논문 2006-43SP-1-12

Distributed Arithmetic을 사용한 OFDM용 저전력 Radix-4 FFT 구조

(Low-power Radix-4 FFT Structure for OFDM using Distributed Arithmetic)

장 영 범*, 이 원 상**, 김 도 한**, 김 비 철**, 허 은 성**

(Young-Beom Jang, Won-Sang Lee, Do-Han Kim, Bee-Chul Kim, and Eun-Sung Hur)

요 약

이 논문에서는 64-Point FFT Radix-4 알고리즘을 DA(Distributed Arithmetic)연산을 이용하여 효율적으로 나비연산 구조를 설계할 수 있음을 보였다. 기존의 convolution 연산에 사용되어 왔던 DA연산이 FFT 나비연산의 트위들 계산에도 효과적으로 사용될 수 있음을 보였다. 제안된 DA 나비연산 구조를 Verilog HDL 코딩으로 구현한 결과, 기존의 승산기를 사용한 나비연산 구조와 비교하여 61.02%의 cell area 감소 효과를 보였다. 또한 제안된 나비연산 구조를 파이프라인 구조에 적용하여 지연변환기와 함께 사용한 전체 64-point Radix-4 FFT 구조의 Verilog-HDL 코딩을 기존의 승산기를 사용한 구조의 코딩과 비교한 결과, 46.1%의 cell area 감소효과를 볼 수 있었다. 따라서 제안된 FFT 구조는 DMB용 OFDM 모뎀과 같은 큰 크기의 FFT에 효율적으로 사용될 수 있는 구조가 될 것이다.

Abstract

In this paper, an efficient butterfly structure for Radix-4 FFT algorithm using DA(Distributed Arithmetic) is proposed. It is shown that DA can be efficiently used in twiddle factor calculation of the Radix-4 FFT algorithm. The Verilog-HDL coding results for the proposed DA butterfly structure show 61.02% cell area reduction comparison with those of the conventional multiplier butterfly structure. Furthermore, the 64-point Radix-4 pipeline structure using the proposed butterfly and delay commutators is compared with other conventional structures. Implementation coding results show 46.1% 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, Distributed Arithmetic, Twiddle factor

I. 서 론

DMB(Digital Multimedia Broadcasting)의 상용화 속도가 빨라짐에 따라 단말기용 MODEM SoC(System on a Chip)의 저전력 구현에 대한 연구가 활발히 진행되고 있다. DMB용 MODEM SoC는 크게 FFT(Fast Fourier Transform) 블록, Interpolation/decimation 필터 블록, 비터비 블록, 변복조 블록, 등화기 블록 등으로 구성된다. DMB와 같은 고속 멀티미디어 시스템에서는 대역효율성이 우수한 OFDM(Orthogonal Frequency Division Multiplexing) 방식을 사용하고 있으며, 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, DMB용 OFDM에서는 2048 point의 FFT를 필요로 하므로 FFT 블록의 구현 비용

* 정회원, 상명대학교 정보통신공학과
(College of Engineering, Sangmyung University)
** 학생회원, 상명대학교 컴퓨터정보통신공학과
(Graduate School, Sangmyung University)
※ 본 논문은 한국소프트웨어진흥원의 IT SoC 핵심 설계인력양성사업으로 수행한 연구결과입니다.
접수일자 : 2005년11월22일, 수정완료일 : 2005년12월30일

과 전력소모를 줄이는 것이 핵심사항이라고 할 수 있다. 지상파 DMB용 OFDM에서는 246 μs 동안에 2048 point FFT를 수행하여야 하므로 높은 처리율을 갖는 Radix-4 FFT 알고리즘이 주로 사용된다. OFDM용 FFT 구조로는, 단일버터플라이연산자 구조[1], 파이프라인 구조[2], 병렬구조[3] 등의 여러 구조가 제안되었다. 본 논문에서는 Radix-4 FFT 알고리즘 기반의 파이프라인 구조에 속하는 저전력 FFT 구조를 제안한다. 제안된 구조는 DA(Distributed Arithmetic) 방식을 사용하여 지금까지 사용되는 FFT 구조보다 구현비용을 줄일 수 있음을 보이게 된다. 본문 Ⅱ장에서는 Radix-4 알고리즘과 DA 연산에 대하여 살피고, Ⅲ장에서 DA 연산을 적용한 저전력 Radix-4 FFT 구조를 제안하였다. Ⅳ장에서는 제안된 구조를 Verilog HDL 코딩으로 구현하여 처리속도와 게이트 수를 기존 구조들과 비교하여 제안 구조의 효율성을 분석하였다.

Ⅱ. Radix-4 FFT 알고리즘과 Distributed Arithmetic 연산

2.1 Radix-4 DIF의 FFT 알고리즘

N-Point의 DFT식은 다음과 같다.

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

위의 (1) 식을 Radix-4 FFT 알고리즘을 사용하기위하여 Radix-4 DIF(Decimation In Frequency)의 FFT 알고리즘으로 나타내면 다음과 같다.

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

식 (2)에서 보듯이 N-Point의 DFT는 4개의 N/4-Point DFT로 분해될 수 있다. 또한 각각의 N/4-Point DFT를 수행하기 위해서는 먼저 덧셈연산과 복소 곱셈연산이 필요함을 알 수 있다. 즉, 위의 식에서

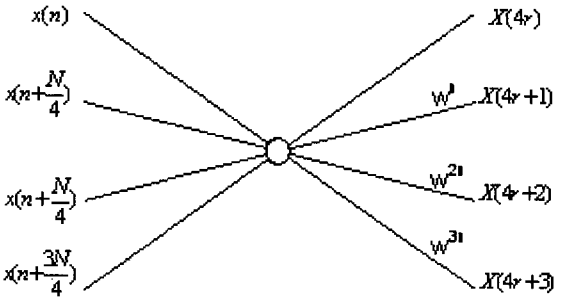


그림 1. Radix-4 FFT 나비연산 구조
Fig. 1. Radix-4 FFT Butterfly structure.

4개의 항을 더하기 위하여 나비연산기를 사용하며, twiddle factor W_N^0, W_N^n, W_N^{2n}, W_N^{3n}을 곱하기 위하여 복소 곱셈기를 사용한다. 이와 같은 나비연산과 복소곱셈 연산을 행렬을 사용하여 나타내면 다음 식과 같다.

$$\begin{bmatrix}X(0,q)\\X(1,q)\\X(2,q)\\X(3,q)\end{bmatrix}=\begin{bmatrix}1&1&1&1\\1&-j&-1&j\\1&-1&1&-1\\1&j&-1&j\end{bmatrix}\begin{bmatrix}W_N^0F(0,q)\\W_N^qF(1,q)\\W_N^{2q}F(2,q)\\W_N^{3q}F(3,q)\end{bmatrix} \tag{3}$$

위 결과 식을 입력단의 값과 그 입력 값에 곱해지는 복소곱셈 계수들을 사용하여 Radix-4 버터플라이 구조로 나타내면 그림 1과 같다.

2.2 Distributed Arithmetic 연산

이 절에서는 Distributed Arithmetic의 연산 원리를 알아본다. 원래 DA 연산은 필터의 곱셈연산을 승산기를 사용하지 않고 ROM과 가산기만으로 구성하는 저전력 필터 구현방식으로 제안되었다[4]. 다음의 4탭 필터를 통하여 DA 연산의 원리를 알아본다.

$$y=A_1x_1+A_2x_2+A_3x_3+A_4x_4 \tag{4}$$

식 (4)를 DA 연산방식을 사용하여 구현하면 그림 2와 같다.

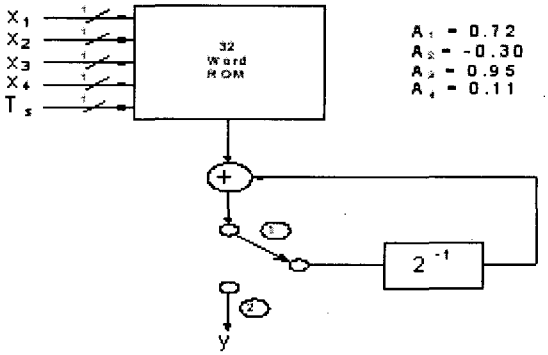


그림 2. 분산 연산 필터 구조
Fig. 2. Distributed Arithmetic Filter.

표 1. 룩업 테이블
Table 1. Look-Up Table.

	Input code T_s B_{1n} B_{2n} B_{3n} B_{4n}	32-Word Memory Contents
1 < n < N-1	0 0 0 0 0	0
	0 0 0 01	A_4 = 0.11
	0 0 0 10	A_3 = 0.95
	0 0 0 11	A_3 +A_4 = 1.06
	0 0 1 00	A_2 = - 0.30
	0 0 1 01	A_2 + A_4 = - 0.19
	0 0 1 10	A_2 + A_3 = 0.65
	0 0 1 11	A_2 + A_3 +A_4 = 0.75
	0 1 0 00	A_1 = 0.72
	0 1 0 01	A_1 + A_4 = 0.83
	0 1 0 10	A_1 + A_3 = 1.67
	0 1 0 11	A_1 + A_3 + A_4 =1.78
	0 1 1 00	A_1 + A_2 =0.42
	0 1 1 01	A_1 + A_2 + A_4 =0.53
	0 1 1 10	A_1 + A_2 + A_3 =1.37
	0 1 1 11	A_1 + A_2 + A_3 + A_4 = 1.48
n = 0	1 0 0 0 0	0
	1 0 0 01	-A_4 = -0.11
	1 0 0 10	-A_3 = -0.95
	1 0 0 11	-(A_3 +A_4)= 1.06
	1 0 1 00	-(A_2)= 0.30
	1 0 1 01	-(A_2 + A_4)= 0.19
	1 0 1 10	-(A_2 + A_3)= -0.65
	1 0 1 11	-(A_2 + A_3 +A_4)= -0.75
	1 1 0 00	-A_1 = -0.72
	1 1 0 01	-(A_1 + A_4)= -0.83
	1 1 0 10	-(A_1 + A_3)= -1.67
	1 1 0 11	-(A_1 + A_3 + A_4)= -1.78
	1 1 1 00	-(A_1 + A_2)= -0.42
	1 1 1 01	-(A_1 + A_2 + A_4)= -0.53
	1 1 1 10	-(A_1 + A_2 + A_3)= -1.37
	1 1 1 11	-(A_1 + A_2 + A_3 + A_4)= -1.48

그림 2에서 보듯이 비트로 표현되는 입력 신호의 LSB부터 병렬로 ROM에 입력시켜 그 값을 ROM의 address 값으로 사용한다. 32-Word ROM에 저장되어 있는 값들은 미리 계산된 모든 경우의 값이며 표 1과 같다.

표 1의 ROM 값들은 비트 입력신호에 의한 address에 의해 출력되며, 그 후에 가산기와 shift 연산기로 입력신호의 비트 수만큼 반복하여 연산하면 계산이 끝나게 된다. 이와 같은 DA 연산은 다음과 같은 식으로 나타낼 수 있다.

$$y=\sum_{n=1}^{N-1}[\sum_{k=1}^{4}A_k b_{kn}]2^{-1}+\sum_{k=1}^{4}A_k(-b_{k0}) \quad (5)$$

식 (5)에서 N은 입력신호의 비트 정세도를 나타낸다. 즉, 16비트의 입력신호가 입력되는 경우에 N=16이 된다. 따라서 16번의 가산기 연산을 반복함으로써 최종 출력신호가 구해진다.

Ⅲ. 제안된 Radix-4 FFT용 나비연산 구조

이 절에서는 DA 방식을 응용하여 저전력으로 동작하는 Radix-4 FFT용 나비연산 구조를 제안한다. DFT와 비슷한 알고리즘으로 변환이 수행되는 DCT나

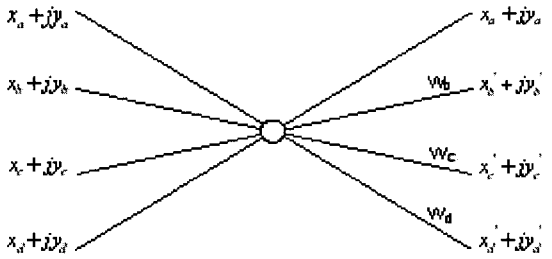


그림 3. 허수부를 고려한 FFT Radix-4 나비연산 구조
Fig. 3. Distributed Arithmetic Filter for image part.

MDCT의 경우는 입력 값과 계수가 모두 실수로 이루어져 있으나 DFT의 경우에는 입력 값과 계수 모두가 실수부와 허수부로 구성된 복소수이므로 실제 연산에 있어 고려하여야 할 사항이 많아진다. 직각형 복소수로 표현된 입력신호와 Twiddle factor를 사용하여 나비연산 블록도를 나타내면 그림 3과 같다.

그림 3에서 보듯이 Radix-4 나비연산의 각각 4개의 입력과 4개의 출력을 모두 복소수로 나타내었으며, Twiddle factor W^n, W^{2n}, W^{3n}도 모두 직각형 복소수로 다음과 같이 나타낼 수 있다.

$$W^n=e^{-j\frac{2\pi n}{N}}=\cos\frac{2\pi n}{N}-j\sin\frac{2\pi n}{N}=C_b+j(-S_b)$$
$$W^{2n}=e^{-j\frac{2\pi 2n}{N}}=\cos\frac{4\pi n}{N}-j\sin\frac{4\pi n}{N}=C_c+j(-S_c)$$
$$W^{3n}=e^{-j\frac{2\pi 3n}{N}}=\cos\frac{6\pi n}{N}-j\sin\frac{6\pi n}{N}=C_d+j(-S_d) \quad (6)$$

그림 3과 같은 4개의 출력을 계산하는 나비연산은 DA 연산을 사용하면 용이하다. 왜냐하면 DA 연산은 4개 이하의 MAC(Multiplication and Accumulation) 연산에 어울리기 때문이다. 5개 이상의 MAC 연산은 ROM의 크기가 커지므로 DA 연산의 효과가 많이 감소된다. 따라서 이 논문에서 Radix-4용 나비연산 알고리즘의 구현에 DA 연산 방식을 적용하였다.

이제 나비연산을 통하여 가산 연산과 Twiddle factor가 연산되어진 후 출력되는 각각의 출력 결과 값을 살펴보도록 한다. 먼저 그림 3에서 x_a'과 y_a'은 다음식과 같다.

$$x_a' = x_a + x_b + x_c + x_d$$
$$y_a' = y_a + y_b + y_c + y_d \quad (7)$$

식 (7)에서 보듯이 나비연산의 첫 출력 값에는 Twiddle factor가 고려되지 않는다. 그러나 그림 3의 나비연산의 두 번째 출력 값을 구하는 과정에는 Twiddle factor 복소수를 입력 값에 곱하는 연산이 포함된다. 따

라서 Twiddle factor 복소수를 곱한 두 번째 출력 값은 다음과 같이 나타낼 수 있다.

$$\begin{aligned} x_b'+jy_b' &= \{(x_a+jy_a)-j(x_b+jy_b)-(x_c+jy_c)+j(x_d+y_d)\} \\ &\quad \{C_b+j(-S_b)\} \\ &= \{(x_a+y_b-x_c-y_d)+j(y_a-x_b-y_c+x_d)\} \\ &\quad \{C_b+j(-S_b)\} \end{aligned} \quad (8)$$

식 (8)로부터 x_b'과 y_b'은 다음과 같이 나타낼 수 있다.

$$\begin{aligned} x_b' &= (x_a+y_b-x_c-y_d)C_b-(y_a-x_b-y_c+x_d)(-S_b) \\ &= x_1C_b-x_2(-S_b) \\ y_b' &= (y_a-x_b-y_c+x_d)C_b+(x_a+y_b-x_c-y_d)(-S_b) \\ &= x_2C_b+x_1(-S_b) \end{aligned} \quad (9)$$

다음으로 세 번째 출력 값을 구하면 다음과 같다.

$$\begin{aligned} x_c'+jy_c' &= \{(x_a+jy_a)-(x_b+jy_b)+(x_c+jy_c)-(x_d+y_d)\} \\ &\quad \{C_c+j(-S_c)\} \\ &= \{(x_a-x_b+x_c-x_d)+j(y_a-y_b+y_c-y_d)\} \\ &\quad \{C_c+j(-S_c)\} \end{aligned} \quad (10)$$

식 (10)으로부터 x_c'과 y_c'은 다음과 같이 나타낼 수 있다.

$$\begin{aligned} x_c' &= (x_a-x_b+x_c-x_d)C_c-(y_a-y_b+y_c-y_d)(-S_c) \\ &= x_3C_c-x_4(-S_c) \\ y_c' &= (y_a-y_b+y_c-y_d)C_c+(x_a-x_b+x_c-x_d)(-S_c) \\ &= x_4C_c+x_3(-S_c) \end{aligned} \quad (11)$$

마지막으로 네 번째 나비연산의 출력 값을 구하면 다음과 같다.

$$\begin{aligned} x_d'+jy_d' &= \{(x_a+jy_a)+j(x_b+jy_b)-(x_c+jy_c)-j(x_d+y_d)\} \\ &\quad \{C_d+j(-S_d)\} \\ &= \{(x_a-y_b-x_c+y_d)+j(y_a+x_b-y_c-x_d)\} \\ &\quad \{C_d+j(-S_d)\} \end{aligned} \quad (12)$$

식 (12)로부터 x_d'과 y_d'은 다음과 같이 나타낼 수 있다.

$$\begin{aligned} x_d' &= (x_a-y_b-x_c+y_d)C_d-(y_a+x_b-y_c-x_d)(-S_d) \\ &= x_5C_d-x_6(-S_d) \\ y_d' &= (y_a+x_b-y_c-x_d)C_d+(x_a-y_b-x_c+y_d)(-S_d) \\ &= x_6C_d+x_5(-S_d) \end{aligned} \quad (13)$$

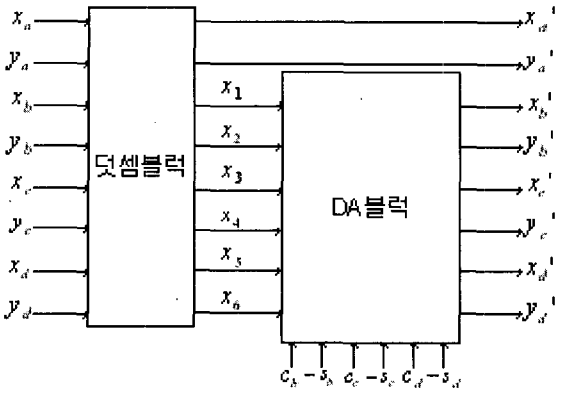


그림 4. 제안된 나비연산 구조
Fig. 4. Proposed butterfly structure.

이 절에서는 식(7), (9), (11), (13)을 그림 4와 같이 덧셈블록과 DA블록으로 나누어 구성하는 구조를 제안하였다. 예를 들어 1024 point FFT에서는 Radix-4 알고리즘을 사용하면 $4^5=1024$이므로 5 스테이지의 연산이 필요하게 된다. 그리고 파이프라인 구조를 사용하면 각 스테이지의 버터플라이는 512번의 버터플라이 연산을 반복하여야 한다. 이와 같은 512번의 고속 버터플라이 연산을 위하여 그림 4와 같은 DA 방식을 사용한 곱셈 연산 구조가 효과적이다.

그림 4의 덧셈블록에서는 식(7), (9), (11), (13)의 덧셈연산을 수행한다. 즉 식 (7)의 x_a'과 y_a'을 계산하며, 식(9), (11), (13)의 $x_1, x_2, x_3, x_4, x_5, x_6$을 계산하도록 설계하였으며 그 덧셈블록의 세부 구조는 그림 5와 같다.

그림 5의 덧셈블록에서 출력되는 신호 중에서 x_a'과 y_a'은 최종 출력이 되며 $x_1, x_2, x_3, x_4, x_5, x_6$은 그림 4의 DA블록으로 입력되어 MUX의 어드레스로 사용된다.

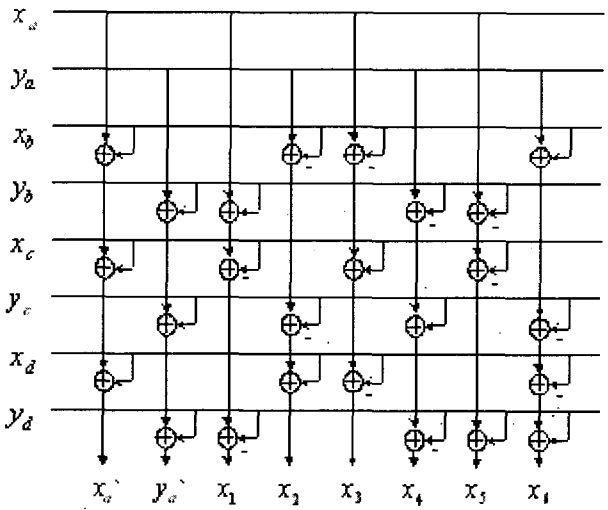


그림 5. 제안된 나비연산용 덧셈블록 구조
Fig. 5. Proposed adder block structure for butterfly.

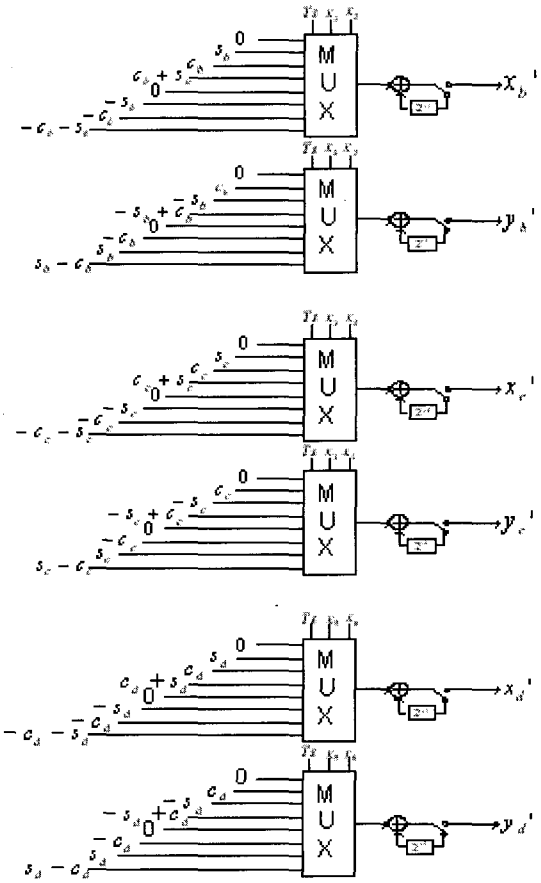


그림 6. 제안된 나비연산용 DA블록 구조
Fig. 6. Proposed DA block structure for butterfly.

제안된 DA블록은 그림 6과 같다.

제안된 그림 6의 구조에서 가장 위부터 b, c, d의 블록으로 구성하였다. 그림 6에서 보듯이 Ts, x1, x2는 8-input MUX의 어드레스로 사용되며 b 블록의 어드레스에 따른 MUX의 출력 값은 다음 표 2와 같다.

b용 DA블록에서는 표 2에서 보듯이 Cb와 −Sb를 받아들여서 Sb, −Sb, Cb, −Cb, Cb+Sb, −Cb−Sb, −Sb+Cb, Sb−Cb 등의 8개의 값들을 만들어내어야 하며, 이렇게 만들어진 8개의 값들은 b용 DA블록의 Ts, x1, x2의 LSB부터 어드레스로 동작하여 MUX의 입력 값들을 출력시킨다. 즉, LSB부터 MSB-1 비트까지는 $n\neq0$의 어드레스에 따라 출력하고, MSB 비트인 부호비트에서는 $n=0$의 어드레스에 따라서 출력한다. 예를 들어 x1과 x2가 16비트로 구성되어 있으면, LSB부터 15비트 까지는 $n\neq0$이므로 Ts=0의 어드레스에 따라 출력하고, MSB 비트에서는 $n=0$이므로 Ts=1의 어드레스에 따라 출력하도록 제어한다. 즉, Ts는 MSB에서만 1이 된다. 어드레스가 결정되면 1개의 MUX는 표 2의 xb에 따라 출력이 결정되며, 또 다른 1개의 MUX는 표 2의 yb에 따라 출력되도록 어드레스 디코더를 설계하였다. 이렇게 출력되는 16개의 값들은 1 비트씩 LSB로 쉬프트 되면서 새 값과 더해짐으로써 최종 출력 x_b'과 y_b'이 얻어진다.

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

	Ts	x1n	x2n	xb 출력	yb 출력
	0	0	0	0	0
	0	0	1	Sb	Cb
	0	1	0	Cb	−Sb
$n\neq0$	0	1	1	Cb+Sb	−Sb+Cb
	1	0	0	0	0
	1	0	1	−Sb	−Cb
$n=0$	1	1	0	−Cb	Sb
	1	1	1	−Cb−Sb	Sb−Cb

c용 DA블록과 d용 DA블록의 구조도 지금까지 설계한 b용 DA블록과 같은 구조를 사용하였으며 어드레스에 따른 MUX의 출력 값은 다음 표 3과 4와 같다.

지금까지 덧셈블록과 DA블록을 사용하여 저전력 버터플라이 구조를 제안하였다. 다음 절에서는 RTL 코딩을 통하여 제안한 구조의 효용성을 입증한다.

표 3. 출력 c용 어드레스별 MUX 출력 값
Table 3. MUX output values of address for c.

	Ts	x3n	x4n	xc 출력	yc 출력
	0	0	0	0	0
	0	0	1	Sc	Cc
$n\neq0$	0	1	0	Cc	−Sc
	0	1	1	Cc+Sc	−Sc+Cc
	1	0	0	0	0
	1	0	1	−Sc	−Cc
$n=0$	1	1	0	−Cc	Sc
	1	1	1	−Cc−Sc	Sc−Cc

표 4. 출력 d용 어드레스별 MUX 출력 값
Table 4. MUX output values of address for d.

	Ts	x5n	x6n	xd 출력	yd 출력
	0	0	0	0	0
	0	0	1	Sd	Cd
	0	1	0	Cd	−Sd
$n\neq0$	0	1	1	Cd+Sd	−Sd+Cd
	1	0	0	0	0
	1	0	1	−Sd	−Cd
$n=0$	1	1	0	−Cd	Sd
	1	1	1	−Cd−Sd	Sd−Cd

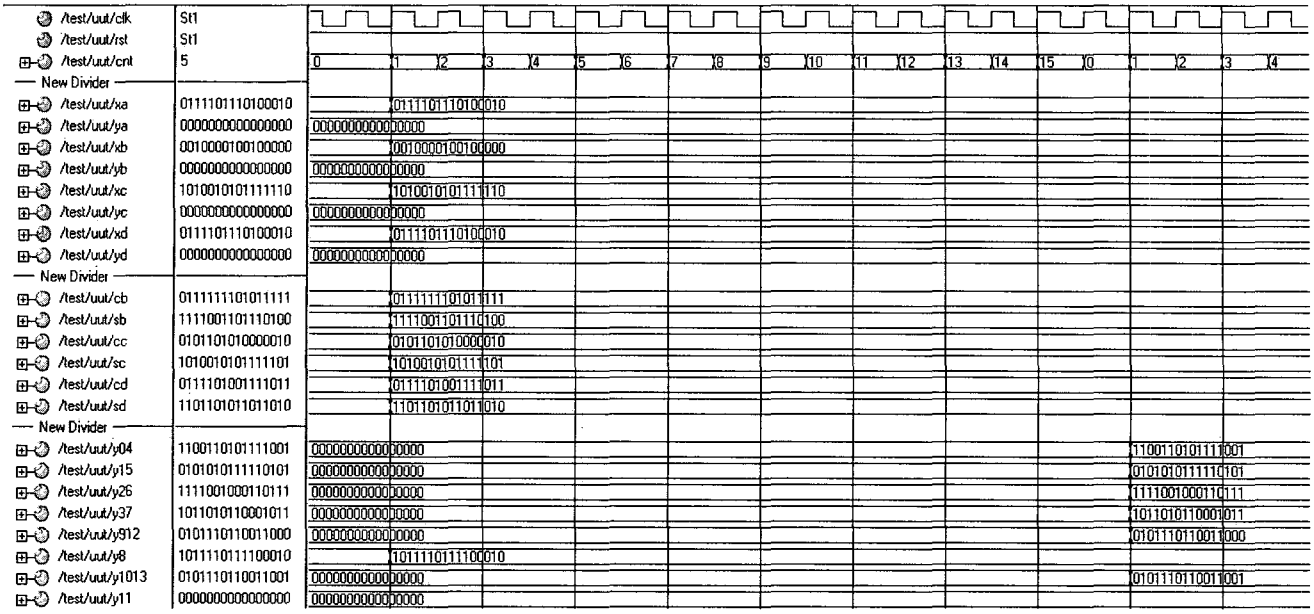


그림 7. DA를 적용한 64-point FFT Radia-4구조의 verilog HDL simulation 결과
Fig 7. verilog HDL simulation result for 64-point FFT Radix-4 algorithm using DA.

Ⅳ. 실험 및 고찰

제안된 구조의 구현면적을 Verilog-HDL 코딩을 통하여 64-point FFT의 구현면적을 비교하였다. 제안 구조에 대하여 Xilinx ISE 6.2i로 verilog코딩을 하여 FPGA 기반으로 타이밍을 검증하였으며, 시뮬레이션 결과 값은 그림 7과 같다. 검증을 위한 64개의 입력신호로서 cosine신호를 샘플링하여 실수의 입력신호를 사용하였다. 또한 각각의 입력값을 16비트 2진수로 변환하여 입력하였다. 위 그림 7에서 출력값 y_8과 y_{11}은 DA블록을 거치지 않고 덧셈블록 만을 통과하여 나오는 x_a'와 y_a'의 계산 값을 보여주며 그 외에 y_{04}, y_{15}, y_{26}, y_{37}, y_{912}, y_{1013}들은 입력 값들이 덧셈블록 통과 후에 DA블록의 연산과정을 거쳐 마지막 입력비트까지 계산된 후 출력된 값들을 보여주고 있다.

표 5에 제안 구조에 대한 synopsis 논리합성 결과를 나타내었다. 제안구조와 비교 구조 모두 64-Point FFT Radix-4 알고리즘의 나비연산 블록에 대한 논리합성 결과이다. 비교된 나비연산 구조는 그림 4의 구조에서 DA 블록을 사용하지 않고, 대신 승산기블록으로 대체한 구조에 대한 논리합성 결과이다. 즉, 표 5의 승산기 구조는 그림 4의 구조를 따르되 DA승산기를 사용하지 않고 기존의 일반 승산기를 사용하여 합성한 결과를 보여주고 있다. 표 5에서 보듯이 제안된 DA승산기를 사용한 결과 cell area를 46,721에서 18,213으로 줄일 수 있었다. 이는 61.02%의 감소효과를 나타내고 있다. 표 5는 64-point FFT에 대한 논리 합성이므로, 만일 2048-point FFT의 경우에는 감소효과가 더욱 크게 됨을 알 수 있다.

표 5는 64-point FFT의 전체 구조를 합성한 것이 아니고, 버터플라이/트위들 블록에 대한 논리 합성이다. 따라서 파이프라인 방식의 지연변환기를 사용하는 전체 구조의 cell area를 비교해보기로 한다. 일반적으로 Radix-4의 64-point FFT는 3 스테이지로 구성되며, 스테이지 사이에 지연변환기가 사용되므로 2개의 지연변환기가 사용된다. 제안된 버터플라이/트위들 블록은 각 스테이지에서 16번의 버터플라이 연산을 반복한다. 제안 구조를 기존의 3개의 구조와 비교하였으며, 논리 합성 결과는 표 6과 같다.

표 6의 구조 1은 Radix-4 알고리즘을 일반 승산기를 사용하여 구현한 MDC(Multi-path Delay Commutator)

표 5. 제안된 나비연산 구조와 기존 구조의 논리합성 결과(64-point FFT)
Table 5. Logic synthesis results for the proposed butterfly and conventional structures(64-point FFT).

구분	블록명	cell area	블록수	sub-total cell area	total cell area
제안 구조	덧셈블록	2,769	3	8,307	18,213
	DA블록	4,953	2	9,906	
승산기 구조	덧셈블록	2,131	3	6,393	46,721
	승산기블록	20,164	2	40,328	

표 6. Radix-4 64-point FFT 제안구조와 기존 구조들의 Cell area 비교

Table 6. Cell areas for proposed and conventional Radix-4 64-point FFT structures.

구분	구조 1	구조 2	구조 3	제안구조
버터플라이/트위들	44,198	27,653	46,721	18,213
지연변환기	12,170	11,853	12,170	12,170
총 cell area	56,368	39,506	58,891	30,383
상대구현 면적	100%	70.9%	104.5%	53.9%

구조인 [5]에서 사용된 논리합성 결과를 참조한 것이다. 구조 2는 [5]에서 제안한 Radix-4와 Radix-2를 함께 사용하는 알고리즘에 대한 결과를 참조한 것이다. 이 구조는 64-point FFT의 구현에 지연변환기를 3개 사용하고 있다. 구조 3은 그림 4의 이 논문에서 제안된 구조를 따르되 DA를 사용하지 않고 일반승산기를 사용한 결과이다.

표 6의 구조 3과 제안구조의 지연변환기 결과 값들은 구조 1의 결과 값들을 그대로 사용하였다. 표 6에서 보듯이 제안된 64-point Radix-4 FFT 구조는 구조 1과 비교하여 cell area가 46.1% 감소됨을 알 수 있다.

V. 결 론

이 논문에서는 64-point Radix-4 알고리즘의 저전력 파이프라인 구조를 제안하였다. 파이프라인 FFT 구조는 버터플라이/트위들 블록과 지연변환기로 구성되는데, 이 논문은 버터플라이/트위들 블록을 덧셈블록과 DA블록으로 구현하는 저전력 구조를 제안하였다. 제안된 버터플라이/트위들 블록은 기존의 일반 승산기를 사용하는 버터플라이/트위들 블록과 비교하여 61.02%의 cell are 감소효과를 볼 수 있었다. 또한 지연변환기를 포함한 제안된 64-point FFT 블록은 기존의 64-point FFT 블록과 비교하여 46.1%의 cell area 감소효과를 나타내었으며 2048-point FFT와 같은 큰 크기의 FFT에서는 더욱 cell area 감소효과를 나타낼 것이다. 따라서 제안된 DA방식의 FFT 구조는 DMB용 OFDM 모뎀과 같은 큰 크기의 FFT를 요구하는 시스템에서 사용될 수 있는 효율적인 구조이다.

참 고 문 헌

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

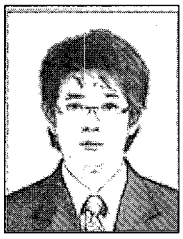
[5] 정윤호, 김재석, "고속 멀티미디어 통신시스템을 위한 효율적인 FFT 알고리즘 및 하드웨어 구현", 전자공학회논문지 제41권 SD편 제3호, pp. 55-64, 2004년 3월.

[6] R. Van Nee and R. Prasad, *OFDM for Wireless Multimedia Communication*, Artech House, pp. 33-50, 2000.

저 자 소 개

장 영 범 (정회원)

1981년 연세대학교 전기공학과 공학사
1990년 Polytechnic University 대학원 공학석사
1994년 Polytechnic University 대학원 공학박사
1981년~1999년 삼성전자 System LSI 사업부 수석연구원
2000년~2002년 이화여자대학교 정보통신학과 연구교수
2002년~현재 상명대학교 정보통신공학과 교수
<주관심분야 : 통신신호처리, 비디오신호처리, SoC 설계>



김 비 철 (학생회원)

2006년 상명대학교 정보통신공학과 졸업 예정 (공학사)
2006년 상명대학교 대학원 컴퓨터정보통신공학과 입학 예정
<주관심분야 : 통신신호처리, SoC 설계>

허 은 성 (학생회원)

2006년 상명대학교 정보통신 공학과 졸업 예정 (공학사)
2006년 상명대학교 대학원 컴퓨터정보통신공학과 입학 예정
<주관심분야 : 통신신호처리, SoC 설계>

이 용 택 (학생회원)

2004년 한국외국어대학교 전자정보공학부 학사
2004년 학국외국어대학교 전자정보공학과 석사 과정
<주관심분야 : RF CMOS 소자 모델링>

김 도 한 (학생회원)

2006년 상명대학교 정보통신공학과 졸업 예정 (공학사)
2006년 상명대학교 대학원 컴퓨터정보통신공학과 입학 예정
<주관심분야 : 통신신호처리, SoC 설계>