논문 2012-49SD-3-2

Radix-4^2 알고리즘을 사용한 저면적 FFT 프로세서 구조

(Low-area FFT Processor Structure using Radix-4^2 Algorithm)

김 한 진*, 장 영 범**

(Han Jin Kim and Young Beom Jang)

요 약

이 논문에서는 Radix-4^2 알고리즘을 사용한 저면적 FFT 구조를 제안한다. 큰 point의 FFT는 여러 개의 직렬연결 스테이지로 구성되는데, Radix-4^2 알고리즘을 사용하면 매 2 스테이지마다 곱셈 종류의 수가 3인 스테이지가 생긴다. 이 사실을 이용하여 곱셈 연산 종류의 수가 3인 스테이지의 구현 면적을 줄이는 구조를 제안하였다. 예를 들면 4096-point FFT는 6개의 스테이지로 구성되는데 Radix-4^2 알고리즘을 사용하면 3개의 스테이지가 곱셈연산 종류의 수가 3이다. 이 3개의 스테이지의 곱셈연산 하드웨어는 CSD(Canonic Signed Digit) 계수 방식과 CSS(Common Sub-expression Sharing) 기술을 사용하여 구현면적을 감소시킬 수 있었다. 제안된 방식을 사용하여 256-point FFT 구조를 설계하여 Verilog-HDL 코딩하였다. 또한 tsmc 0.18 μm CMOS 라이브러리를 사용하여 합성하여 구현한 결과 1.971mm^2의 cell area를 얻었다. 이와 같은 합성 결과는 기존 구조와 비교하여 약 23%의 cell area 감소 효과를 보였다.

Abstract

In this paper, a low-area FFT structure using Radix-4^2 algorithm is proposed. The large point FFT structure consists of cascade connection of the many stages. In implementation of large point FFT using Radix-4^2 algorithm, stages which number of different coefficients are only 3 appear in every 2 stages. For example, in the 4096-point FFT, the stages that number of different coefficients are 3 appear in stage 1, 3, and 5 among 6 stages. Multiplication block area of these 3 stages can be reduced using CSD(Canonic Signed Digit) and common sub-expression sharing techniques. Using the proposed structure, the 256-point FFT is implemented with the Verilog-HDL coding and synthesized by 1.971mm^2 cell area in tsmc 0.18μm CMOS library. This result shows 23% cell area reduction compared with the conventional structure.

Keywords : FFT, Radix-4^2, Radix-4, CSD, Common Sub-expression Sharing

Ⅰ. 서 론

FFT(Fast Fourier Transform) 블록은 OFDM (Orthogonal Frequency Division Multiplexing) 통신 방식의 MODEM SoC(System on Chip)에서 가장 큰 구현 면적을 차지하는 블록이다. 특히 FFT의 구현에서, 복소 곱셈연산 블록을 효과적으로 구현하는 구조가 많이 연구되고 있다. 복소 곱셈연산 블록은 Modified Booth 곱셈기를 사용하는 방법이 가장 널리 사용되고 있으며, CORDIC (COordinate Rotation DIgital Computer) 곱셈기를 사용하여 구현하는 방식도 연구되었다.[1~2] 이와 더불어 CSD(Canonic Signed Digit) 곱셈기를 사용하여 복소 곱셈연산을 수행하는 방식도 연구되었다.[3] CSD 곱셈기 방식은 수를 CSD형으로 표기한 후에 이를 덧셈기와 쉬프트를 사용하여 구현하는 저면적 구현방법으로서 디지털 필터의 저면적 구현에도 널리 사용되고 있다.[4~5] FFT의 복소 곱셈연산 블록에서도 CSD 곱셈기를 이용한 방식이 많이 발표되었다.[6~9]

이 논문에서는 큰 point의 FFT에 대한 저면적 FFT

* 학생회원, 상명대학교 컴퓨터정보통신공학과
(Graduate School, Sangmyung University)
** 정회원-교신저자, 상명대학교 정보통신공학과
(College of Engineering, Sangmyung University)
접수일자: 2011년8월26일, 수정완료일: 2012년2월29일

구조를 제안한다. 큰 point FFT는 많은 스테이지의 직렬 연결로 구성되는데, Radix-4^2 알고리즘을 사용하면 매 두 스테이지 마다 3 종류의 곱셈만 사용되는 스테이지가 발생한다. 따라서 이와 같은 곱셈연산의 종류가 3개인 스테이지는 곱셈연산 블록을 저면적으로 설계할 수 있음을 보인다.

이 논문의 II장에서는 Radix-4^2 알고리즘에 대하여 알아보고 III장에서는 저면적 FFT 구조를 제안한다. IV장에서는 구현을 통하여 제안 구조의 효용성을 입증하며 V장에서 결론을 맺는다.

II. Radix-4^2 알고리즘의 선택

OFDM용 FFT 블록에서는 Radix 4 계열의 파이프라인 구조가 널리 사용되고 있다. 1024-point FFT의 경우, Radix 4 계열의 알고리즘을 사용하면 5개의 스테이지로 구성되며 마지막 스테이지를 제외하고 앞의 4개 스테이지에서 곱셈연산이 필요하다. 따라서 각 스테이지에서는 곱셈연산을 수행하는 곱셈연산 하드웨어가 필요하게 된다. 표 1에서 256, 1024, 4096-point FFT에 대한 스테이지 별 곱셈 종류의 수를 비교하였다. 이 표 1에서 비교한 구조는 Radix-4와 Radix-4^2 알고리즘의 구조이다.

표 1에서 보듯이 Radix-4 알고리즘의 경우에는 스테이지가 진행되어 갈수록 곱셈종류의 수가 감소되는 반면에 Radix-4^2 알고리즘은 매 2 스테이지마다 곱셈 종류의 수가 3인 스테이지가 발생한다. 예를 들면 Radix-4^2 알고리즘을 사용한 4096-point FFT의 경우에 전체 6개의 스테이지 중에서 3개의 스테이지에서 곱셈 종류의 수가 3인 것을 알 수 있다. 즉 첫 번째, 세 번째, 다섯 번째 스테이지에서 곱셈 종류의 수가 각각 3개이다. 이 논문에서는 이 3개의 스테이지에서 곱셈연산 블록의 구현 면적을 줄이는 방법, 즉 곱셈 연산의 종류가 3인 스테이지의 곱셈 연산 블록을 작게 만들 수 있는 방법을 제안한다. 제안된 방법은 다음의 N-point의 DFT의 정의로부터 시작한다.

표 1. 스테이지 별 계수의 수 비교
Table 1. Number of coefficients in each stage.

point	Radix	1st stage	2nd stage	3rd stage	4th stage	5th stage	6th stage
256	4	63	15	3	0		
	4^2	3	63	3	0		
1024	4	255	63	15	3	0	
	4^2	3	255	3	15	0	
4096	4	1023	255	63	15	3	0
	4^2	3	1023	3	63	3	0

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

식 (1)에서 Twiddle Factor는 $W_N^{nk}=e^{-j2\pi nk/N}$이고 n은 시간 index, k는 주파수 index이다. n과 k를 CFA(Common Factor Algorithm)에 의하여 다음과 같이 3차원으로 분리하여 전개한다.

$$\begin{aligned} n&=\frac{N}{4}n_1+\frac{N}{16}n_2+n_3 \quad \left\{0\le n_1,n_2\le 3,\ 0\le n_3\le \frac{N}{4}-1\right\}\\ k&=k_1+4k_2+16k_3 \quad \left\{0\le k_1,k_2\le 3,\ 0\le k_3\le \frac{N}{4}-1\right\}\end{aligned} \tag{2}$$

식 (1)에 식 (2)를 대입하면 Radix-4^2의 FFT를 다음과 같이 나타낼 수 있다.

$$\begin{aligned} &X(k_1+4k_2+16k_3) \qquad (3)\\ &=\sum_{n_3=0}^{\frac{N}{16}-1}\sum_{n_2=0}^{3}\sum_{n_1=0}^{3}x\left(\frac{N}{4}n_1+\frac{N}{16}n_2+n_3\right)W_N^{\left(\frac{N}{4}n_1+\frac{N}{16}n_2+n_3\right)(k_1+4k_2+16k_3)}\\ &=\sum_{n_3=0}^{\frac{N}{16}-1}\sum_{n_2=0}^{3}\sum_{n_1=0}^{3}x\left(\frac{N}{4}n_1+\frac{N}{16}n_2+n_3\right)W_N^{\frac{N}{4}n_1k_1}W_N^{\left(\frac{N}{16}n_2+n_3\right)(k_1+4k_2+16k_3)}\end{aligned}$$

위의 식에서 첫 번째 스테이지의 4-point FFT는 다음과 같이 $B(\ )$로 정의한다.

$$B\left(\frac{N}{16}n_2+n_3,k_1\right)=\sum_{n_1=0}^{3}x\left(\frac{N}{4}n_1+\frac{N}{16}n_2+n_3\right)W_N^{\frac{N}{4}n_1k_1} \tag{4}$$

식 (3)의 Twiddle Factor는 다음과 같이 간략화될 수 있다.

$$W_N^{\left(\frac{N}{16}n_2+n_3\right)(k_1+4k_2+16k_3)}=W_N^{\frac{N}{16}n_2k_1}W_N^{\frac{N}{4}n_2k_2}W_N^{n_3(k_1+4k_2)}W_{\frac{N}{16}}^{n_3k_3} \tag{5}$$

식 (4)와 (5)를 식 (3)에 대입하면 다음과 같은 최종식을 얻을 수 있다.

$$\begin{aligned} &X(k_1+4k_2+16k_3)=\\ &\sum_{n_3=0}^{\frac{N}{16}-1}\left\{\sum_{n_2=0}^{3}\left[B\left(\frac{N}{16}n_2+n_3,k_1\right)W_N^{\frac{N}{16}n_2k_1}\right]W_N^{\frac{N}{4}n_2k_2}W_N^{n_3(k_1+4k_2)}\right\}W_{\frac{N}{16}}^{n_3k_3}\end{aligned} \tag{6}$$

위의 식에서 [ ]의 괄호는 첫 번째 스테이지의 연산으로서 다음과 같이 정의한다.

$$P\left[\frac{N}{16}n_2+n_3,k_1\right]=\left(\sum_{n_1=0}^{3}x\left(\frac{N}{4}n_1+\frac{N}{16}n_2+n_3\right)W_N^{\frac{N}{4}n_1k_1}\right)W_N^{\frac{N}{16}n_2k_1}$$

$$=B\left(\frac{N}{16}n_2+n_3,k_1\right)W_N^{\frac{N}{16}n_2k_1} \tag{7}$$

식 (6)에서 { }의 괄호는 두 번째 스테이지의 연산으로서 다음과 같이 정의한다.

$$Q[n_3,k_1+4k_2]=\left(\sum_{n_2=0}^{3}P\left[\frac{N}{16}n_2+n_3,k_1\right]W_N^{\frac{N}{4}n_2k_2}\right)W_N^{n_3(k_1+4k_2)} \tag{8}$$

이와 같이 2개의 스테이지가 구성되면 나머지는 다음과 같이 $N/16$-point FFT로 나타낼 수 있다.

$$X(k_1+4k_2+16k_3)=\sum_{n_3=0}^{\frac{N}{16}-1}Q[n_3,k_1+4k_2]\,W_{\frac{N}{16}}^{n_3k_3} \tag{9}$$

식 (9)에 대하여 다시 Radix-4^2 알고리즘을 사용하여 2개의 스테이지를 구성할 수 있다. 이와 같은 식 (7)과 (8)에서 필요한 곱셈연산에 대하여 살펴보기로 한다. 식 (7)과 (8)에서 $W_N^{(N/4)n_1k_1}$과 $W_N^{(N/4)n_2k_2}$는 4-point FFT이므로 곱셈연산이 아니다. 또한 식 (7)의 $W_N^{(N/16)n_2k_1}$의 복소곱셈은 다음과 같이 나타낼 수 있다.

$$W_N^{\frac{N}{16}n_2k_1}=W_{16}^{n_2k_1} \tag{10}$$

첫 번째 스테이지는 총 6개의 $W_{16}^1, W_{16}^2, W_{16}^3, W_{16}^4, W_{16}^6, W_{16}^9$가 사용된다. 그러나 Twiddle Factor는 주기함수이므로 중복되는 것을 제외하고 $N/8$ 만큼의 각도만 고려하면 되므로 W_{16}^1, W_{16}^2만 남는다. 즉 W_N의 곱셈에서 사용되는 복소곱셈 종류의 수 L은 $N/8$으로 나타낼 수 있으므로 $N=16$의 식 (10)에서 필요한 복소곱셈의 종류의 수 L은 다음과 같다.

$$L=\frac{N}{8}=2 \tag{11}$$

따라서 이 스테이지에서 필요한 곱셈 종류의 수 M은 다음과 같다.

$$M=2L-1=\frac{N}{4}-1=3 \tag{12}$$

즉 Radix-4^2 알고리즘을 사용하여 구성한 2개의 스테이지 중에서 첫 번째 스테이지의 곱셈 종류의 수는 3개이다. 따라서 Radix-4^2 알고리즘을 사용하면 2개의 스테이지마다 1개의 스테이지에서 곱셈 종류의 수가 3인 스테이지가 발생한다. 이 논문에서는 이와 같은 곱셈 종류의 수가 3인 스테이지에서 곱셈연산 블록을 작게 설계하는 방법을 제안한다.

Ⅲ. 제안된 FFT 구조

1. 첫 번째 스테이지 설계

이 절에서는 Radix-4^2 알고리즘을 사용하여 매 2 스테이지마다 곱셈연산 하드웨어의 크기를 감소시킨 FFT 구조를 제안한다. 제안 구조의 유도를 위하여 256-point FFT를 사용하였으며 전체 구조의 블록도는 그림 1과 같이 4개의 스테이지로 구성된다.

그림 1에서 보듯이 각각의 스테이지는 Delay commutator(DC), 덧셈기 블록(A), 그리고 곱셈연산 블록(M)으로 구성된다. 256-point FFT는 Radix-4^2 알고리즘을 사용하면 4x4x16으로 분해되고, 남은 16-point FFT에 대하여 다시 Radix-4^2 알고리즘을 사용하여 분해한다. 따라서 그림 1에서 M1과 M3의 곱셈연산 하드웨어의 구현 면적을 줄일 수 있다. 첫 번째 스테이지의 DC1은 MDC 방식을 사용하여 설계하였다. 이제 첫 번째 스테이지의 버터플라이 블록을 설계해보기로 한다. 먼저 구현해야할 식은 다음과 같다.

$$\left(\sum_{n_1=0}^{3}x\left(\frac{N}{4}n_1+\frac{N}{16}n_2+n_3\right)W_N^{\frac{N}{4}n_1k_1}\right)W_N^{\frac{N}{16}n_2k_1} \tag{13}$$

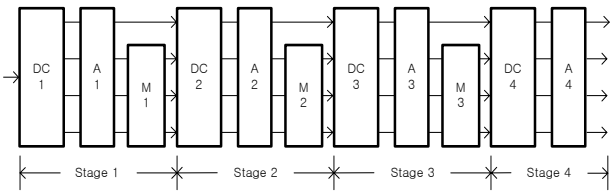

그림 1. Radix-4^2 256-point FFT 블록도
Fig. 1. Block diagram of the Radix-4^2 256-point FFT.

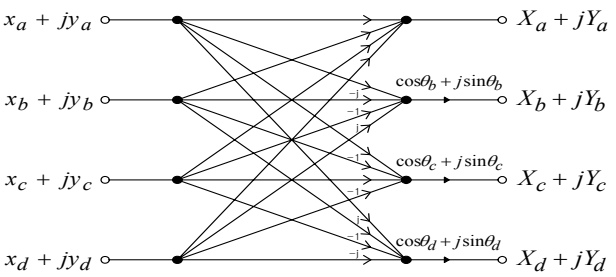

그림 2. 첫 번째 스테이지의 버터플라이 블록도
Fig. 2. Block diagram of the first stage butterfly.

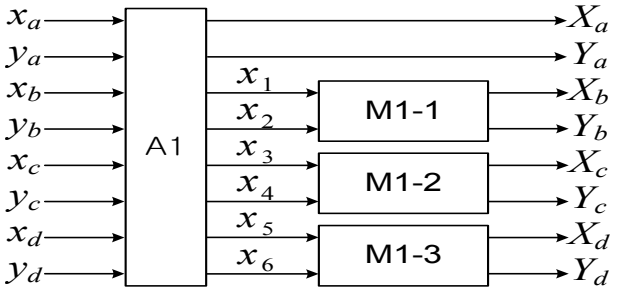

그림 3. 첫 번째 스테이지 버터플라이 회로도
Fig. 3. Block diagram of the first stage butterfly.

식 (13)을 구현하기 위한 버터플라이 구조는 그림 2와 같다.

그림 2의 출력의 실수부와 허수부를 연산하기 위한 버터플라이 블록도는 그림 3과 같다.

그림 3의 덧셈 블록 A1에서 수행해야 하는 연산은 다음과 같다.

$$X_a = x_a + x_b + x_c + x_d, \quad Y_a = y_a + y_b + y_c + y_d$$
$$x_1 = x_a + y_b - x_c - y_d, \quad x_2 = y_a - x_b - y_c + x_d$$
$$x_3 = x_a - x_b + x_c - x_d, \quad x_4 = y_a - y_b + y_c - y_d$$
$$x_5 = x_a - y_b - x_c + y_d, \quad x_6 = y_a + x_b - y_c - x_d \qquad (14)$$

그림 3의 곱셈 블록 M1-1, 2, 3에서 수행해야 하는 곱셈연산은 가각 다음과 같다.

$$X_b = x_1\cos\theta_b + x_2\sin\theta_b, \quad Y_b = x_2\cos\theta_b - x_1\sin\theta_b$$
$$X_c = x_3\cos\theta_c + x_4\sin\theta_c, \quad Y_c = x_4\cos\theta_c - x_3\sin\theta_c$$
$$X_d = x_5\cos\theta_d + x_6\sin\theta_d, \quad Y_d = x_6\cos\theta_d - x_5\sin\theta_d \qquad (15)$$

위의 곱셈 연산에서 사용되는 곱셈연산 종류의 수는 3개이며 그 값은 각각 0.7071, 0.9239, 0.3827이다. 첫 번째 스테이지에서는 그림 3의 곱셈연산 블록이 각기 다른 64번의 연산을 수행해야 하며 그림 4와 같이 16 clock마다 연산이 달라져야한다.

Clock 1부터 clock 16까지의 곱셈연산은 그림 4(a)와 같이 연산 없이 그대로 통과시킨다. 그림 4의 4가지 경우를 모두 연산하기 위해 MUX를 사용하여 그림 5와 같은 회로를 제안한다.

그림 5의 MUX는 4개의 입력을 선택하도록 설계하였으며, 16 clock 마다 입력을 위에서부터 차례로 선택하도록 설계하였다. 그림 5에서 보듯이 곱셈연산 하드웨어는 회로가 매우 간단함을 알 수 있다. 즉 사용되는 곱셈 종류가 3개이므로 레지스터나 ROM에 저장할 필요 없이 바로 하드웨어를 만들어 사용함으로써 구현 면적

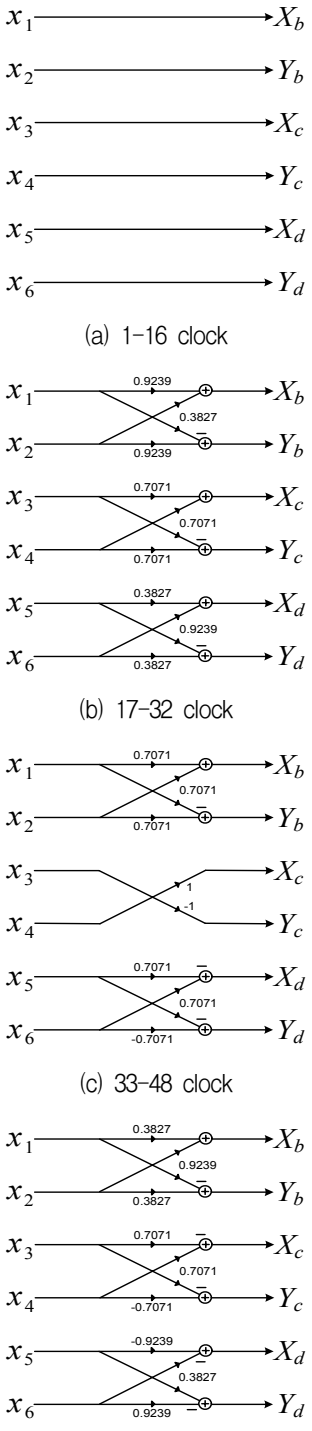

그림 4. 첫 번째 스테이지의 clock 별 곱셈연산
Fig. 4. Multiplication per each clock in the first stage butterfly.

을 줄일 수 있게 된다. 그림 5의 곱셈 연산을 덧셈기를 사용하여 구현하기 위하여 먼저 곱셈 계수들을 표 2와 같은 CSD(Canonic Signed Digit)형으로 나타낸다.

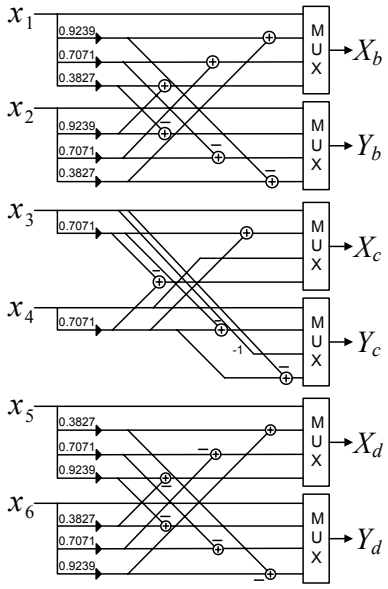

그림 5. 제안된 첫 번째 스테이지 곱셈연산 블록
Fig. 5. Proposed multiplication block of the first stage butterfly.

표 2. 첫 번째 스테이지 Twiddle Factor의 CSD형 계수
Table 2. CSD type coefficients of the first stage twiddle Factors.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0.9239	1	0	0	0	N	0	N	0	0	1	0	0	0	0	1	0
0.3827	0	1	0	N	0	0	0	1	0	0	0	0	0	N	0	0
0.7071	1	0	N	0	N	0	1	0	1	0	0	0	0	0	1	0

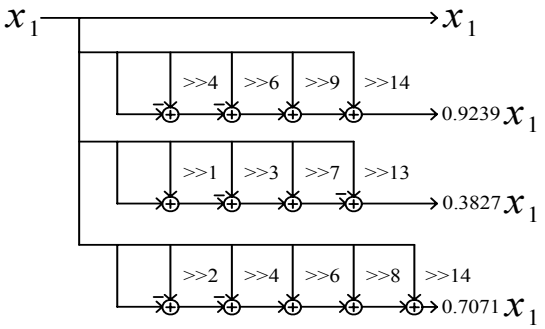

그림 6. 곱셈 연산 하드웨어 설계 회로도
Fig. 6. Circuit diagram for multiplication hardware.

표 2의 CSD 형 계수를 사용하여 그림 5의 X_b용 곱셈 연산 하드웨어를 설계하면 그림 6과 같다.

2. 두 번째 스테이지 설계

두 번째 스테이지의 DC2도 MDC 방식을 사용하여 설계하였다. 두 번째 스테이지의 버터플라이 블록에서 구현해야할 식은 다음과 같다.

$$\left(\sum_{n_2=0}^{3} P\left[\frac{N}{16}n_2+n_3,k_1\right] W_N^{\frac{N}{4}n_2k_2}\right) W_N^{n_3(k_1+4k_2)} \quad (16)$$

식 (16)에서 사용되는 곱셈연산 계수의 수는 63개이므로 곱셈연산 하드웨어는 Booth 곱셈기나 CORDIC 곱셈기를 사용하여 구현한다.

3. 세 번째 스테이지 설계

Radix-4^2 알고리즘을 사용하여 두 스테이지를 설계하였으므로 16-point FFT가 남는다. 이에 대하여 다시 Radix-4^2 알고리즘을 적용하기로 한다.

세 번째 스테이지의 버터플라이에서 계산해야 할 연산은 첫 번째 스테이지의 식 (13)과 정확히 같다. 따라서 버터플라이는 그림 5를 그대로 사용한다. 다만 MUX를 그림 5와는 다르게 제어하여야 한다. 즉 매 Clock 마다 MUX 입력을 위에서부터 선택하며 4 Clock 마다 이를 16번 반복하도록 설계하였다.

4. 네 번째 스테이지 설계

네 번째 스테이지는 마지막 스테이지이므로 곱셈 연산 블록이 없다. 따라서 DC4의 출력 값이 버터플라이로 들어와 덧셈 연산을 수행하고 최종 출력이 된다.

Ⅳ. 구 현

이 절에서는 제안된 FFT 프로세서 구조의 효율성을 살펴보기로 한다. 즉 제안 구조의 Verilog-HDL 코딩과 합성을 통하여 면적이 실제 얼마나 감소하였는지 알아보기로 한다.

1. 테스트 벡터 생성 및 Function simulation

제안 구조를 시뮬레이션하기 위해 256-point FFT를 파이프라인 Radix-4^2 MDC 방식으로 설계하였고, 첫 번째 스테이지와 세 번째 스테이지의 곱셈 블록에서 제안된 방식을 이용하였다. 설계 검증을 위한 입력 테스트 벡터는 Matalb을 통해 랜덤으로 만든 임의의 256개 샘플값을 사용하였고, 출력 테스트 벡터는 Matalb의 FFT 함수를 사용하여 추출하였다. 제안 구조의 동작 검증을 위한 Function simulation은 C 코딩을 사용하였다. 즉 C 코딩을 사용하여 제안 구조를 코딩하여 출력된 결과 값을 MatLab을 통해서 생성된 테스트 벡터와 비교하였다.

2. Verilog-HDL simulation 및 Synthesis

제안된 FFT 구조를 Verilog-HDL로 모델링하였으며, Function 시뮬레이션 검증을 위해 Modelsim을 이용하여 waveform을 확인하였다. Verilog Waveform 결과는 앞서 추출한 Function 시뮬레이션 결과와 비교하여 결과 값이 일치하는지 확인하였다.

합성은 tsmc 0.18μm CMOS 라이브러리를 이용하였으며 합성 결과의 schematic view는 그림 7과 같다.

기존 구조와의 비교를 위하여 [8]에서 제안된 구조1을 기존 구조로 선택하였다. 기존 구조는 Radix-4 알고리즘을 사용하여 256-point FFT를 설계하였으며, 첫 번째 스테이지의 곱셈 블록은 CORDIC을 사용하고, 두 번째와 세 번째 스테이지의 곱셈블록은 CSD를 사용하여 구현한 구조이다. [8]의 기존 구조는 Chartered 0.18 μm CMOS 라이브러리를 사용하여 구현된 결과를 인용하였다. 먼저 기존 구조와 제안 구조의 곱셈 블록의 면적을 비교하면 표 3과 같다.

표 4에서 보듯이 기존 구조 구조의 곱셈연산 블록의 구현 면적은 0.786mm^2이다.[8] 제안된 구조의 곱셈연산

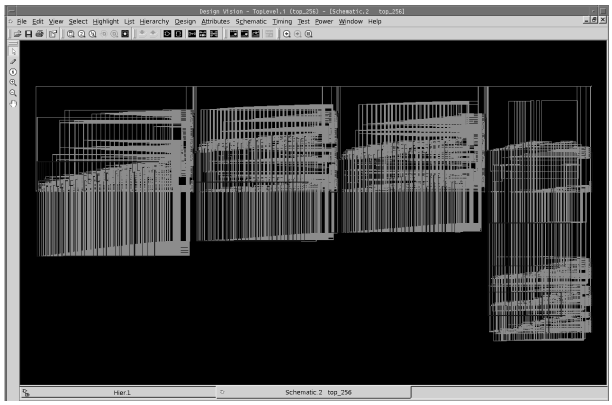

그림 7. tsmc 0.18μm 논리 합성 결과의 schematic view

Fig. 7. Schematic view of the tsmc 0.18μm Logic synthesis result.

표 3. 제안된 복소 곱셈블록의 cell area 비교(mm^2)

Table 3. Cell area comparison of the proposed multiplier block(mm^2).

	Radix-4	Radix-4^2 (proposed)
1st stage	0.556	0.067
2nd stage	0.199	0.264
3rd stage	0.031	0.067
total	0.786 (100%)	0.398 (50.6%)

표 4. 제안된 256-point FFT 구조의 면적 비교(mm^2)

Table 4. Cell area comparison for the proposed 256-point FFT structure(mm^2).

	Radix-4			Radix-4^2 (proposed)		
	DC	A	M	DC	A	M
1 stage	0.596	0.028	0.556	0.488	0.024	0.067
2 stage	0.781	0.028	0.199	0.731	0.024	0.264
3 stage	0.235	0.028	0.031	0.196	0.024	0.067
4 stage	0.064	0.028		0.062	0.024	
Total	2.574 (100%)			1.971 (76.6%)		

블록 합성 결과는 은 0.398mm^2이다. 이제 256-point FFT 구조 전체에 대한 면적을 비교해보기로 한다. 곱셈연산 블록, DC 블록, 덧셈연산 블록의 전체 256-point FFT 구조에 대한 합성한 결과는 표 4와 같다.

표 4에서 보듯이 기존 구조의 합성 결과 구현 면적은 2.574mm^2이며[8] 제안구조의 구현 면적은 1.971mm^2로 합성되었다. 제안된 구조가 기존 구조와 비교하여 면적이 23.4% 감소된 것을 확인 할 수 있다. 이 비교에서 기존 구조와 제안 구조는 0.18μm 공정을 사용하였지만 똑같은 공정을 사용한 합성 결과는 아니다. 그러나 제안 구조의 저면적 구현 효과는 입증할 수 있었다.

V. 결 론

이 논문에서는 OFDM용 MODEM SoC의 FFT 블록에 대한 저면적 구현 방법을 제안하였다. Radix-4^2 알고리즘을 사용하면 매 2 스테이지마다 곱셈 종류의 수가 3개인 스테이지가 나타나는 것을 이용하여 저면적 구조를 유도하였다. 즉 곱셈 연산 종류의 수가 3인 곱셈 연산 블록을 CSD와 CSS 방식을 사용하여 저면적으로 구현할 수 있음을 보였다. 제안된 256-point FFT 블록에 대하여 Verilog-HDL 코딩과 tsmc 0.18μm 공정의 합성을 통하여 cell area를 계산한 결과 23.4%의 면적 감소 효과를 얻을 수 있었다.

참 고 문 헌

[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.

[2] M. Bekooij, J. Huisken, and K. Nowak,

"Numerical accuracy of Fast Fourier Transforms with CORDIC arithmetic", Journal of VLSI Signal Processing 25, pp. 187-193, 2000.
[3] S. M Kim, J. G. Chung, and K. K. Parhi, "Low error Fixed-width CSD Multiplier with Efficient Sign Extension", IEEE Trans. Circuits and Systems-II, vol. 50, No. 12, Dec. 2003.
[4] R. I. Hartley, "Sub-expression sharing in filters using canonic signed digit multipliers", IEEE Trans. Circuits and Systems-II: Analog and Digital Signal Processing, vol. 43, No.10, pp. 677-688, Oct. 1996.
[5] Y. Jang, and S. Yang, "Low-power CSD linear phase FIR filter structure using vertical common sub-expression" IEE Electronics Letters, vol. 38, No. 15, pp. 777-779, Jul. 2002.
[6] J. Lee and H. Lee, "High-Speed 2-Parallel Radix-2^4 FFT/IFFT Processor for MB-OFDM UWB Systems", IEICE Trans. on Fundamentals of Electronics, communications, and Computer Science, vol. E91-A, No. 4, pp. 1206-1211, April, 2008.
[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.
[8] 최동규, 장영범, "Common sub-expression sharing과 CORDIC을 이용한 OFDM 시스템의 저면적 파이프라인 FFT 구조", 전자공학회논문지, 제46권, SP편 제 4호, pp. 157-164, 2009.
[9] 장영범, 이동훈, "Common sub-expression sharing을 사용한 저면적 FFT 프로세서 구조", 한국산학기술학회논문지, 제12권, 제4호, pp. 1867-1875, 2011.

저 자 소 개

장 영 범(정회원)
1981년 연세대학교 전기공학과 공학사 졸업.
1990년 Polytechnic University, EE 공학석사 졸업.
1994년 Polytechnic University, EE 공학박사 졸업.
1983년~1999년 삼성전자 System LSI 사업부 수석연구원.
2000년~2002년 이화여자대학교 정보통신학과 연구교수.
2002년~현재 상명대학교 정보통신공학과 교수.
<주관심분야 : 통신신호처리, 비디오신호처리, SoC 설계>

김 한 진(학생회원)
2002년~2008년 2월 상명대학교 정보통신공학과 공학사 졸업.
2009년 9월~2011년 8월 상명대학교 대학원 컴퓨터정보통신공학과 공학석사 졸업.
<주관심분야 : 통신신호처리, SoC 설계>