Journal of the Korea Academia-Industrial cooperation Society
Vol. 16, No. 1 pp. 683-690, 2015

http://dx.doi.org/10.5762/KAIS.2015.16.1.683
ISSN 1975-4701 / eISSN 2288-4688

새로운 DIT Radix-4 FFT 구조 및 구현

장영범[1*], 이상우[1]
[1]상명대학교 정보통신공학과

A New DIT Radix-4 FFT Structure and Implementation

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

요 약 FFT(Fast Fourier Transform) 알고리즘에는 DIT(Decimation-In-Time)와 DIF(Decimation-In- Frequency)가 있다. DIF 알고리즘은 Radix-2/4/8 등의 다양한 종류와 그 구현 방법이 개발되어 사용되고 잇으나, DIT 알고리즘은 순차적인 출력을 낼 수 있는 장점이 있음에도 불구하고 다양한 알고리즘이 연구되지 못하였다. 이 논문에서는 새로운 DIT Radix-4 FFT의 나비연산기(butterfly) 구조를 제안하고 검증하였다. 제안 구조를 사용하여 64-point FFT 구조를 설계하고 Verilog로 코딩하여 구현함으로써 제안 구조의 효용성을 입증하였다. 48개의 곱셈기를 사용하여 합성하였으며 678만 게이트 수를 나타내었다. 따라서 제안된 DIT Radix-4 FFT 구조는 순차적인 FFT 출력을 필요로 하는 OFDM 통신용 SoC(System on a Chip)에 사용될 수 있을 것이다.

Abstract Two basic FFT(Fast Fourier Transform) algorithms are the DIT(Decimation-In-Time) and the DIF (Decimation-In- Frequency). In spite of the advantage of the DIT algorithm is to generate a sequential output, various structures have not been made. In this paper, a new DIT Radix-4 FFT butterfly structure are proposed and implemented using Verilog coding. Through synthesis, it is shown that the 64-point FFT is implemented by 6.78 million gates. Since the proposed FFT structure has the advantage of a sequential output, it can be used in OFDM communication SoC(System on a Chip) which need a high speed FFT output.

Key Words : Butterfly, DIT, FFT, Radix-4, SoC

1. 서론

FFT(Fast Fourier Transform) 알고리즘은 DIT (Decimation-In-Time) 방식과 DIF (Decimation-In-Frequency) 방식이 있다. 그 중에서 DIF 알고리즘은 Radix-2/4/8 등의 알고리즘이 개발되었고 다양한 구현방법이 제안되었다[1-8]. 이와 비교하여 DIT는 순차적인 출력을 낼 수 있는 장점이 있음에도 다양한 알고리즘이 개발되지 못하였다[9-10]. 다양한 구조가 개발되어 있는 DIF 알고리즘은 출력이 순차적으로 나오지 않으므로 출력을 순차적으로 정돈하기 위한 회로와 시간이 필요하다. 특히 OFDM SoC에서는 FFT 블록의 병렬 출력이 순차적인 직렬 출력으로 변환(PSC, Parallel to Serial Conversion)되어야 하므로 DIT 구조가 유리하다. 따라서 이 논문에서는 새로운 DIT Radix-4 구조를 설계하고 구현함으로써 제안 구조의 효용성을 보인다. 이 논문의 2장에서는 DIT Radix-4 알고리즘을 유도하는 방법과 SFG(Signal Flow Graph)를 설계하며, 반도체 구현을 위한 효율적인 나비연산기(butterfly) 구조를 제안한다. 3장에서는 function simulation, Verilog coding, 합성을 통하여 64-point FFT 제안 구조의 반도체 구현 게이트 수를 산출하고 4장에서 결론을 맺는다.

본 논문은 상명대학교 2013학년도 교내연구비에 의하여 수행되었음.
*Corresponding Author : Young-Beom Jang(Sangmyung Univ.)
Tel: +82-41-550-5353 email: ybjang@smu.ac.kr
Received December 2, 2014 Revised (1st December 16, 2014, 2nd December 19, 2014) Accepted January 8, 2015

2. 제안된 DIT Radix-4 FFT 구조

2.1 DIT Radix-4 알고리즘과 SFG

이 절에서는 DIT Radix-4 알고리즘을 유도하는 과정과 유도된 식을 사용하여 SFG를 설계하는 과정을 소개한다[10]. 먼저 N-point DFT(Discrete Fourier Transform)의 정의는 다음 식과 같다.

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

여기서 W_N은 $e^{-j2\pi/N}$이며, DIT Radix-4 알고리즘을 유도하기 위해서 먼저 다음과 같이 분해한다.

$$\begin{aligned}X(k)=&\sum_{n=0,4,8,\cdots}x(n)\,W_N^{nk}+\sum_{n=1,5,9,\cdots}x(n)\,W_N^{nk}\\&+\sum_{n=2,6,10,\cdots}x(n)\,W_N^{nk}+\sum_{n=3,7,11,\cdots}x(n)\,W_N^{nk}\end{aligned} \tag{2}$$

이 식은 새로 인덱스 r을 사용하고, $N_1=N/4$으로 정의하면 다음과 같이 나타낼 수 있다.

$$\begin{aligned}X(k)=&\sum_{r=0}^{N_1-1}x(4r)\,W_N^{4rk}\\&+\sum_{r=0}^{N_1-1}x(4r+1)\,W_N^{(4r+1)k}\\&+\sum_{r=0}^{N_1-1}x(4r+2)\,W_N^{(4r+2)k}\\&+\sum_{r=0}^{N_1-1}x(4r+3)\,W_N^{(4r+3)k}\end{aligned} \tag{3}$$

또한 $a(r)=x(4r)$, $b(r)=x(4r+1)$, $c(r)=x(4r+2)$, $d(r)=x(4r+3)$으로 정의하고 $W_N^4=W_{N_1}$을 사용하여 간략화하면 다음과 같이 나타낼 수 있다.

$$\begin{aligned}X(k)=&\sum_{r=0}^{N_1-1}a(r)\,W_{N_1}^{rk}+W_N^k\sum_{r=0}^{N_1-1}b(r)\,W_{N_1}^{rk}\\&+W_N^{2k}\sum_{r=0}^{N_1-1}c(r)\,W_{N_1}^{rk}+W_{N_1}^{3k}\sum_{r=0}^{N_1-1}d(r)\,W_N^{rk}\end{aligned} \tag{4}$$

식 (4)에서 $a(r)$, $b(r)$, $c(r)$, $d(r)$의 N_1-point DFT를 각각 $A(k)$, $B(k)$, $C(k)$, $D(k)$로 정의하면 다음과 같이 나타낼 수 있다.

$$X(k)=A(k)+W_N^kB(k)+W_N^{2k}C(k)+W_{N_1}^{3k}D(k) \tag{5}$$

그런데 식 (5)는 오직 인덱스 k가 0부터 N_1-1까지 유효하므로 이제 k가 N_1부터 $2N_1-1$까지 유효한 식을 유도해보기로 한다. 식 (5)를 사용하여 k가 N_1부터 $2N_1-1$인 경우를 나타내면 다음과 같다.

$$\begin{aligned}X(k+N_1)=&A(k+N_1)+W_N^{(k+N_1)}B(k+N_1)\\&+W_N^{2(k+N_1)}C(k+N_1)+W_{N_1}^{3(k+N_1)}D(k+N_1)\end{aligned} \tag{6}$$

이 식에서 $A(k+N_1)=\sum a(r)\,W_{N_1}^{r(k+N_1)}=A(k)$ 이므로 $B(k+N_1)=B(k)$, $C(k+N_1)=C(k)$, $D(k+N_1)=D(k)$가 된다. 또한 $W_N^{N_1}=-j$, $W_N^{2N_1}=-1$, $W_N^{3N_1}=j$, 이므로 이를 사용하여 식 (6)을 다음과 같이 쓸 수 있다.

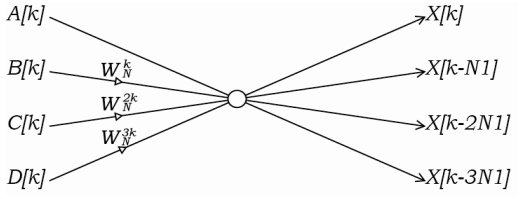
$$\begin{aligned}X(k+N_1)=&A(k)-jW_N^kB(k)-W_N^{2k}C(k)\\&+jW_{N_1}^{3k}D(k)\end{aligned} \tag{7}$$

같은 방법으로 k가 $2N_1$부터 $3N_1-1$까지 유효한 식과 $3N_1$부터 $4N_1-1$까지 유효한 식을 각각 유도하면 다음 식과 같다.

$$\begin{aligned}X(k+2N_1)=&A(k)-W_N^kB(k)+W_N^{2k}C(k)\\&-W_{N_1}^{3k}D(k)\end{aligned} \tag{8}$$

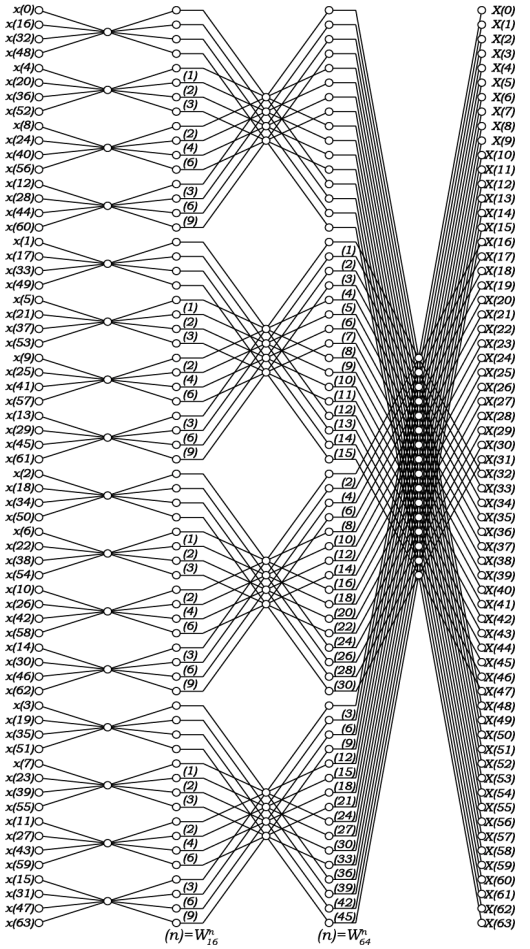
$$\begin{aligned}X(k+3N_1)=&A(k)+jW_N^kB(k)-W_N^{2k}C(k)\\&-jW_{N_1}^{3k}D(k)\end{aligned} \tag{9}$$

Fig. 1은 지금까지 유도한 식 (5), (7), (8), (9)를 사용한 나비연산기의 구조를 보여준다.



[Fig. 1] DIT Radix-4 butterfly structure

Fig. 1의 나비연산기를 사용하여 64-point FFT 구조를 설계하면 Fig. 2와 같은 SFG를 얻을 수 있다. 즉 Fig. 2는 제안된 Radix-4 DIT 알고리즘을 사용하여 설계한 SFG이다.

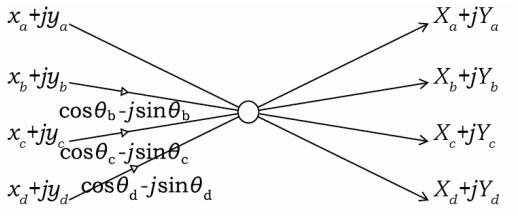


[Fig. 2] SFG of the 64-point DIT Radix-4 FFT

2.2 제안된 DIT Radix-4 나비연산기 구조

이 절에서는 1 절에서 유도한 DIT Radix-4 SFG에 사용되는 나비연산기의 효율적인 구현 방법을 제안한다. 반도체 칩을 사용하여 FFT를 구현하기 위해서는 모든 입출력이 실수와 허수로 정의되어야 한다. 따라서 먼저 Fig. 1의 나비연산기를 Fig. 3과 같이 정의한다. Fig. 3에서 보듯이, 구현을 위하여 나비연산기의 입력과 출력은 다음과 같이 정의한다.

$$\begin{aligned} A(k) &= x_a + jy_a, \quad X(k) = X_a + jY_a \\ B(k) &= x_b + jy_b, \quad X(k+N_1) = X_b + jY_b \\ C(k) &= x_c + jy_c, \quad X(k+2N_1) = X_c + jY_c \\ D(k) &= x_d + jy_d, \quad X(k+3N_1) = X_d + jY_d \end{aligned} \tag{10}$$



[Fig. 3] D Radix-4 butterfly for implementation

또한 트위들 인자(twiddle factor)도 연산을 위해 다음과 같이 직각형 복소수로 정의한다.

$$\begin{aligned} W_N^{k} &= \cos\theta_b - j\sin\theta_b \\ W_N^{2k} &= \cos\theta_c - j\sin\theta_c \\ W_N^{3k} &= \cos\theta_d - j\sin\theta_d \end{aligned} \tag{11}$$

이제부터 Fig. 3의 나비연산기에서 출력을 구하는 효율적인 구조를 유도한다. 먼저 Fig. 3에서 입력과 트위들 인자 곱셈이 이루어진 후의 노드를 각각 $x'_b + jy'_b$, $x'_c + jy'_c$, $x'_d + jy'_d$로 정의한다.

$$\begin{aligned} x'_b + jy'_b &= (\cos\theta_b - j\sin\theta_b)(x_b + jy_b) \\ x'_c + jy'_c &= (\cos\theta_c - j\sin\theta_c)(x_c + jy_c) \\ x'_d + jy'_d &= (\cos\theta_d - j\sin\theta_d)(x_d + jy_d) \end{aligned} \tag{12}$$

식 (12)에서 실수부와 허수부를 각각 정리 하면 다음 식과 같다.

$$\begin{aligned} x'_b &= x_b\cos\theta_b + y_b\sin\theta_b, \quad y'_b = y_b\cos\theta_b - x_b\sin\theta_b \\ x'_c &= x_c\cos\theta_c + y_c\sin\theta_c, \quad y'_c = y_c\cos\theta_c - x_c\sin\theta_c \\ x'_d &= x_d\cos\theta_d + y_d\sin\theta_d, \quad y'_d = y_d\cos\theta_d - x_d\sin\theta_d \end{aligned} \tag{13}$$

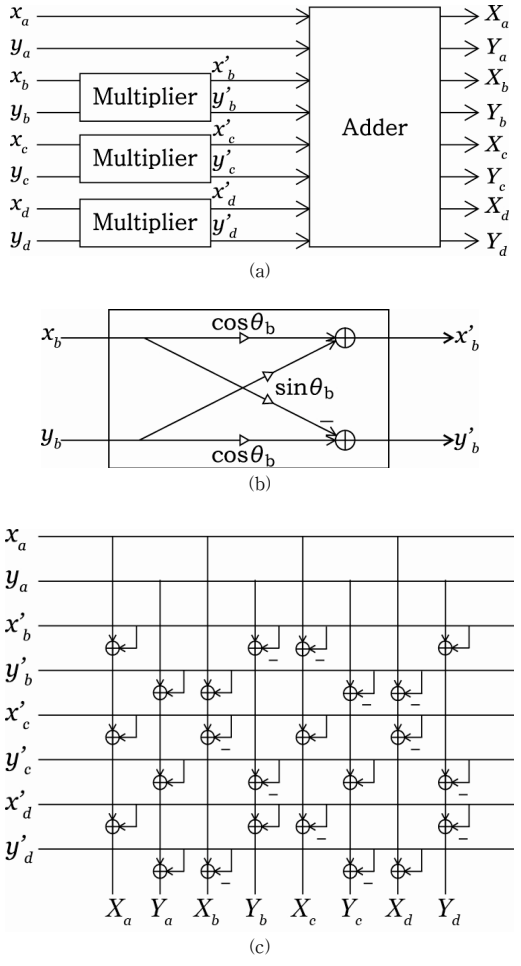
이제 Fig. 3의 나비연산기 출력을 식 (13)을 사용하여 나타내면 다음과 같다.

$$\begin{aligned} X_a + jY_a &= (x_a + jy_a) + (x'_b + jy'_b) \\ &\quad + (x'_c + jy'_c) + (x'_d + jy'_d) \\ X_b + jY_b &= (x_a + jy_a) - j(x'_b + jy'_b) \\ &\quad - (x'_c + jy'_c) + j(x'_d + jy'_d) \\ X_c + jY_c &= (x_a + jy_a) - (x'_b + jy'_b) \\ &\quad + (x'_c + jy'_c) - (x'_d + jy'_d) \\ X_d + jY_d &= (x_a + jy_a) + j(x'_b + jy'_b) \\ &\quad - (x'_c + jy'_c) - j(x'_d + jy'_d) \end{aligned} \tag{14}$$

식 (14)에서 실수부와 허수부를 각각 정리하면 다음 식과 같다.

$$X_a = x_a + x'_b + x'_c + x'_d,\ Y_a = y_a + y'_b + y'_c + y'_d \quad (15)$$
$$X_b = x_a + y'_b - x'_c - y'_d,\ Y_b = y_a - x'_b - y'_c + x'_d$$
$$X_c = x_a - x'_b + x'_c - x'_d,\ Y_c = y_a - y'_b + y'_c - y'_d$$
$$X_b = x_a - y'_b - x'_c + y'_d,\ Y_b = y_a + x'_b - y'_c - x'_d$$

지금까지 유도한 식 (13)과 식 (15)를 사용하면 Fig. 4와 같은 나비연산기가 만들어진다[10].



[Fig. 4] Proposed butterfly structure (a) overall block diagram, (b) multiplier block, (c) adder block

Fig. 4의 제안 구조에서 (a)는 전체 블록도이고 (b)와 (c)는 각각 곱셈기와 덧셈기의 내부 구조이다. 다음 절에서 Fig. 2와 Fig. 4의를 구조를 구현해보기로 한다.

3. Function simulation 및 구현

3.1 Function simulation

Fig. 2와 4의 구조에 대하여 정상 동작을 확인하기 위해 function simulation을 수행하였다. 사용된 입력 벡터 $x[0] \sim x[63]$은 표 1과 같은 $-1.0 \sim 1.0$사이의 64개의 실수 값을 사용하였다.

[Table 1] Input and output vectors for function simulation

x[0]= 0.4605	X(0)= −2.3592 + 0.0000i
x[1]= −0.3122	X(1)= 0.1146 + 3.2110i
x[2]= 0.1681	X(2)= 1.0585 + 1.6702i
x[3]= −0.7845	X(3)= 3.5875 − 6.3863i
x[4]= 0.8126	X(4)= 2.4889 + 3.4334i
x[5]= 0.7593	X(5)= −6.9437 − 4.6805i
x[6]= 0.6355	X(6)= −5.5614 + 0.5575i
x[7]= −0.4785	X(7)= 1.7772 + 2.2365i
x[8]= 0.1887	X(8)= −2.4585 − 4.7305i
x[9]= −0.9550	X(9)= 1.0650 + 2.8321i
x[10]=−0.1495	X(10)= 3.5596 + 1.1961i
x[11]=−0.3746	X(11)= 6.4955 + 3.6918i
x[12]=−0.6770	X(12)=−2.7702 + 0.6047i
x[13]=−0.6425	X(13)= 3.8098 + 1.9517i
x[14]=−0.1542	X(14)= 0.7528 − 6.0029i
x[15]=−0.8115	X(15)= 2.9978 + 0.5043i
x[16]= 0.1970	X(16)= 1.6813 + 0.0899i
x[17]=−0.0582	X(17)= 1.3509 + 0.6740i
x[18]= 0.3919	X(18)=−2.1556 + 0.9694i
x[19]= 0.3998	X(19)= 0.2590 + 0.9829i
x[20]= 0.2771	X(20)=−4.0221 − 0.9514i
x[21]=−0.9328	X(21)=−2.5312 − 0.5286i
x[22]=−0.8624	X(22)=−3.4193 + 3.4465i
x[23]=−0.3608	X(23)=−1.6570 + 0.4295i
x[24]= 0.0617	X(24)= 0.4753 − 2.7567i
x[25]= 0.3089	X(25)= 0.6678 − 3.5513i
x[26]=−0.1848	X(26)= 3.2590 − 1.8220i
x[27]= 0.6400	X(27)= 1.7467 + 1.4740i
x[28]= 0.4367	X(28)=−0.1609 + 3.9376i
x[29]= 0.9373	X(29)= 0.2749 − 2.5098i
x[30]= 0.0627	X(30)= 2.4537 + 1.3847i
x[31]=−0.3497	X(31)= 6.9724 + 5.9154i
x[32]=−0.7887	X(32)= 1.4950 + 0.0000i
x[33]= 0.2219	X(33)= 6.9724 − 5.9154i
x[34]= 0.5576	X(34)= 2.4537 − 1.3847i
x[35]=−0.1531	X(35)= 0.2749 + 2.5098i
x[36]=−0.8184	X(36)=−0.1609 − 3.9376i
x[37]=−0.4671	X(37)= 1.7467 − 1.4740i
x[38]=−0.6927	X(38)= 3.2590 + 1.8220i
x[39]=−0.4380	X(39)= 0.6678 + 3.5513i
x[40]=−0.1198	X(40)= 0.4753 + 2.7567i

x[41]= 0.0543 X(41)=-1.6570 - 0.4295i
x[42]=-0.0852 X(42)=-3.4193 - 3.4465i
x[43]= 0.7507 X(43)=-2.5312 + 0.5286i
x[44]= 0.0361 X(44)=-4.0221 + 0.9514i
x[45]= 0.8872 X(45)= 0.2590 - 0.9829i
x[46]= 0.2754 X(46)=-2.1556 - 0.9694i
x[47]= 0.9154 X(47)= 1.3509 - 0.6740i
x[48]=-0.5186 X(48)= 1.6813 - 0.0899i
x[49]= 0.3522 X(49)= 2.9978 - 0.5043i
x[50]=-0.4219 X(50)= 0.7528 + 6.0029i
x[51]= 0.3436 X(51)= 3.8098 - 1.9517i
x[52]= 0.3903 X(52)=-2.7702 - 0.6047i
x[53]=-0.8640 X(53)= 6.4955 - 3.6918i
x[54]=-0.4904 X(54)= 3.5596 - 1.1961i
x[55]=-0.5519 X(55)= 1.0650 - 2.8321i
x[56]= 0.3357 X(56)=-2.4585 + 4.7305i
x[57]= 0.6888 X(57)= 1.7772 - 2.2365i
x[58]=-0.3111 X(58)=-5.5614 - 0.5575i
x[59]= 0.5610 X(59)=-6.9437 + 4.6805i
x[60]= 0.3507 X(60)= 2.4889 - 3.4334i
x[61]=-0.9866 X(61)= 3.5875 + 6.3863i
x[62]= 0.2043 X(62)= 1.0585 - 1.6702i
x[63]=-0.2265 X(63)= 0.1146 - 3.2110i

MatLab을 사용하여 위의 입력 벡터를 Fig. 2와 Fig. 4의 구조에 입력시켜서 표 1과 같은 FFT 출력 $X(0) \sim X(63)$의 출력을 얻었다. 이 출력 벡터가 MatLab의 FFT 함수를 사용하여 얻은 출력과 일치함을 확인하였다. 따라서 Fig. 2와 Fig. 4의 제안 구조는 기능적으로 FFT 변환을 올바르게 수행한다.

3.2 Verilog RTL 코딩

이 절에서는 Fig. 2와 4의 구조에 대하여 Verilog RTL 코딩을 수행하였다. 곱셈기 설계시 각각의 나비연산기마다 고정된 cos과 sin 값을 곱하여 곱셈연산을 수행하였다. 곱셈 연산은 64 bit으로 sign extension하여 계산하였다. 입력 값의 bit-width는 32 bit으로 정하였으며 MSB인 bit[31]은 sign bit, [30:16]은 integer bit(15bit), [15:0]은 fractional bit(16bit)으로 구성하였으며 실수와 허수를 분리하여 계산하였다. Simulation에서는 32 bit로 구성된 실수 64개가 입력되며, 32 bit로 구성된 실수 64개와 허수 64개가 출력된다. Verilog simulation에 사용된 입력 값은 위의 MatLab에서 사용한 실수값을 이진수로 변환하여 사용하였다. Verilog simulation 결과 최종적으로 표 2와 같은 실수부 이진(binary) 출력 벡터를 얻었다.

[Table 2] Verilog simulation results(real part of the output vectors)

X(0)= fffd a410 = -2.35913085938
X(1)= 0000 1d52 = 0.114532470703
X(2)= 0001 0eee = 1.0583190918
X(3)= 0003 965a = 3.58731079102
X(4)= 0002 7d1e = 2.48873901367
X(5)= fff9 0e7b = -6.94343566895
X(6)= fffa 7054 = -5.56121826172
X(7)= 0001 c6fb = 1.77726745605
X(8)= fffd 8aa3 = -2.45845031738
X(9)= 0001 10a2 = 1.06497192383
X(10)= 0003 8f40 = 3.5595703125
X(11)= 0006 7ed0 = 6.49536132812
X(12)= fffd 3ad9 = -2.77012634277
X(13)= 0003 cf47 = 3.80967712402
X(14)= 0000 c0b3 = 0.752731323242
X(15)= 0002 ff69 = 2.99769592285
X(16)= 0001 ae65 = 1.6812286377
X(17)= 0001 59d0 = 1.35083007812
X(18)= fffd d832 = -2.15548706055
X(19)= 0000 424f = 0.259017944336
X(20)= fffb fa61 = -4.02195739746
X(21)= fffd 780c = -2.53106689453
X(22)= fffc 94a4 = -3.41937255859
X(23)= fffe 57d6 = -1.65689086914
X(24)= 0000 79aa = 0.475250244141
X(25)= 0000 aaf1 = 0.667739868164
X(26)= 0003 4249 = 3.2589263916
X(27)= 0001 bf26 = 1.74667358398
X(28)= ffff d6ca = -0.160980224609
X(29)= 0000 4660 = 0.27490234375
X(30)= 0002 741e = 2.45358276367
X(31)= 0006 f8e6 = 6.97225952148
X(32)= 0001 7eb2 = 1.49490356445
X(33)= 0006 f8e6 = 6.97225952148
X(34)= 0002 741e = 2.45358276367
X(35)= 0000 4660 = 0.27490234375
X(36)= ffff d6ca = -0.160980224609
X(37)= 0001 bf25 = 1.7466583252
X(38)= 0003 4248 = 3.25891113281
X(39)= 0000 aaf1 = 0.667739868164
X(40)= 0000 79ab = 0.47526550293
X(41)= fffe 57d6 = -1.65689086914
X(42)= fffc 94a6 = -3.41934204102
X(43)= fffd 780c = -2.53106689453
X(44)= fffb fa61 = -4.02195739746
X(45)= 0000 4251 = 0.259048461914
X(46)= fffd d833 = -2.15547180176
X(47)= 0001 59d1 = 1.35084533691
X(48)= 0001 ae65 = 1.6812286377
X(49)= 0002 ff6c = 2.99774169922
X(50)= 0000 c0b6 = 0.752777099609

X(51)= 0003 cf4b = 3.80973815918
X(52)= fffd 3adb = -2.7700958252
X(53)= 0006 7ed0 = 6.49536132812
X(54)= 0003 8f40 = 3.5595703125
X(55)= 0001 10a2 = 1.06497192383
X(56)= fffd 8aa4 = -2.45843505859
X(57)= 0001 c6fb = 1.77726745605
X(58)= fffa 7055 = -5.56120300293
X(59)= fff9 0e7a = -6.94345092773
X(60)= 0002 7d20 = 2.48876953125
X(61)= 0003 9664 = 3.58746337891
X(62)= 0001 0ef4 = 1.05841064453
X(63)= 0000 1d5c = 0.11468505859

표 2에서는 이진수의 출력 벡터와 십진수로 변환한 출력 벡터를 같이 나타내었다. 이 십진수의 Verilog 출력 값이 function simulation의 실수부 출력과 같음을 확인하였다. 표 3은 Verilog simulation 결과 최종적으로 얻은 허수부 이진 출력 벡터이다.

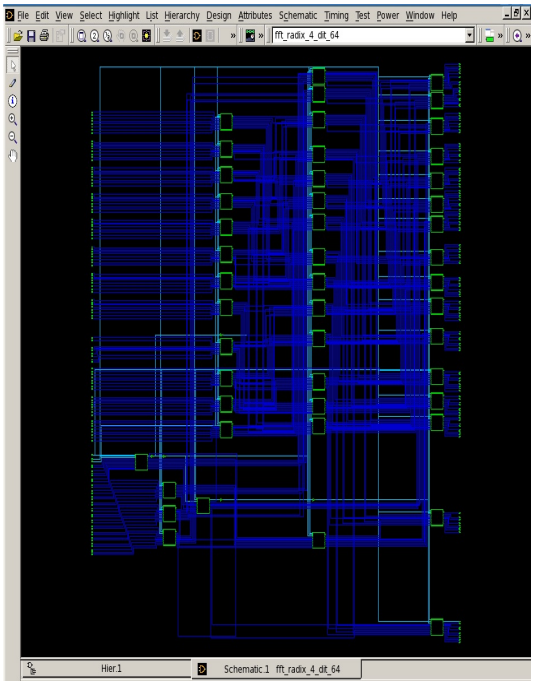
[Table 3] Verilog simulation results(imaginary part of the output vectors)

X(0)= 0000 0000 = 0.0
X(1)= 0003 35f8 = 3.21081542969
X(2)= 0001 ab8f = 1.67015075684
X(3)= ff9 9d23 = -6.38618469238
X(4)= 0003 6eeb = 3.43327331543
X(5)= fffb 51d7 = -4.68031311035
X(6)= 0000 8eb7 = 0.557479858398
X(7)= 0002 3c87 = 2.23643493652
X(8)= fffb 4505 = -4.73039245605
X(9)= 0002 d4ff = 2.83201599121
X(10)= 0001 322c = 1.19598388672
X(11)= 0003 b113 = 3.69169616699
X(12)= 0000 9acf = 0.604721069336
X(13)= 0001 f39d = 1.95161437988
X(14)= fff9 ff4c = -6.00274658203
X(15)= 0000 811b = 0.504318237305
X(16)= 0000 1707 = 0.089950561523
X(17)= 0000 ac88 = 0.673950195312
X(18)= 0000 f82b = 0.96940612793
X(19)= 0000 fb9a = 0.982818603516
X(20)= ffff 0c70 = -0.951416015625
X(21)= ffff 78b2 = -0.528533935547
X(22)= 0003 7241 = 3.44630432129
X(23)= 0000 6df2 = 0.429473876953
X(24)= fffd 3e4e = -2.75662231445
X(25)= fffc 72e2 = -3.55123901367
X(26)= fffe 2d93 = -1.82197570801
X(27)= 0001 795b = 1.4740447998
X(28)= 0003 f004 = 3.93756103516
X(29)= fffd 7d7e = -2.50979614258
X(30)= 0001 6279 = 1.38465881348
X(31)= 0005 ea48 = 5.91516113281
X(32)= 0000 0000 = 0.0
X(33)= fffa 15b8 = -5.91516113281
X(34)= fffe 9d87 = -1.38465881348
X(35)= 0002 8283 = 2.50981140137
X(36)= fffc 0ffd = -3.93754577637
X(37)= fffe 86a7 = -1.47401428223
X(38)= 0001 d26f = 1.82200622559
X(39)= 0003 8d1f = 3.55125427246
X(40)= 0002 c1b3 = 2.75663757324
X(41)= ffff 9211 = -0.429428100586
X(42)= fffc 8dc0 = -3.4462890625
X(43)= 0000 874f = 0.528549194336
X(44)= 0000 f391 = 0.951431274414
X(45)= ffff 0465 = -0.982833862305
X(46)= ffff 07d6 = -0.969390869141
X(47)= ffff 5379 = -0.673934936523
X(48)= ffff e8f9 = -0.089950561523
X(49)= ffff 7ee4 = -0.504333496094
X(50)= 0006 00b3 = 6.00273132324
X(51)= fffe 0c60 = -1.95166015625
X(52)= ffff 6530 = -0.604736328125
X(53)= fffc 4eec = -3.69171142578
X(54)= fffe cdd1 = -1.19602966309
X(55)= fffd 2b00 = -2.83203125
X(56)= 0004 bafa = 4.73037719727
X(57)= fffd c37a = 2.23641967773
X(58)= ffff 714d = -0.557418823242
X(59)= 0004 ae2b = 4.68034362793
X(60)= fffc 9114 = -3.43328857422
X(61)= 0006 62dc = 6.38616943359
X(62)= fffe 546d = -1.67021179199
X(63)= fffc ca04 = -3.21087646484

표 3에서는 허수부 이진수 출력 벡터와 변환된 십진수 벡터를 함께 표기하였다. 표 3의 십진수 Verilog 출력 값이 function simulation의 허수 출력과 같음을 확인하였다.

3.3 합성

이 절에서는 설계된 Verilog 코드에 대한 합성을 통하여 제안 구조의 효용성을 실험하였다. 합성에는 Magnachip Library를 사용하였으며 clock은 100MHz를 사용하였다. 64-point FFT에 대한 Fig. 2와 Fig. 4 구조의 schematic은 Fig. 5와 같다.



[Fig. 5] Schematic of the proposed DIT Radix-4 structure(64-point FFT)

```
****************************************
Report : area
Design : fft_radix_4_dit_64
Version: G-2012.06-SP5-5
Date   : Tue Nov  4 17:16:21 2014
****************************************

Library(s) Used:

    m18ewm180d_wci (File: /MAGNACHIP/primitive/m18ewm180d_synopsyslib_121121/db/m18ewm180d_wci.db)

Number of ports:                 8196
Number of nets:                 16395
Number of cells:                   53
Number of combinational cells:      5
Number of sequential cells:         0
Number of macros:                   0
Number of buf/inv:                  5
Number of references:              52

Combinational area:       5969106.762949
Buf/Inv area:              254434.652134
Noncombinational area:     813495.055305
Net Interconnect area:    2329.668903

Total cell area:          6782601.818254
Total area:               6784931.487157
1
```

[Fig. 6] Gate count of the proposed DIT Radix-4 structure(64-point FFT)

Schematic에서 보듯이 3개의 스테이지로 구성되며 각 스테이지마다 16개의 곱셈기 모듈을 사용하였다. 합성 결과에 대한 게이트 수는 Fig. 6과 같다. Fig. 6에서 보듯이 총 게이트 수는 678만 게이트이다. 48개의 곱셈기 모듈을 사용하였으며 합성 후 combinational 게이트 수가 596만 게이트이다. 만일 각 스테이지에서 1 개의 곱셈기만을 사용하여 16번의 곱셈연산을 반복 수행하도록 설계하면 게이트 수를 감소시킬 수 있으나 연산 시간은 16 배로 늘어나게 된다. 이 절에서 Fig. 2와 Fig. 4 구조의 동작을 입증하였고 구현을 통하여 효용성을 입증하였다.

4. 결론

이 논문에서는 DIT Radix-4 FFT 알고리즘의 구현을 위한 효율적인 나비연산기 구조를 제안하고, 구현을 통하여 그 구조의 효용성을 입증하였다. DIF 구조와 비교하여 DIT 구조의 장점은 FFT 출력이 순차적으로 출력되는 것이다. OFDM 통신용 SoC에 사용되는 FFT 블록은 연산후 병렬 출력이 순차적인 직렬 출력으로 변환되어야 하므로 순차적인 출력이 나오는 DIT 구조가 효용성이 높다. 이 논문에서는 첫째, 새로운 DIT Radix-4 FFT 구조를 제안하였다. 둘째, 제안 구조의 64-point FFT 구현을 통하여 게이트 수를 산출함으로써 제안 구조의 효용성을 입증하였다. 따라서 제안된 DIT Radix-4 FFT 구조는 순차적인 FFT 출력을 필요로 하는 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 proce ssor 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] J. Lee and H. Lee, "A high-Speed 2-Parallel Radix-2^4 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

[3] H. J. Kim and Y. B. Jang, "Low-area FFT processor

structure using radix-4^2 algorithm", *Journal of IEEK*, vol. 49-SD, No. 3, pp. 8-14, Mar. 2012.

[4] 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. DOI: http://dx.doi.org/10.1109/TSCAS.2011.5938099

[5] 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

[6] 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

[7] J. Y. Oh, J. S. Cha, S. K. Kim, and M. S. Lim, "Implementation of Orthogonal Frequency Division Multiplexing using radix-N Pipeline Fast Fourier Transform(FFT) Processor", *Jpn. J. Appl. Phys.*, vol. 42, No. 4B, pp. 1-6, April, 2003. DOI: http://dx.doi.org/10.1143/JJAP.42.2176

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

[9] K. Rao, D. N. Kim, and J. J. Hwang, "Fast Fourier Transform - Algorithms and Applications" *Springer*, 2011.

[10] 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

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, SoC설계, 비디오신호처리

이 상 우(Sang-Woo Lee) **[준회원]**

- 2013년 8월 : 상명대학교 정보통신공학과 (공학사)
- 2013년 9월 ~ 현재 : 상명대학교 일반대학원 정보통신공학과 석사과정

<관심분야>
통신신호처리, SoC설계