논문 13-38A-12-20

한국통신학회논문지(J-KICS) '13-12 Vol.38A No.12
http://dx.doi.org/10.7840/kics.2013.38A.12.1145

DIT Radix-4 FFT 구현을 위한 저전력 Butterfly 구조

장 영 범◆°, 이 상 우*

Low-power Butterfly Structure for DIT Radix-4 FFT Implementation

Young-Beom Jang◆°, Sang-Woo Lee*

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 사용되는데 반하여 DIT 알고리즘은 순차적인 출력을 낼 수 있는 장점에도 불구하고 다양한 구현방법이 연구되지 못하였다. 이 논문에서는 DIT Radix-4 알고리즘을 유도하며 반도체 구현을 위한 효율적인 butterfly 구조를 제안한다.

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

ABSTRACT

There are two FFT(Fast Fourier Transform) algorithms, which are DIT(Decimation-In-Time) and DIF(Decimation-In- Frequency). Even the DIF algorithm is more widely used because of its various implementation architectures, the DIT structures have not been investigated. In this paper, the DIT Radix-4 algorithm is derived and its efficient butterfly structure is proposed for SoC(System on a Chip) implementation.

Ⅰ. 서 론

FFT(Fast Fourier Transform) 알고리즘은 DIT(Decimation-In-Time)와 DIF(Decimation -In-Frequency)가 있다. 그 중에서 DIF 알고리즘은 Radix-2/4/8 등의 알고리즘이 개발되었고 다양한 구현 방법이 제안되었다[1-4]. 이와 비교하여 DIT는 순차적인 출력을 낼 수 있는 장점이 있음에도 다양한 알고리즘이 개발되지 못하였다[5]. 이 논문에서는 DIT Radix-4 알고리즘을 유도하는 방법과 SFG(Signal Flow Graph)를 설명하며, 반도체 구현을 위한 효율적인 butterfly 구조를 제안한다.

Ⅱ. 제안된 DIT Radix-4 FFT 구조

2.1. DIT Radix-4 알고리즘과 SFG

이 절에서는 DIT Radix-4 알고리즘을 유도하는 과정과 유도된 식을 사용한 SFG(signal flow graph)를 살펴보기로 한다. 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.

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

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

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

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.

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

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.

※ 본 연구는 2012학년도 상명대학교 교내연구비를 지원받아 수행하였음.
◆° First Author and Corresponding Author : 상명대학교 정보통신공학과, ybjang@smu.ac.kr, 정회원
* 상명대학교 정보통신공학과, lsw0143@nate.com, 학생회원
논문번호 : KICS2013-11-491, 접수일자 : 2013년 11월 11일, 심사일자 : 2013년 11월 28일, 최종논문접수일자 : 2013년 12월 5일

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

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.

$$X(k+N_1)=A(k+N_1)+W_N^{(k+N_1)}B(k+N_1) + W_N^{2(k+N_1)}C(k+N_1)+W_{N_1}^{3(k+N_1)}D(k+N_1) \quad (6)$$

이 식에서 $A(k+N_1)=\sum a(r)W_{N_1}^{r(k+N_1)}=A(k)$이며 $B(k+N_1)=B(k)$, $C(k+N_1)=C(k)$, $D(k+N_1)=D(k)$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.

$$X(k+N_1)=A(k)-jW_N^{k}B(k)-W_N^{2k}C(k)+jW_{N_1}^{3k}D(k) \quad (7)$$

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.

$$X(k+2N_1)=A(k)-W_N^{k}B(k)+W_N^{2k}C(k)-W_{N_1}^{3k}D(k) \quad (8)$$

$$X(k+3N_1)=A(k)+jW_N^{k}B(k)-W_N^{2k}C(k)-jW_{N_1}^{3k}D(k) \quad (9)$$

그림 1은 지금까지 유도한 식 (5), (7), (8), (9)를 사용한 butterfly를 보여주며 이 butterfly를 사용한 64-point DIT Radix-4의 SFG는 그림 2와 같다.

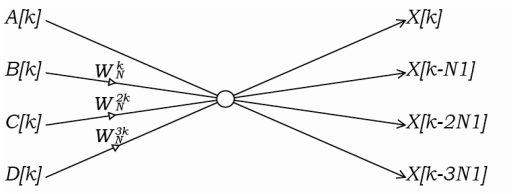

그림 1. DIT Radix-4 나비연산기 구조
Fig. 1. DIT Radix-4 butterfly structure

2.2. 제안된 저전력 DIT Radix-4 butterfly 구조

이 절에서는 1절에서 유도한 DIT Radix-4 SFG에 사용되는 butterfly의 효율적인 구현 방법을 제안한다. 반도체 칩을 사용하여 FFT를 구현하기 위해서는 모든 입출력이 실수와 허수로 기술되어야 한다. 따라서 그림 1의 butterfly를 다음 그림과 같이 정의한다.

그림 3에서 보듯이, 구현을 위하여 나비연산기의 입력과 출력은 다음과 같이 정의한다.

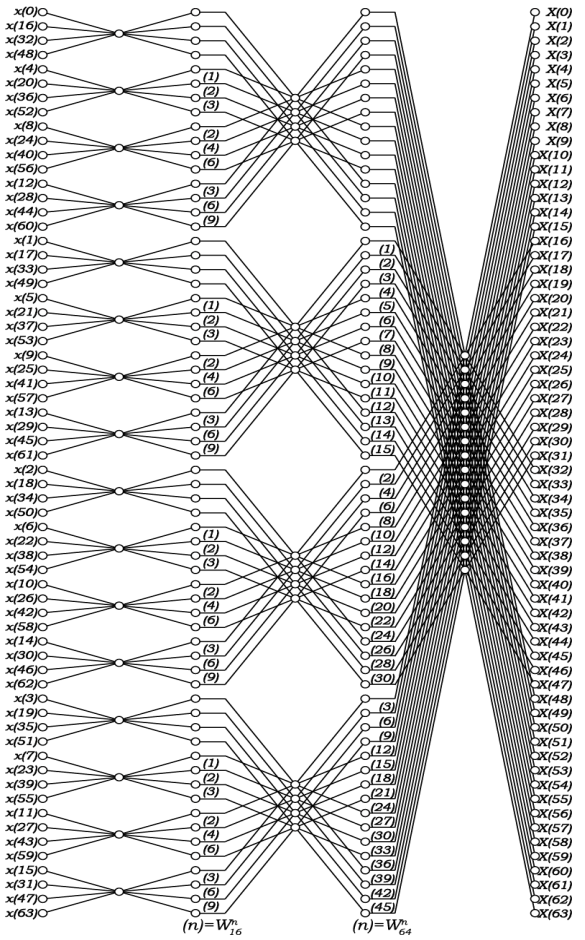

그림 2. 64-point DIT Radix-4 FFT의 SFG
Fig. 2. SFG of the 64-point DIT Radix-4

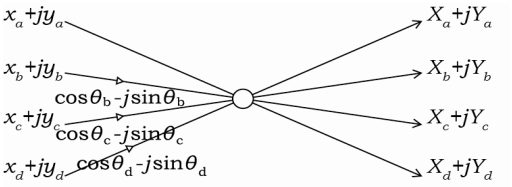

그림 3. 구현을 위한 DIT Radix-4 나비연산기
Fig. 3. DIT Radix-4 butterfly for implementation

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

또한 twiddle factor도 연산을 위해 다음과 같이 직각형 복소수로 정의한다.

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

이제부터 그림 3의 나비연산기에서 출력을 구하는 효율적인 구조를 유도한다. 먼저 그림 3에서 입력과 twiddle factor 곱셈이 이루어진 후의 노드를 각각

$x'_b+jy'_b$, $x'_c+jy'_c$, $x'_d+jy'_d$로 정의한다.

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

식 (12)에서 실수부와 허수부를 각각 정리 하면 다음 식과 같다.

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

이제는 그림 3 butterfly의 출력을 식 (13)을 사용하여 나타내면 다음과 같다.

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

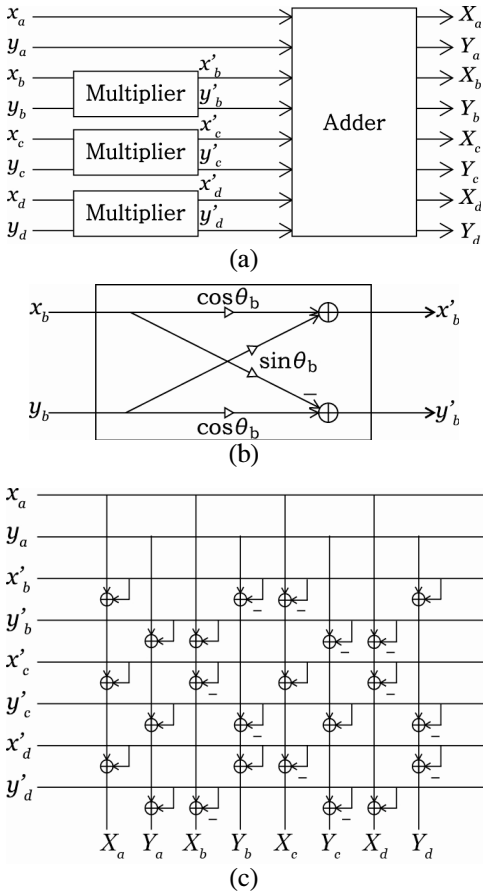

그림 4. 제안된 나비연산기 블록도 (a) 전체 블록도, (b) multiplier 내부구조, (c) adder 내부구조
Fig 4. Proposed butterfly structure (a) overall block diagram, (b) multiplier block, (c) adder block.

식 (14)에서 실수부와 허수부를 각각 정리하면 다음 식과 같다.

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

지금까지 유도한 식 (13)과 식 (15)를 사용하여 그림 4와 같은 butterfly를 제안한다. 그림 4에서 (a)는 전체 블록도이고 (b)와 (c)는 각각 multiplier와 adder의 내부 구조이다.

Ⅲ. 결 론

이 논문에서는 DIT Radix-4 FFT 알고리즘의 구현을 위한 효율적인 butterfly 구조를 제안한다. 먼저 DIT Radix-4 SFG를 설명하고 이 SFG의 butterfly를 효율적으로 구현하기 위한 multiplier 구조와 adder 구조를 제안한다. 제안된 구조는 DIF와 비교하여 사용된 곱셈기와 덧셈기의 수가 같으므로 순차적으로 FFT 출력이 요구되는 시스템에서 널리 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. VLSI Syst.*, vol. 6, no. 1, pp. 18-30, Mar. 1998.

[2] J. Lee and H. Lee, “A high-Speed two-Parallel Radix-24 FFT/IFFT processor for MB-OFDM UWB Systems,” *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, vol. E91-A, no. 4, pp. 1206-1211, Apr. 2008.

[3] H. J. Kim and Y. B. Jang, “Low-area FFT processor structure using radix-42 algorithm,” *J. Inst. Electron. Eng. Korea (IEEK)*, vol. 49-SD, no. 3, pp. 8-14, Mar. 2012.

[4] E. J. Kim and M. H. Sunwoo, “High speed 8-parallel FFT/IFFT processor using efficient pipeline architecture and scheduling scheme,” *J. Korea Inform. Commun. Soc. (KICS)*, vol. 36, no. 3, pp. 175-182, Mar. 2011.

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