OFDM 을 위한 64 점 R2² SDF 파이프라인 FFT 프로세서 설계

이상한, 이태욱, 이종화, 조상복
울산대학교 전기전자정보시스템 공학부

Design of 64-point R2²SDF pipeline FFT processor in OFDM

Sanghan Lee, Taewook Lee, Jonghwa Lee, Sangbok Cho
School of Electrical Engineering

Abstract

A 64-point R2² SDF pipeline FFT processor using a new efficient computation sharing multiplier was designed. Computation sharing multiplication specifically targets computation re-use in multiplication of coefficient vector by scalar and is effectively used in DSP(Digital Signal Processing). To reduce the number of multipliers in FFT, we used the proposed computation sharing multiplier. The 64-point pipeline FFT processor was implemented by VHDL and synthesized using Max+PLUSII of Altera. The simulation result shows that the proposed computation sharing multiplier can be reduced to about 17.8% logic cells compared with a conventional multiplier. This processor can operate at 33MHz and calculate a 64-point pipeline FFT in 1.94 μs.

I. 서론

최근 고품질의 이동 멀티미디어 서비스를 위한 통신 시스템에서 고속으로 데이터를 송신하고 수신하기 위해 OFDM(Orthogonal Frequency Division Multiplexing) 변조방식이 많이 사용되고 있다. OFDM 변복조는 다수의 부반송파를 사용하기 때문에 부반송파의 수가 많아지면 부반송파간의 직교성을 유지하기 위한 어려움이 발생하여 실제 시스템에 구현하기 어려워진다. 이러한 문제점을 DFT(Discrete Fourier Transform)에 의해 구현할 수 있으며 DFT 의 많은 연산량을 줄이기 위해 FFT 알고리즘을 이용하여 구현한다[1].

본 논문에서는 파이프라인 FFT 구조 중 가장 많이 사용되는 Radix-2² SDF(Single Delay Feedback) 파이프라인 FFT 구조 사용하였다[2]. 또한 전체 FFT 면적의 약 50%를 차지하는 복소 승산기의 면적을 줄이기 위해 기존의 승산기 대신 내적(inner product)연산에 많이 사용되는 연산 공유 승산기를 변형하여 사용하였다.

논문의 구성은 다음과 같다. II 장에서는 파이프라인 FFT 구조 중 가장 많이 사용되는 Radix-2² 알고리즘에 대해 언급하고 III 장에서는 새로운 연산 공유 승산 알고리즘에 살펴본다. IV 장에서는 수정된 복소 승산기를 적용한 64 점 파이프라인 FFT 프로세서 설계 과정에 대해 소개하고 VHDL 을 통한 시뮬레이션을 수행한다. 마지막으로 VI 장에서 결론을 맺는다.

II. Radix-2² 알고리즘

Radix-2² 알고리즘은 복소 승산기의 수를 줄이기 위해 Radix-4 알고리즘의 계산적 요구량을 반영하고 전체 구조의 제어를 용이하기 위해 Radix-2 알고리즘의 구조를 반영한다[2].

N 점 DFT 는 수식 (1)과 같이 정의된다.

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

Radix-2² 알고리즘에 대한 수식을 추출하기 위하여, 3 차원 선형 인덱스 맵(3-dimensional linear index map)을 적용하면

$$n=<\frac{N}{2}n_1+\frac{N}{4}n_2+n_3>_N \tag{2}$$
$$k=<k_1+2k_2+4k_3>_N$$

공통 인자 알고리즘(Common factor algorithm)은 아래의 형태를 가진다.

$$X(k_1+2k_2+4k_3)$$
$$\sum_{n_3=0}^{\frac{N}{4}-1}\sum_{n_2=0}^{1}\sum_{n_1=0}^{1}x(\frac{N}{2}n_1+\frac{N}{4}n_2+n_3)W_N^{(\frac{N}{2}n_1+\frac{N}{4}n_2+n_3)(k_1+2k_2+4k_3)}$$
$$=\sum_{n_3=0}^{\frac{N}{4}-1}\sum_{n_2=0}^{1}\left\{B_{\frac{N}{2}}^{k_1}(\frac{N}{4}n_2+n_3)W_N^{(\frac{N}{4}n_2+n_3)k_1}\right\}W_N^{(\frac{N}{4}n_2+n_3)(2k_2+n_3)(2k_2+4k_3)} \quad (3)$$

회전인자(twiddle factor) $W_N^{(\frac{N}{4}n_2+n_3)k_1}$ 를 포함한 부분은

$$W_N^{(\frac{N}{4}n_2+n_3)(k_1+2k_2+4k_3)}$$
$$=W_N^{Nn_2k_3}W_N^{\frac{N}{4}n_2(k_1+2k_2)}W_N^{n_3(k_1+2k_2)}W_N^{4n_3k_3} \quad (4)$$
$$=(-j)^{n_2(k_1+2k_2)}W_N^{n_3(k_1+2k_2)}W_N^{4n_3k_3}$$

로 정리될 수 있다.

식(4)를 식(3)에 대입하여 간단화 하면 $N/4$ 길이의 4 DFTs 를 가진다.

$$X(k_1+2k_2+4k_3)=\sum_{n_3=0}^{\frac{N}{4}-1}\left[H(k_1,k_2,n_3)W_N^{n_3(k_1+2k_2)}\right]W_{\frac{N}{4}}^{n_3k_3} \quad (5)$$

여기서 $H(k_1,k_2,k_3)$는 식(6)에 표현 되었다.

$$H(k_1,k_2,k_3)=\overbrace{\left[x(n_3)+(-1)^{k_1}x(n_3+\frac{N}{2})\right]}^{\text{BF I}}+(-j)^{(k_1+2k_2)}\overbrace{\left[x(n_3+\frac{N}{4})+(-1)^{k_1}x(n_3+\frac{3}{4}N)\right]}^{\text{BF I}} \quad (6)$$
$$\text{BF II}$$

식(6)은 그림 1 의 BF I 과 BF II 두개의 버터플라이와 같이 두 단계에서 간단한 승산만이 이루어짐을 보여준다. 이 두 단계 이후에 식(5)에서 회전인자 $W^{n_3(k_1+2k_2)}$ 를 계산하기 위해 승산기가 필요 하게 된다. 그림 1 에서처럼 Radix-2^2 알고리즘은 Radix-4 알고리즘과 같은 승산 구조를 가지지만 여전히 Radix-2 의 버터플라이 구조를 가진다.

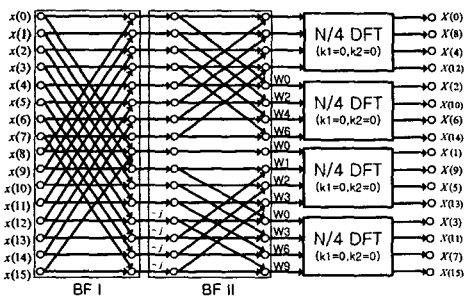

그림 1. N=16 에 대한 Radix-2^2 DIF FFT 흐름도

III. 새로운 연산 공유 승산 알고리즘

연산 공유 승산 알고리즘은 영상처리 및 DSP 에서 많이 사용되는 내적 연산을 좀 더 효율적으로 하드웨어로 구현하기 위해 제안된 알고리즘이다. 연산 공유 승산 알고리즘은 내적 연산을 벡터 스케일링 오퍼레이션으로 변환하여 상수 벡터 $c=[c_0,c_1,...,c_{M-1}]$ 와 스케일러(x) 와의 승산으로 바꾸어 계산한다. 이때 상수 벡터는 4 비트씩 나누어 계산되는데 4 비트의 상수와 스케일러의 곱을 공유하여 사용함으로써 하드웨어의 크기를 줄이는 데 목적을 두고 있다[3][4].

3.1 기존의 연산 공유 승산기

기존의 연산 공유 승산기는 전처리기, 선택기 그리고 가산기의 3 부분으로 구성된다. 그림 2 는 기존의 연산 공유 승산기의 구조를 나타낸 것으로 각 부분의 동작은 다음과 같다.

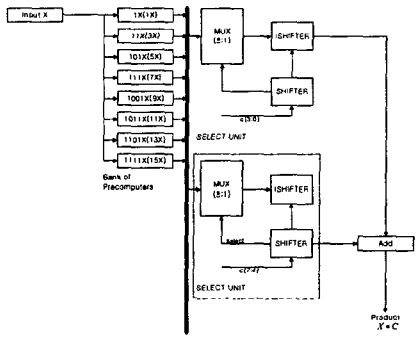

그림 2. 기존의 연산 공유 승산기 구조

1. 전처리기 : 스케일러 x 와 알파벳간의 승산을 수행하여 벡터 스케일링 오퍼레이션의 결과 값을 미리 계산 하는 부분이다.
2. 선택기 : 전처리기에서 나온 승산 결과를 상수의 상위 4 비트와 하위 4 비트에 따라 선택하고 쉬프트 시키는 부분으로 멀티플렉서, Shifter, Ishifter 로 구성된다. 멀티플렉서는 표 1 의 정의에 따라 Shifter 에서 나오는 select 신호에 따라 전처리기의 값을 선택한다. 그리고 Shifter 는 멀티플렉서에 사용될 select 신호와 Ishifter 에 사용될 shift 신호를 생성한다. 마지막으로 Ishifter 는 Shifter 에서 나오는 shift 신호에 따라 멀티플렉서의 값을 이동시킨다.
3. 가산기 : 선택기의 출력 값은 상수의 상위 4 비트와 하위 4 비트에 따른 승산결과 이므로 이

표 1. 상수 값에 따른 select 와 shift 신호

Select input	Mux output
000	$1x$
001	$3x$
010	$4x$
011	$7x$
100	$9x$
101	$11x$
110	$13x$
111	$15x$

값을 입력 받아 최종 승산 결과를 구해야 한다. 가산기는 선택기의 상위 4 비트의 값을 좌측으로 4 비트 이동시켜 하위 4 비트의 값과 가산하여 승산 결과를 출력한다.

3.2 제안된 연산 공유 승산기

기존의 연산 공유 승산기는 다음의 3 가지 단점이 있다.

첫째, 전처리기의 출력을 위해 5 개의 가산기와 2 개의 감산기가 필요하므로 전처리기 자체의 하드웨어가 복잡하다. 둘째, 전처리기의 출력 $11x$ 와 $13x$ 의 $3x$ 와 $5x$ 의 출력을 입력 받아 계산하므로 프로세서 전체의 지연시간이 커질 수 있다. 셋째, 선택기의 Shifter 와 Ishifter 구현을 위한 하드웨어 용량이 크다.

본 장에서는 위의 3 가지 단점을 보완하고 FFT 프로세서에 사용되는 복소 승산기에 적용하기 위해 음수의 상수 값도 계산 가능 하도록 하여 새로운 연산 공유 승산기를 제안한다. 제안된 연산 공유 승산기는 그림 4 와 같이 3 개의 블록으로 구성되며 각 블록에 대한 설명은 다음과 같다.

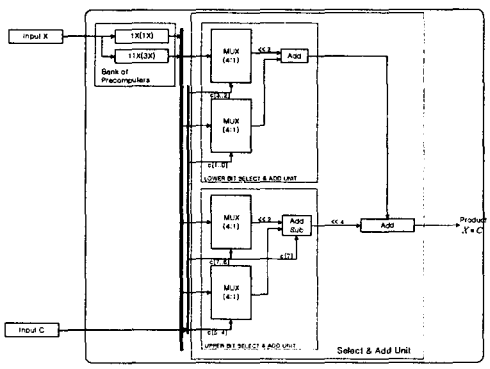

그림 3. 제안된 연산 공유 승산기

1. **전처리기** : 전처리기의 알파벳은 가산기로 구성되므로 알파벳의 개수를 줄이는 것이 전체 프로세서의 하드웨어 복잡도를 줄일 수 있고, 또한 $11x$ 와 $13x$ 의 경우와 같이 전처리기의 출력을 다시 입력으로 사용하는 경우의 지연 시간을 줄일 수 있다.
2. **선택/가감산기** : 제안된 선택/가감산기는 기존의 선택기에 사용된 Shifter 와 Ishifter 를 없앤 대신 2 개의 4× 1 멀티플렉서는(표 2) select 신호가 된다.

표 2. Select 신호에 따른 멀티플렉서의 출력

Select input	Mux I output	Mux II output
00	0	0
01	x	x
10	$2x\,(x<<1)$	$2x\,(x<<1)$
11	$3x$	x

3. **가산기** : 기존의 가산기 블록과 같이 8 비트 상수 값에 대한 승산 값을 구한다.

그림 4 는 부스 승산기, 기존의 연산 공유 승산기 그리고 제안된 연산 공유 승산기의 로직 셀 사용도를 보여 주고있다. 제안된 연산 공유 승산기는 기존의 연산 공유 승산기와 비교(8× 8 승산의 경우)하여 볼 때 약 40%의 로직 셀을 줄일 수 있었다.

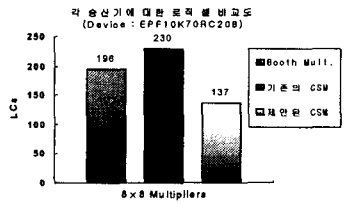

그림 4. 승산기들의 로직 셀 사용도 비교

IV. 64 점 파이프라인 FFT 프로세서 설계

Radix-2^2 SDF 방식의 64 점 파이프라인 FFT 의 전체 구조는 그림 5 와 같다. 전체 3 개의 PE(Processing Element)블록과 2 개의 복소 승산기 그리고 제어블록으로 구성이 되어 지는데 각 PE 블록은 버터플라이 연산을 수행하는 BF I 과 BF II, 입력 받은 데이터를 저장하는 FIFO(First In Fist Out)으로 구성된다.

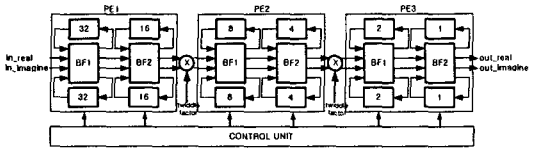

그림 5. Radix-22 SDF 64 점 파이프라인 FFT 구조

4.1 복소 승산기 설계

기존의 복소 승산기는 식(7)처럼 4 개의 승산기와 1 개의 가산기, 그리고 1 개의 감산기로 구성되어진다.

$$(a_r + ja_i)(b_r + jb_i) = (a_r b_r - a_i b_i) + j(a_r b_i + a_i b_r) \quad (7)$$

제안된 연산 공유 승산기를 적용하기 위해 스케일러 입력으로 a_r 과 a_i 을 사용하고 상수 벡터의 입력으로 b_r 과 b_i 를 사용하였다. 연산공유 승산기가 적용된 구조는 그림 6 과 같다.

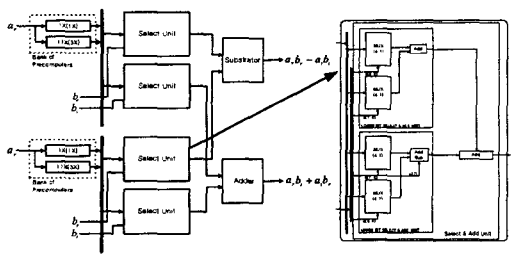

그림 6. 연산공유 승산기를 적용한 복소 승산기 구조

V. 시뮬레이션

회로의 설계는 VHDL 을 사용하였으며 하드웨어 구현 및 시뮬레이션은 Altera 의 Max+PLUSII 를 사용하였다. 그림 7 은 Altera 툴을 이용한 시뮬레이션 파형을 보여주고 있다.

입력 클럭 61 번째부터 연산 결과 값이 출력되게 되고 연산에만 소요되는 클럭은 총 64 개의 클럭이 소모되게 된다. 따라서 연산 클럭(Computational Clock)을 33MHz 이상으로 할 경우, 연산 수행 시간은 1.94 μs 이내가 된다.

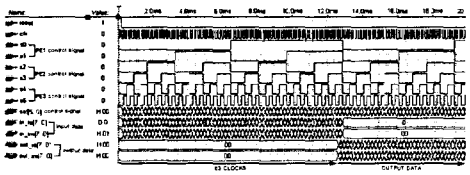

그림 7. 64 점 FFT 프로세서 시뮬레이션 파형

표 3 은 기존의 복소 승산기와 수정된 복소 승산기를 이용한 64 점 파이프라인 FFT 프로세서를 합성하여 비교한 것이다. 로직 셀 사용도를 보여 주고 있고 제안된 연산 공유 승산기를 적용한 복소 승산기를 이용한 64 점 파이프라인 FFT 는 전체 로직 셀의 약 17.8%를 절약할 수 있게 된다.

표 3. 2 가지 64 점 FFT 파이프라인 프로세서 합성 결과

	기존의 복소 승산기 적용	수정된 복소 승산기 적용
Device	EPF10K70RC208	EPF10K70RC208
Input bit	8	8
Output bit	8	8
Logic cell used	3253	2675

VI. 결론

본 논문에서는 기존의 복소 승산기에 새로운 연산 공유 승산기를 적용하여 64 점 R2^{2}SDF 파이프라인 FFT 를 설계하였다. 기존 공유 승산기는 전처리기의 크기가 크고 선택기에 Shifter 와 Ishifter 를 사용함으로서 승산기 전체의 성능이 저하되는 단점을 가지고 있다. 이러한 단점을 없애기 위해 전처리의 알파벳 수를 줄이고 선택기 대신 Shifter 와 Ishifter 를 없앤 선택/가산기를 사용하여 새로운 연산 공유 승산기를 제안하였다. Altera 사의 Max+PLUSII 를 이용하여 64 점 R2^{2}SDF 파이프라인 FFT 프로세서를 구현 하여 합성한 결과 전체 면적의 약 17.8%의 로직 셀을 절약할 수 있었다.

※감사의 글 : 감사의 글 : 본 논문은 한국과학재단 재정 네트워크기반 자동화 연구센터(NARC)와 반도체 설계 교육 센터(IDEC)의 지원에 의해 이루어 졌습니다.

참고문헌

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

[2] S. He and M. Torkelson, "Design and implementation of a 1024-point pipeline FFT processor," in Proc. IEEE Custom Integrated Circuits Conf., pp.131-135, 1998.

[3] K. Muhammad and K. Roy, "Minimally redundant parallel implementation of digital filters and vector scaling", in Proc. IEEE Int. Conf., Vol 6, pp. 3295-3298, 2000.

[4] J. S. Park, S. K. Kwon and K.Roy, "Low power reconfigurable DCT design based on sharing multiplication", in Proc. IEEE Int. Conf., Vol 3, pp.3116-3119, 2002.