+jk)x(n+\frac{N}{8})-jx(n+\frac{2N}{8}) \\
&\quad +(k+jk)x(n+\frac{3N}{8})-x(n+\frac{4N}{8}) \\
&\quad +(k-jk)x(n+\frac{5N}{8})+jx(n+\frac{6N}{8}) \\
&\quad +(-k-jk)x(n+\frac{7N}{8})\,W_N^{5n} \\
x_6(n) &= [x(n)+jx(n+\frac{N}{8})-x(n+\frac{2N}{8}) \\
&\quad -jx(n+\frac{3N}{8})+x(n+\frac{4N}{8})+jx(n+\frac{5N}{8}) \\
&\quad -x(n+\frac{6N}{8})-jx(n+\frac{7N}{8})]\,W_N^{6n} \\
x_7(n) &= [x(n)+(k+jk)x(n+\frac{N}{8})+(j)x(n+\frac{2N}{8}) \\
&\quad +(-k+jk)x(n+\frac{3N}{8})+(-1)x(n+\frac{4N}{8}) \\
&\quad +(-k-jk)x(n+\frac{5N}{8})+(-j)x(n+\frac{6N}{8}) \\
&\quad +(k-jk)x(n+\frac{7N}{8})]\,W_N^{7n}
\end{aligned}
\tag{3}
$$

식 (3)은 Radix-8 나비연산이라 부르고, 덧셈과 복소 곱셈을 요구한다. 식 (3)에서 입력과 출력 값들은 모두 복소수이므로 다음 식과 같이 실수부와 허수부로 나누어 표기하는 것이 계산을 위해 편리하다.

$$
\begin{aligned}
x(n) &= x_a + jy_a, & x_0(n) &= x_a' + jy_a' \\
x(n+\frac{N}{8}) &= x_b + jy_b, & x_1(n) &= x_b' + jy_b' \\
x(n+\frac{2N}{8}) &= x_c + jy_c, & x_2(n) &= x_c' + jy_c' \\
x(n+\frac{3N}{8}) &= x_d + jy_d, & x_3(n) &= x_d' + jy_d' \\
x(n+\frac{4N}{8}) &= x_e + jy_e, & x_4(n) &= x_e' + jy_e' \\
x(n+\frac{5N}{8}) &= x_f + jy_f, & x_5(n) &= x_f' + jy_f' \\
x(n+\frac{6N}{8}) &= x_g + jy_g, & x_6(n) &= x_g' + jy_g' \\
x(n+\frac{7N}{8}) &= x_h + jy_h, & x_7(n) &= x_h' + jy_h'
\end{aligned}
\tag{4}
$$

트위들 factor값 역시 복소수이므로 다음과 같이 나타내는 것이 편리하다.

$$
\begin{aligned}
W_N^{\frac{N}{8}} &= \cos(\frac{\pi}{4}) - j\sin(\frac{\pi}{4}) = C_b + j(-S_b) \\
W_N^{\frac{2N}{8}} &= \cos(\frac{2\pi}{4}) - j\sin(\frac{2\pi}{4}) = C_c + j(-S_c) \\
W_N^{\frac{3N}{8}} &= \cos(\frac{3\pi}{4}) - j\sin(\frac{3\pi}{4}) = C_d + j(-S_d) \\
W_N^{\frac{4N}{8}} &= \cos(\frac{4\pi}{4}) - j\sin(\frac{4\pi}{4}) = C_e + j(-S_e) \\
W_N^{\frac{5N}{8}} &= \cos(\frac{5\pi}{4}) - j\sin(\frac{5\pi}{4}) = C_f + j(-S_f) \\
W_N^{\frac{6N}{8}} &= \cos(\frac{6\pi}{4}) - j\sin(\frac{6\pi}{4}) = C_g + j(-S_g) \\
W_N^{\frac{7N}{8}} &= \cos(\frac{7\pi}{4}) - j\sin(\frac{7\pi}{4}) = C_h + j(-S_h)
\end{aligned}
\tag{5}
$$

식 (4)와 식 (5)의 복소수 값을 사용하여 Radix-8 나비 연산기를 나타내면 그림 1과 같다.

그림 1의 나비연산기를 효과적으로 수행하기 위하여 나비연산기 출력을 입력 복소수와 트위들을 사용하여 나타내어야 한다. 먼저 식 (4)와 식 (5)를 식 (3)에 대입하면 첫 번째 나비연산기의 출력 값 x_a'와 y_a'는 다음과 같이 연산할 수 있다.

$$
\begin{aligned}
x_a' &= x_a + x_b + x_c + x_d + x_e + x_f + x_g + x_h \\
y_a' &= y_a + y_b + y_c + y_d + y_e + y_f + y_g + y_h
\end{aligned}
\tag{6}
$$

식 (6)에서 보듯이, 나비연산기의 첫 출력 값에는 트위들 factor가 1이므로 곱셈연산이 필요하지 않다. 그러나 그림 1의 나비연산의 두 번째 출력 값을 구하는 과정부터 트위들 factor 복소수를 곱하는 연산이 포함된다. 역시 식 (4)와 식 (5)를 식 (3)에 대입하면 나비연산기의 두 번째 출력 값은 다음과 같이 나타낼 수 있다.

$$
\begin{aligned}
& x_b' + jy_b' \\
&= [(x_a + jy_a) + (k-jk)(x_b + jy_b) - j(x_c + jy_c) \\
&\quad +(-k-jk)(x_d + jy_d) - (x_e + jy_e) + (-k+jk)(x_f + jy_f) \\
&\quad +j(x_g + jy_g) + (k+jk)(x_h + jy_h)][C_b + j(-S_b)]\}
\end{aligned}
\tag{7}
$$

식 (7)로부터 x_b'와 y_b'는 다음과 같이 나타낼 수 있다.

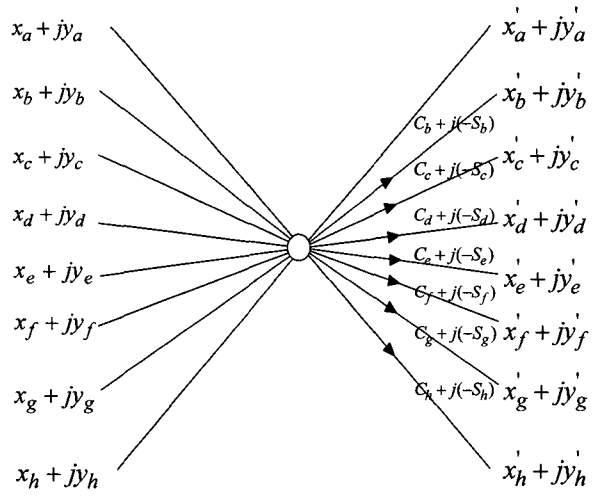

그림 1. 복소수를 사용한 Radix-8 나비연산기 블록도

Fig. 1. Radix-8 butterfly block diagram represented by complex number.

$$
\begin{aligned}
x_b' &= [(x_a + y_c - x_e - y_g) \\
&\quad + k(x_b + y_b - x_d + y_d - x_f - y_f + x_h - y_h)]C_b \\
&\quad + [(y_a - x_c - y_e + x_g) \\
&\quad + k(y_b - x_b - y_d - x_d - y_f + x_f + y_h + x_h)]S_b \\
&= (x_{1a} + kx_{1b})C_b + (x_{2a} + kx_{2b})S_b \\
&= x_1 C_b + x_2 S_b \\
y_b' &= [(y_a - x_c - y_e + x_g) \\
&\quad + k(y_b - x_b - y_d - x_d - y_f + x_f + y_h + x_h)]C_b \\
&\quad - [(x_a + y_c - x_e - y_g) \\
&\quad + k(x_b + y_b - x_d + y_d - x_f - y_f + x_h - y_h)]S_b \\
&= (x_{2a} + kx_{2b})C_b - (x_{1a} + kx_{1b})S_b \\
&= x_2 C_b - x_1 S_b
\end{aligned} \tag{8}
$$

다음으로 세 번째 나비연산기 출력 값을 구하면 다음과 같다.

$$
\begin{aligned}
&x_c' + jy_c' \\
&= [(x_a + jy_a) - j(x_b + jy_b) - (x_c + jy_c) + j(x_d + jy_d) \\
&\quad + (x_e + jy_e) - j(x_f + jy_f) - (x_g + jy_g) + j(x_h + jy_h)] \\
&\quad [C_c + j(-S_c)] \\
&= [(x_a + y_b - x_c - y_d + x_e + y_f - x_g - y_h) \\
&\quad + j(y_a - x_b - y_c + x_d + y_e - x_f - y_g + x_h)][C_c + j(-S_c)]
\end{aligned} \tag{9}
$$

식 (9)로부터 x_c'와 y_c'는 다음과 같이 나타낼 수 있다.

$$
\begin{aligned}
x_c' &= [(x_a + y_b - x_c - y_d + x_e + y_f - x_g - y_h)C_c \\
&\quad + (y_a - x_b - y_c + x_d + y_e - x_f - y_g + x_h)S_c] \\
&= x_3 C_c + x_4 S_c \\
y_c' &= [(y_a - x_b - y_c + x_d + y_e - x_f - y_g + x_h)C_c \\
&\quad - (x_a + y_b - x_c - y_d + x_e + y_f - x_g - y_h)S_c] \\
&= x_4 C_c - x_3 S_c
\end{aligned} \tag{10}
$$

다음으로 네 번째 출력 값을 구하면 다음과 같다.

$$
\begin{aligned}
&x_d' + jy_d' \\
&= [(x_a + jy_a) + (-k - jk)(x_b + jy_b) + j(x_c + jy_c) \\
&\quad + (k - jk)(x_d + jy_d) - (x_e + jy_e) + (k + jk)(x_f + jy_f) \\
&\quad - j(x_g + jy_g) + (-k + jk)(x_h + jy_h)][C_d + j(-S_d)] \\
&= [(x_a - kx_b + ky_b - y_c + kx_d + ky_d \\
&\quad - x_e + kx_f - ky_f + y_g - kx_h - ky_h) \\
&\quad + j(y_a - ky_b - kx_b + x_c - kx_d + ky_d \\
&\quad - y_e + ky_f + kx_f - x_g - ky_h + kx_h)][C_d + j(-S_d)]
\end{aligned} \tag{11}
$$

식 (11)로부터 x_d'와 y_d'는 다음과 같이 나타낼 수 있다.

$$
\begin{aligned}
x_d' &= [(x_a - y_c - x_e + y_g) \\
&\quad + k(-x_b + y_b + x_d + y_d + x_f - y_f - x_h - y_h)]C_d \\
&\quad + [(y_a + x_c - y_e - x_g) \\
&\quad + k(-y_b - x_b + y_d - x_d + y_f + x_f - y_h + x_h)]S_d \\
&= (x_{5a} + kx_{5b})C_d + (x_{6a} + kx_{6b})S_d \\
&= x_5 C_d + x_6 S_d \\
y_d' &= [(y_a + x_c - y_e - x_g) \\
&\quad + k(-y_b - x_b + y_d - x_d + y_f + x_f - y_h + x_h)]C_d \\
&\quad - [(x_a - y_c - x_e + y_g) \\
&\quad + k(-x_b + y_b + x_d + y_d + x_f - y_f - x_h - y_h)]S_d \\
&= (x_{6a} + kx_{6b})C_d - (x_{5a} + kx_{5b})S_d \\
&= x_6 C_d - x_5 S_d
\end{aligned} \tag{12}
$$

다음으로 다섯 번째 출력 값을 구하면 다음과 같다.

$$
\begin{aligned}
&x_e' + jy_e' \\
&= [(x_a + jy_a) - (x_b + jy_b) + (x_c + jy_c) - (x_d + jy_d) \\
&\quad + (x_e + jy_e) - (x_f + jy_f) + (x_g + jy_g) - (x_h + jy_h)] \\
&\quad [C_e + j(-S_e)] \\
&= [(x_a - x_b + x_c - x_d + x_e - x_f + x_g - x_h) \\
&\quad + j(y_a - y_b + y_c - y_d + y_e - y_f + y_g - y_h)][C_e - jS_e]
\end{aligned} \tag{13}
$$

식 (13)로부터 x_e'와 y_e'는 다음과 같이 나타낼 수 있다.

$$
\begin{aligned}
x_e' &= [(x_a - x_b + x_c - x_d + x_e - x_f + x_g - x_h)C_e \\
&\quad + (y_a - y_b + y_c - y_d + y_e - y_f + y_g - y_h)S_e] \\
&= x_7 C_e + x_8 S_e \\
y_e' &= [(y_a - y_b + y_c - y_d + y_e - y_f + y_g - y_h)C_e \\
&\quad - (x_a - x_b + x_c - x_d + x_e - x_f + x_g - x_h)S_e] \\
&= x_8 C_e - x_7 S_e
\end{aligned} \tag{14}
$$

다음으로 여섯 번째 출력 값을 구하면 다음과 같다.

$$
\begin{aligned}
&x_f' + jy_f' \\
&= [(x_a + jy_a) + (-k + jk)(x_b + jy_b) - j(x_c + jy_c) \\
&\quad + (k + jk)(x_d + jy_d) - (x_e + jy_e) + (k - jk)(x_f + jy_f) \\
&\quad + j(x_g + jy_g) + (-k - jk)(x_h + jy_h)][C_f + j(-S_f)] \\
&= [(x_a - kx_b - ky_b + y_c + kx_d - ky_d \\
&\quad - x_e + kx_f + ky_f - y_g - kx_h + ky_h) \\
&\quad + j(y_a - ky_b + kx_b - x_c + kx_d + ky_d \\
&\quad - y_e + ky_f - kx_f + x_g - ky_h - kx_h)][C_f + j(-S_f)]
\end{aligned} \tag{15}
$$

식 (15)로부터 x_f' 와 y_f' 는 다음과 같이 나타낼 수 있다.

$$
\begin{aligned}
x_f' &= [(x_a + y_c - x_e - y_g) \\
&\quad + k(-x_b - y_b + x_d - y_d + x_f + y_f - x_h + y_h)]C_f \\
&\quad + [(y_a - x_c - y_e + x_g) \\
&\quad + k(-y_b + x_b + y_d + x_d + y_f - x_f - y_h - x_h)]S_f \\
&= (x_{9a} + kx_{9b})C_f + (x_{10a} + kx_{10b})S_f \\
&= x_9 C_f + x_{10} S_f
\end{aligned}
\tag{16}
$$

$$
\begin{aligned}
y_f' &= [(y_a - x_c - y_e + x_g) \\
&\quad + k(-y_b + x_b + y_d + x_d + y_f - x_f - y_h - x_h)]C_f \\
&\quad - [(x_a + y_c - x_e - y_g) \\
&\quad + k(-x_b - y_b + x_d - y_d + x_f + y_f - x_h + y_h)]S_f \\
&= (x_{10a} + kx_{10b})C_f - (x_{9a} + kx_{9b})S_f \\
&= x_{10} C_f - x_9 S_f
\end{aligned}
$$

다음으로 일곱 번째 출력 값을 구하면 다음과 같다.

$$
\begin{aligned}
&x_g' + jy_g' \\
&= [(x_a + jy_a) + j(x_b + jy_b) - (x_c + jy_c) - j(x_d + jy_d) \\
&\quad + (x_e + jy_e) + j(x_f + jy_f) - (x_g + jy_g) - j(x_h + jy_h)] \\
&\quad [C_g + j(-S_g)] \\
&= [(x_a - y_b - x_c + y_d + x_e - y_f - x_g + y_h) \\
&\quad + j(y_a + x_b - y_c - x_d + y_e + x_f - y_g - x_h)][C_g + j(-S_g)]
\end{aligned}
\tag{17}
$$

식 (17)로부터 x_g'와 y_g'는 다음과 같이 나타낼 수 있다.

$$
\begin{aligned}
x_g' &= [(x_a - y_b - x_c + y_d + x_e - y_f - x_g + y_h)C_g \\
&\quad + (y_a + x_b - y_c - x_d + y_e + x_f - y_g - x_h)S_g] \\
&= x_{11} C_g + x_{12} S_g \\
y_g' &= [(y_a + x_b - y_c - x_d + y_e + x_f - y_g - x_h)C_g \\
&\quad - (x_a - y_b - x_c + y_d + x_e - y_f - x_g + y_h)S_g] \\
&= x_{12} C_g - x_{11} S_g
\end{aligned}
\tag{18}
$$

마지막으로 여덟 번째 출력 값을 구하면 다음과 같다.

$$
\begin{aligned}
&x_h' + jy_h' \\
&= [(x_a + jy_a) + (k + jk)(x_b + jy_b) + j(x_c + jy_c) \\
&\quad + (-k + jk)(x_d + jy_d) - (x_e + jy_e) + (-k - jk)(x_f + jy_f) \\
&\quad - j(x_g + jy_g) + (k - jk)(x_h + jy_h)][C_h + j(-S_h)] \\
&= [(x_a + kx_b - ky_b - y_c - kx_d - ky_d \\
&\quad - x_e - kx_f + ky_f + y_g + kx_h + ky_h) \\
&\quad + j(y_a + ky_b + kx_b + x_c + kx_d - ky_d \\
&\quad - y_e - ky_f - kx_f - x_g + ky_h - kx_h)][C_h + j(-S_h)]
\end{aligned}
\tag{19}
$$

식 (19)로부터 x_h'와 y_h'는 다음과 같이 나타낼 수 있다.

$$
\begin{aligned}
x_h' &= [(x_a - y_c - x_e + y_g) \\
&\quad + k(x_b - y_b - x_d - y_d - x_f + y_f + x_h + y_h)]C_h \\
&\quad + [(y_a + x_c - y_e - x_g) \\
&\quad + k(y_b + x_b - y_d + x_d - y_f - x_f + y_h - x_h)]S_h \\
&= (x_{13a} + kx_{13b})C_h + (x_{14a} + kx_{14b})S_h \\
&= x_{13} C_h + x_{14} S_h \\
y_h' &= [(y_a + x_c - y_e - x_g) \\
&\quad + k(y_b + x_b - y_d + x_d - y_f - x_f + y_h - x_h)]C_h \\
&\quad - [(x_a - y_c - x_e + y_g) \\
&\quad + k(x_b - y_b - x_d - y_d - x_f + y_f + x_h + y_h)]S_h \\
&= (x_{14a} + kx_{14b})C_h - (x_{13a} + kx_{13b})S_h \\
&= x_{14} C_h - x_{13} S_h
\end{aligned}
\tag{20}
$$

나비연산기의 구현을 위하여, (6), (8), (10), (12), (14), (16), (18), (20)의 수식이 계산되어져야 한다.

III. 제안된 Radix-8 DIF 나비연산기 구조

3.1 제안 구조의 블록도

그림 1의 나비연산기에서 식 (6), (8), (10), (12), (14), (16), (18), (20)을 효과적으로 계산하여야 한다. 이 식들에서는 28개의 곱셈과 184개의 덧셈연산이 필요하므로 그림2 와 같은 덧셈 블록과 SA블록, DA블록, 트위들블록으로 구성된 나비연산구조를 제안한다.

제안구조 그림 2의 덧셈블록에서는 식 (6), (8), (10), (12), (14), (16), (18), (20)의 덧셈연산을 계산한다. Shift와 Add 연산을 수행하는 8개의 SA블록(Shift Add 블록)에서는 0.7071의 k 값을 곱하는 회로이다. DA블록은 DA(Distributed Arithmetic) 방식을 사용하여 곱셈연산을 수행하는 블록이다.[7] 곱셈기를 사용하면 구현면적이 커지므로 구현면적을 감소시키기 위하여 DA 방식을 사용하였다. 곱셈연산에 사용되는 트위들 factor는 트위들 블록에 저장하여 사용한다.

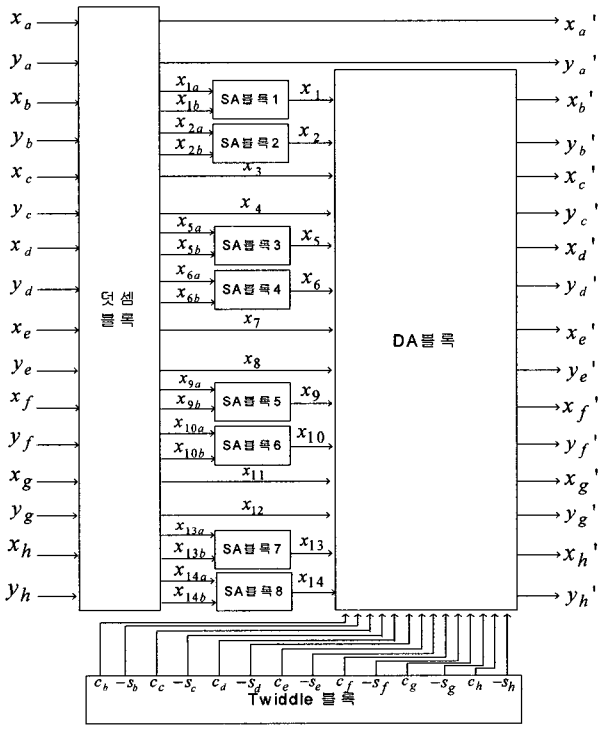

그림 2. 제안된 나비연산기 구조의 블록도
Fig. 2. Proposed block structure for butterfly.

3.2 덧셈 블록의 세부구조

그림 2의 덧셈블록에 대한 세부 구조는 그림 3과 같

다. 그림 3의 덧셈블록에서는 식 (6), (8), (10), (12), (14), (16), (18), (20)의 덧셈연산만을 수행하도록 설계하였다. 그림 3에서 보듯이, 덧셈블록에서는 총 136개의 덧셈연산을 수행한다. 덧셈블록에서 출력되는 16개의 출력 값 중에서 x_a'와 y_a'은 최종 출력이 되고, $x_3, x_4, x_7, x_8, x_{11}, x_{12}$는 DA블록으로 입력되도록 설계하였다.

마지막으로 $x_{1a}, x_{1b}, x_{2a}, x_{2b}, x_{5a}, x_{5b}, x_{6a}, x_{6b}, x_{9a}, x_{9b}, x_{10a}, x_{10b}, x_{13a}, x_{13b}, x_{14a}, x_{14b}$는 SA블록으로 입력되어 다음 계산에 사용되도록 설계하였다.

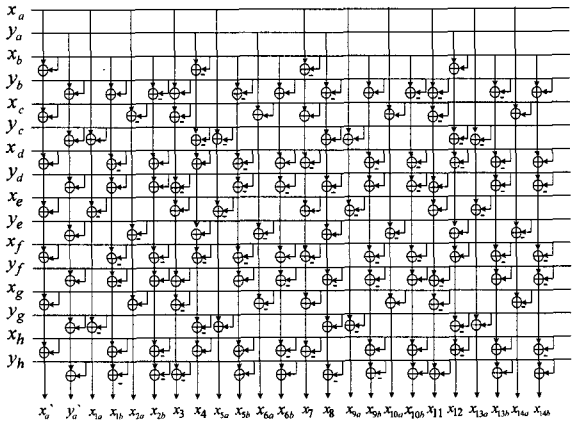

그림 3. 제안된 나비연산기 덧셈블록
Fig. 3. Proposed adder block structure for butterfly.

3.3 SA 블록의 세부구조

식 (8), (12), (16), (20)에는 각각 2개의 k를 곱하는 연산이 필요하다. 이와 같은 k에 대한 곱셈연산은 덧셈과 Shift만으로 수행할 수 있다. 곱셈을 덧셈과 Shift를 사용하여 구현하는 경우에는 Shift는 하드웨어로 구현할 때에는 비용이 거의 들지 않으므로 덧셈의 수가 곧 하드웨어 구현비용이 된다. k는 0.7071068이므로 16비트의 2의 보수형으로 나타내면 표 1과 같다.

표 1. k의 shift & adder형 표현
Table 1. shift & adder for k constant.

비트	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2의보수형	0	1	0	1	1	0	1	0	1	0	0	0	0	0	1	0

k를 곱하는 연산이 총 8개가 필요하므로 그림 2에서 보듯이 8개의 SA 블록을 사용하여 구현하였다. 표 1의 k를 사용하여 그림 2의 SA 블록1을 구현하면 다음 그림 4와 같다.

그림 4에서 보듯이, 각각의 SA블록에서는 6개의 덧셈기와 6개의 shift를 사용하였다. 나머지 7개의 SA블록들도 모두 그림 4와 같은 연산을 하기 때문에, SA블록에서는 총 48개의 덧셈기를 필요로 하게 된다. 이러한 연산과정을 거친 총 8개의 SA블록 출력 $x_1, x_2, x_5, x_9, x_{10}, x_{13}, x_{14}$는 그림 2의 DA블록으로 입력되도록 설계하였다.

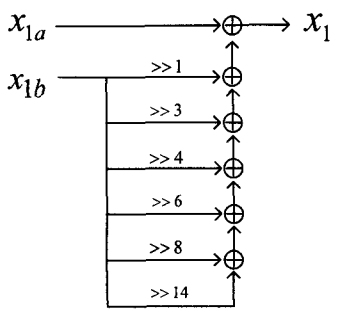

그림 4. SA 블록1의 세부구조
Fig. 4. SA block 1 structure.

3.4 DA 블록의 세부구조

덧셈블록과 SA블록을 거친 14개의 출력 값은 곱셈연산을 수행하기 위하여 DA블록으로 입력되도록 설계하였다. 제안된 DA 블록의 세부구조는 그림 5와 같다.

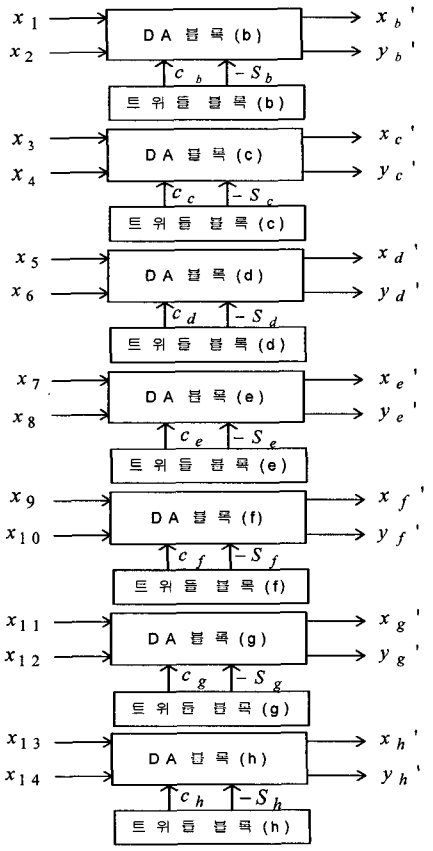

그림 5. 제안된 DA블록의 구조
Fig. 5. Proposed DA block structure.

그림 5에서 각각의 DA블록은 2탭의 곱셈연산을 수행한다. 예를 들면 DA 블록(b)에서 수행되는 곱셈연산은 다음과 같다.

$$x_b' = x_1 C_b + x_2 S_b \quad y_b' = x_2 C_b - x_1 S_b \qquad (21)$$

x_1과 x_2의 입력을 이용하여 x'_b와 y'_b 을 구하는 DA 블록(b)의 세부구조는 그림 6과 같다. 그림 6에서 보듯이 x_1, x_2는 4-input MUX의 어드레스로 사용된다.

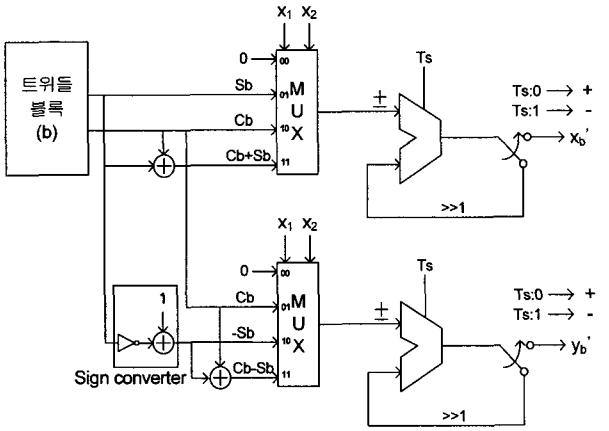

그림 6. 제안된 DA 블록(b)의 세부 구조
Fig. 6. Proposed DA block(b) structure.

그림 6의 DA 블록(b)에서는 C_b, S_b를 받아들여서 $C_b, S_b, -S_b, C_b+S_b, C_b-S_b$등 5개의 값들을 만들어내어야 하며, 이렇게 만들어진 4개의 값들은 MUX를 통하여 선택된다. MUX회로는 x_1, x_2의 LSB부터 어드레스로 동작하여 MUX의 입력 값들을 출력시키게 설계되었다. MUX의 어드레스에 따른 MUX의 출력은 표 2와 같다. 그림 6에서 T_s 신호는 입력신호의 MSB에서만 뺄셈기로 동작하도록 제어하는 역할을 하게 된다. 예를 들어, x_1, x_2가 16비트로 구성된 경우에 LSB부터 15비트 까지는 T_s=0 이므로 어드레스에 따라서 출력 값들을 쉬프트하여 더해주는 덧셈기로 동작하고, MSB 비트에서는 T_s=1 이 되어 MUX 출력 값을 쉬프트되어 온 값과 빼주는 뺄셈기로 동작하도록 제어하였다. 이렇게 출력되는 16개의 값들은 1비트씩 LSB로 쉬프트되며 새로운 값과 더해짐으로써 최종 출력 x'_b와 y'_b 가 얻어진다. 나머지 DA블록들의 구조도 지금까지 설계한 b용 DA블록과 같은 구조를 사용하였으며, 어드레스에 따른 MUX의 출력 값은 표 2와 같다. 다음 절에서는 RTL 코딩을 통하여 제안한 구조의 효율성을 입증한다.

Ⅳ. 실험 및 고찰

4.1 64-point FFT를 통한 비교

64-point FFT에 대하여 제안된 Radix-8 DA 나비연산기와 기존의 Radix-4 DA 나비연산기를 각각 사용하여 구현함으로써 제안 구조의 효율성을 살펴본다. 제안 구조는 2 스테이지로 구성되며, 제안 구조를 사용한 64-point FFT의 블록도는 그림 7과 같다. 기존의 3개의

표 2. 어드레스별 MUX 출력 값
Table 2. MUX output values of address.

x1n	x2n	출력 xb	출력 yb
0	0	0	0
0	1	Sb	Cb
1	0	Cb	-Sb
1	1	Cb+Sb	-Sb+Cb
x3n	x4n	출력 xc	출력 yc
0	0	0	0
0	1	Sc	Cc
1	0	Cc	-Sc
1	1	Cc+Sc	-Sc+Cc
x5n	x6n	출력 xd	출력 yd
0	0	0	0
0	1	Sd	Cd
1	0	Cd	-Sd
1	1	Cd+Sd	-Sd+Cd
x7n	x8n	출력 xe	출력 ye
0	0	0	0
0	1	Se	Ce
1	0	Ce	-Se
1	1	Ce+Se	-Se+Ce
x9n	x10n	출력 xf	출력 yf
0	0	0	0
0	1	Sf	Cf
1	0	Cf	-Sf
1	1	Cf+Sf	-Sfe+Cf
x11n	x12n	출력 xg	출력 yg
0	0	0	0
0	1	Sg	Cg
1	0	Cg	-Sg
1	1	Cg+Sg	-Sg+Cg
x13n	x14n	출력 xh	출력 yh
0	0	0	0
0	1	Sh	Ch
1	0	Ch	-Sh
1	1	Ch+Sh	-Sh+Ch

스테이지로 구성된 Radix-4 구조는 3개의 지연변환기가 사용되며, 이 구조에 대한 블록도는 그림 8과 같다

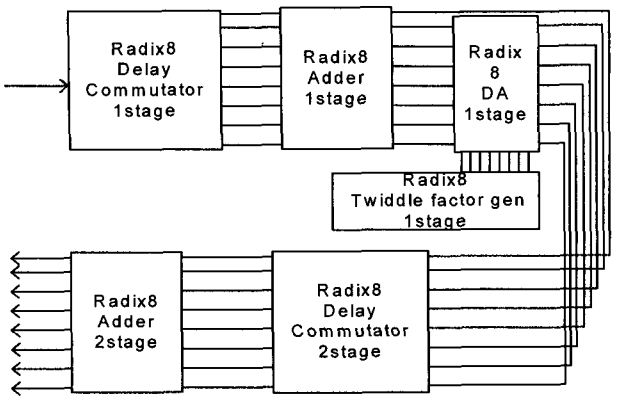

그림 7. 제안된 Radix-8 64-point FFT 전체블록도

Fig. 7. Proposed Radix-8 64-point FFT for over-all structure.

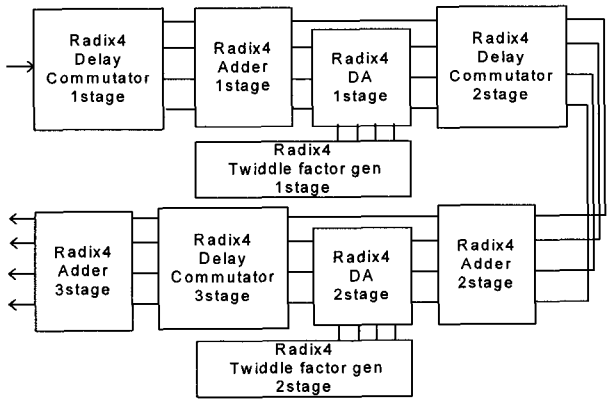

그림 8. 기존의 Radix-4 64-point FFT 전체블록도

Fig. 8. Conventional Radix-4 64-point FFT for over-all structure.

4. 2 Simulation

이 절에서는 제안된 나비연산기와 기존의 나비 연산기를 각각 사용하여 64-point FFT를 구현하여 면적을

표 3. DA를 적용한 64point FFT Radix-8구조의 C simulation 결과

Table 3. C simulation result for 64-point FFT Radix-8 using DA.

$X_R(0)$	0010011000001111	$X_R(1)$	0011000101000100
$X_R(7)$	1111010010010001	$X_R(8)$	1110010010101111
$X_R(15)$	0010010011011111	$X_R(16)$	1111011010011100
$X_R(23)$	0001100101010101	$X_R(24)$	0000001100001111
$X_R(31)$	1111101101101111	$X_R(32)$	0001100110111010
$X_R(39)$	0001100101010101	$X_R(40)$	0011000000000101
$X_R(47)$	0010010011011111	$X_R(48)$	0000101111100010
$X_R(55)$	1111010010010001	$X_R(56)$	0000010110000001
$X_I(0)$	0000000000000000	$X_I(0)$	0000010111011010
$X_I(7)$	1110100101011001	$X_I(8)$	0000111110101000
$X_I(15)$	1111110101010000	$X_I(16)$	1110111101100000
$X_I(23)$	1111111100011001	$X_I(24)$	1111011100100010
$X_I(31)$	0000000000000000	$X_I(32)$	1111110101100000
$X_I(39)$	0000000011100111	$X_I(40)$	1101111001010100
$X_I(47)$	0000001010110000	$X_I(48)$	0000100100101110
$X_I(55)$	0001011010100111	$X_I(56)$	1111101110010010

비교하였다. 제안 구조와 기존 구조는 Xilinx ISE 6.2i로 verilog로 코딩하여 FPGA기반으로 타이밍을 검증하였으며, Device Family는 Virtex4, Device는 xc4vlx100을 사용하였다. 표 3은 C 시뮬레이션 결과의 일부이며, Verilog-HDL 시뮬레이션 결과 값은 그림 9와 같이 C 시뮬레이션의 결과와 같음을 알 수 있다. 검증을 위한 64개의 입력신호로서 matlab을 이용하여 64개의 랜덤 입력 값들을 사용하였다.

64개의 입력 값들은 16비트의 정세도를 사용하였으며, 부호 1 비트, 정수 4 비트, 소수 11 비트로 구성하였다.

표 4에 제안 구조와 기존 구조의 gate-count를 비교

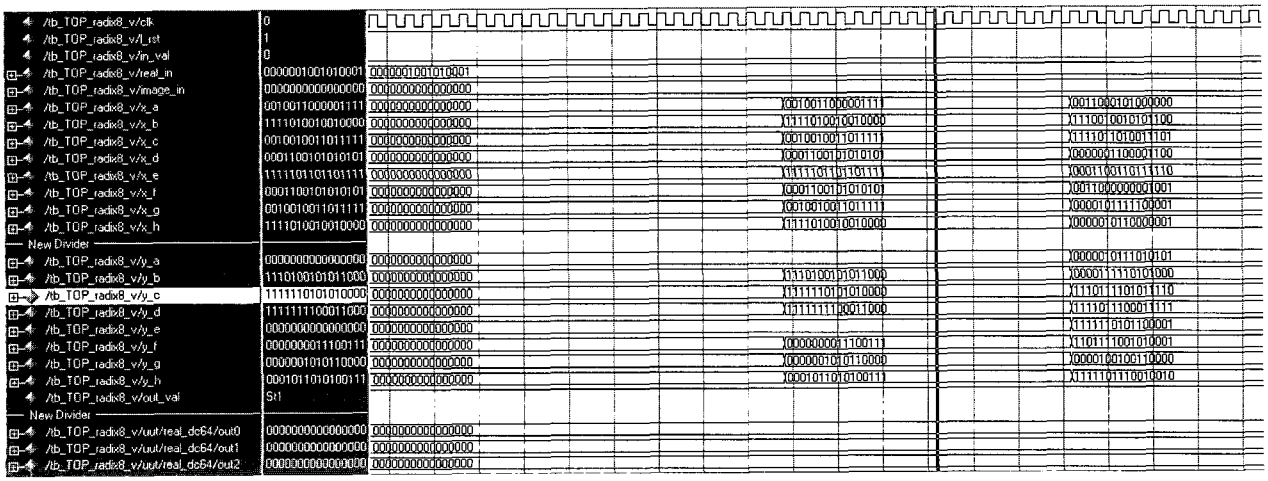

그림 9. DA를 적용한 64-point FFT Radix-8구조의 verilog HDL simulation 결과

Fig. 9. Verilog HDL simulation result for 64-point FFT Radix-8 algorihm using DA structure.

표 4. 64-point FFT 기존구조와 제안구조의 Gate count 비교
Table 4. Gate count for conventional and proposed 64-point FFT structures.

구분	1 stage				2 stage				3 stage				sub-total cell area
블록명	DC	덧셈 블록	DA 블록	TW	DC	덧셈 블록	DA 블록	TW	DC	덧셈 블록	DA 블록	TW	
기존 구조	9660 (4,830 x2)	4,800	13,569 (4,523 x3)	772	41,892 (20,946 x2)	4,800	13,569 (4,523 x3)	356	11942 (5,971 x2)	4,800			104,829
제안 구조	18,784 (9,392 x2)	34,333	32,221 (4,603 x7)	1,474	43,868 (21,934 x2)	34,333							156,436

하였다. 표 4 기존 구조는 Radix-4 알고리즘의 나비연산기 구조를 사용하였으며, DA를 사용하여 나비연산기를 설계하였다.[8] 기존 구조와 제안 구조 모두 지연변환기는 MDC(Multi-path Delay Commutator)구조를 사용하여 설계하였다. 표 4에서 보듯이 64-point FFT의 구현에 제안 구조는 3 스테이지가 필요하고 제안 구조는 Radix-8이므로 2 스테이지로 구성된다. 제안 구조의 gate-count가 기존 구조의 gate-count보다 49.2% 증가한 것을 알 수 있다. 하지만 제안 구조는 기존 구조와 비교하여 Throughput이 2배이므로 고속으로 출력을 계산해야하는 OFDM용 FFT에 사용하기 적합하다. 또한 상대적인 전력소모는 동작속도x구현면적에 비례하므로 기존 구조의 동작속도와 구현면적을 각각 1이라 정의하면 전력소모는 1이 된다. 제안구조의 동작속도는 0.5이고 구현면적은 1.492이므로 전력소모는 0.746이 됨을 알 수 있다. 따라서 전력소모의 효율성면에서는 제안구조가 기본 구조보다 25.4%의 감소 효과를 갖는다.

V. 결 론

이 논문에서는 고속 FFT 구조에 적합한 Radix-8 DIF 알고리즘을 제안하였다. Radix-8 알고리즘은 고속처리는 가능하나 구현면적이 증가하는 단점이 발생하는데, 제안된 나비연산기에서는 곱셈연산을 DA 방식으로 구현하여 구현면적이 증가하는 문제를 해결하였다. Radix-4 DA 방식의 나비연산기를 사용하는 64-point FFT를 구현하여 비교한 결과 구현면적이 49.2%가 증가하였다. 즉, Throughput을 2배로 증가시키기 위하여 하드웨어는 49.2%만 증가함을 Verilog-HDL 코딩을 통하여 확인하였다. 또한 기존 구조와 제안 구조가 같은 Throughput을 얻는 경우에는 전력소모가 25.4%가 감소하게 된다. 따라서 제안된 나비연산기를 사용하는 FFT 구조는 고속/저전력 FFT를 필요로하는 OFDM용 통신단말기에 사용될 수 있다.

참 고 문 헌

[1] B. M. Baas, "*A 9.5mW 330us 1024-point FFT Processor*", IEEE Custom Integrated Circuits Conference, pp. 127-130, 1998.

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

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

[4] A. V. Oppenheim and Ronald W. schafer, *Discrete Time Signal Processing*, Prentice Hall, pp581-605, 1989.

[5] P. Duhamel and H. Hollman, "*Split radix FFT algorithm*", Electronics Letters, vol. 20, no. 1, pp. 14-16, Jan. 1984.

[6] R. A. Haddad and Thomas W. Parsons, *Digital Signal Processing Theory, Applications, and Hardware*, Computer Science Press, pp. 147-149, 1991.

[7] S. A. White, "*Applications of Distributed Arithmetic to Digital Signal Processing: A tutorial Review*", IEEE ASSP MAGAZINE July 1989.

[8] 장영범, 이원상, 김도한, 김비철, 허은성, "Distributed Arithmetic을 사용한 OFDM용 저전력 Radix-4 FFT 구조", 전자공학회논문지 제43권 SP편 제1호, pp101-108, 2006년 1월.

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>

박 진 수(학생회원)
2006년 2월 상명대학교 정보통신공학과 졸업(공학사)
2006년~현재 상명대학교 대학원 컴퓨터정보통신공학과 석사과정
<주관심분야 : 통신신호처리, SoC 설계>

허 은 성(학생회원)
2006년 2월 상명대학교 정보통신공학과 졸업(공학사)
2006년~현재 상명대학교 대학원 컴퓨터정보통신공학과 석사과정
<주관심분야 : 통신신호처리, SoC 설계>

홍 대 기(정회원)
2003년 연세대학교 전자공학과 박사 졸업
2002년~2006년 전자부품연구원 선임 연구원
2006년~현재 상명대학교 정보통신공학과 교수
<주관심분야 : Digital and Wireless Communication, WPAN>