고속 및 면적 효율적인 FFT 알고리즘 개발 및 하드웨어 구현

탁 연 지, 정 윤 호, 김 재 석, *박 현 철, *김 동 규, *박 준 현, *유 봉 위
연세대학교 전기 전자 공학과, *(주) 삼성 전자
전화 : 02-2123-7670 / 핸드폰 : 016-273-7670

A High Speed and Area Efficient FFT Algorithm and Its Hardware Implementation

YonJi Tak , Yun-Ho Jung, JaeSeok Kim,
*HyunCheol Park, *Dong Gyu Kim, *JunHyun Park, *Bong-Wee Kim
E-mail : plusj@asic.yonsei.ac.kr
Dept. of Electrical & Electronic Engineering, Yonsei Univ.
*Samsung Electronics Co., Ltd

Abstract

This paper proposes a high-speed and area-efficient FFT algorithm and performs a hardware implementation. This algorithm, named by "Radix-4/2", uses the feature of existing radix-2^3 algorithm, It reduces the number of non-trivial multipliers in SFG to the ratio of 3 to 2 campared with radix-2 or radix-4 algorithm and radix-4/2 has also twice throughput as radix-2^3 algorithm's.

It is proved that FFT processor using the proposed algorithm and 64-point MDC pipeline architecture has twice throughput as radix-2^3 algorithm's, and reduces areas by 25 percentages in contrast to radix-4 algorithm's.

I. 서론

무선 데이터 통신 시스템은 다양한 품질의 서비스를 제공받고자 하는 사용자의 요구가 증가함에 따라 점차 고속화되고 있는 추세이다. 고속의 무선 데이터 통신을 위해서는 단일 반송파 통신 방법보다는 다중 반송파를 이용한 통신 방법이 고려되고 있으며, 이 중 OFDM (Orthogonal Frequency Division Multiplexing) 방식은 높은 대역 효율성, 다중 경로 페이딩에 대한 저항성과 같은 이점으로 고속 무선 데이터 통신 시스템의 변조기법으로 널리 사용되고 있다. OFDM방식에서는 직렬로 입력되는 데이터 열을 N개의 병렬 데이터 열로 변환하여 각각을 부반송파에 실어 전송함으로써, 데이터 율을 높이는 원리이다. 이 때, 직교 반송파를 생성하기 위한 고속의 FFT 프로세서를 구현하는 것이 필수적이다.

본 논문은 기존의 여러 FFT 알고리즘에 대한 분석을 기반으로 새로운 방식의 고속 및 면적 효율적인 Radix-4/2 알고리즘을 제안한다. 또한, 이 알고리즘을 적용하여 무선 LAN 시스템에 적합한 FFT 프로세서를 MDC 파이프라인 구조를 이용하여 하드웨어로 구현하며, 논리 합성 결과를 통해 기존 알고리즘과 비교하여 성능을 분석하고 검증 결과를 제시한다.

II. 기존의 FFT 알고리즘

2.1 Radix-2 와 Radix-4 FFT 알고리즘

N-point DFT(Discrete Fourier Transform)는 식 (1)과 같이 정의된다.

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

여기서 $W_N^i = e^{-j\frac{2\pi}{N}i}$이며, 트위들 팩터(Twiddle Factor)라고 불린다. 트위들 팩터는 인덱스 i 에 따라 단순 승산(Trivial Multiplication)과 비단순 승산(Non-trivial Multiplication)으로 나뉘어질 수 있다. $i=\frac{N}{2}$ (W_2)와 $i=\frac{N}{4}$ (W_4)인 경우에는 입력 데이터의 부호를 바꿔주거나, 실수항과 허수항의 값을 바꿔줌으로서 간단하게 구현할 수 있고, $i=\frac{N}{8}$ (W_8)인 경우에는 $\frac{1}{\sqrt{2}}$에 대한 스케일링을 쉬프터와 가산기를 사용하여 구현함으로서 복소 승산기를 사용하지 않는 단순 승산으로 구현이 가능하다. 일반적으로 FFT 알고리즘은 버터플라이 연산자를 이용하여 DFT의 연산량을 줄인다. Radix-2 FFT 알고리즘의 경우에는 N개의 입력 시퀀스를 2개의 N/2개의 시퀀스로 정렬한 다음 2-입력 버터플라이 연산자를 통과시키고, W_2의 단순 승산을 수행한다. Radix-4 FFT 알고리즘의 경우에는 N개의 입력 시퀀스를 4개의 N/4개의 시퀀스로 정렬한 다음 4-입력 버터플라이 연산자를 통과시키고, W_4의 단순 승산을 수행한다. Radix를 4로 할 경우에는 버터플라이 연산자가 복잡해지는 단점이 있지만, 4개의 데이터를 동시에 처리하기 때문에, Radix를 2로 하는 것보다 2배 큰 수율을 갖는다.

2.2 Radix-2^2 FFT 알고리즘

FFT 알고리즘은 인덱스 분해법을 사용하면 보다 일반적인 형태로 전개될 수 있다[1]. 인덱스 분해법을 적용하면 Radix-2 FFT 알고리즘은 일차원인 N개의 입력 시퀀스를 2차원인 $2\times\frac{N}{2}$ 배열로 표현된다. 인덱스 분해과정을 한 단계 더 수행하여, 즉 n과 k를 $2\times2\times\frac{N}{4}$의 3차원 배열로 나타내면 식(2)와 같다.

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

DFT의 정의식인 식(1)에 식(2)를 대입하면 다음과 같다.

$$\begin{aligned} X(k) &= X(k_1+2k_2+4k_3) \\ &= \sum_{n_3=0}^{\frac{N}{4}-1}\sum_{n_2=0}^{1}\sum_{n_1=0}^{1}\left(x(\frac{N}{2}n_1+\frac{N}{4}n_2+n_3)\times W_N^{(\frac{N}{2}n_1+\frac{N}{4}n_2+n_3)(k_1+2k_2+4k_3)}\right) \\ &= \sum_{n_3=0}^{\frac{N}{4}-1}\sum_{n_2=0}^{1}\left(\left[BF_2(\frac{N}{4}n_2+n_3,k_1)\right]\times W_N^{(\frac{N}{4}n_2+n_3)k_1}\ W_N^{(\frac{N}{4}n_2+n_3)(2k_2+4k_3)}\right) \end{aligned} \tag{3}$$

여기서 $BF_2(\frac{N}{4}n_2+n_3,k_1)$는 Radix-2 DIF 버터플라이 연산자에 해당한다. 트위들 팩터에 해당하는 부분을 정리하면,

$$\begin{aligned} & W_N^{(\frac{N}{4}n_2+n_3)(k_1+2k_2+4k_3)} \\ &= W_N^{Nn_2k_3}\ W_N^{\frac{N}{4}n_2(k_1+2k_2)}\ W_N^{n_3(k_1+2k_2)}\ W_N^{4n_3k_3} \\ &= (-j)^{n_2(k_1+2k_2)}\ W_N^{n_3(k_1+2k_2)}\ W_{\frac{N}{4}}^{n_3k_3} \end{aligned} \tag{4}$$

이고, 여기서 $W_N^{\frac{N}{4}n_2(k_1+2k_2)}$는 단순 승산이므로, $X(k)$는 식(5)로 정리된다.

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

이다. 여기서, $H(k_1,k_2,n_3)$는

$$\begin{aligned} & H(k_1,k_2,n_3) \\ &= \sum_{n_2=0}^{1}\left[BF_2(\frac{N}{4}n_2+n_3,k_1)\right](-j)^{n_2(k_1+2k_2)} \\ &= BF_2(n_3,k_1) + (-j)^{(k_1+2k_2)}BF_2(n_3+\frac{N}{4},k_1) \end{aligned} \tag{6}$$

이고, 식(5)와 식(6)에서 Radix-2^2 FFT알고리즘은 두 단계의 Radix-2 버터플라이 연산과 N/4 point DFT로 구성됨을 확인할 수 있다. Radix-2^2 FFT 알고리즘은 Radix-2 버터플라이 연산자를 유지하면서 복소 승산 회수를 Radix-4와 같게 줄이는 장점이 있다[2].

2.3 Radix-2^3 FFT 알고리즘

Radix-2^3 FFT 알고리즘은 Radix-2^2 FFT 알고리즘에서 한 단계의 분해과정을 더 진행함으로써 구현된다. 즉, Radix-2^3 FFT 알고리즘은 4차원 인덱스 맵에 의해 분해하고, 분해된 n과 k는 식(7)과 같다.

$$n = \frac{N}{2}n_1 + \frac{N}{4}n_2 + \frac{N}{8}n_3 + n_4, \quad 0 \le n_1, n_2, n_3 \le 1,\ 0 \le n_3 \le \frac{N}{8} - 1$$
$$k = k_1 + 2k_2 + 4k_3 + 8k_4, \quad 0 \le k_1, k_2, k_3 \le 1,\ 0 \le k_3 \le \frac{N}{8} - 1 \tag{7}$$

식(7)을 DFT 정의식에 대입하면 식(8)과 같다.

$$\begin{aligned} X(k) &= X(k_1+2k_2+4k_3+8k_4) \\ &= \sum_{n_3=0}^{\frac{N}{8}-1} \sum_{n_2=0}^{1} \left(\left[BF_2\left(\frac{N}{4}n_2 + \frac{N}{8}n_3 + n_4, k_1\right)\right] W_4^{n_2(k_1+2k_2)} \times W_8^{n_3(k_1+2k_2+4k_3)} W_N^{n_4(k_1+2k_2+4k_3)} W_{\frac{N}{8}}^{n_4k_4} \right) \\ &= \sum_{n_4=0}^{\frac{N}{8}-1} \sum_{n_3=0}^{1} \left(\left[H\left(k_1, k_2, \frac{N}{8}n_3 + n_4\right) W_8^{n_3(k_1+2k_2+4k_3)} \right] \times W_N^{n_4(k_1+2k_2+4k_3)} W_{\frac{N}{8}}^{n_4k_4} \right) \\ &= \sum_{n_4=0}^{\frac{N}{8}-1} \left[T(k_1, k_2, k_3, n_4)\, W_N^{n_4(k_1+2k_2+4k_3)} \right] W_{\frac{N}{8}}^{n_4k_4} \end{aligned} \tag{8}$$

여기서, $T(k_1, k_2, k_3, n_4)$는

$$T(k_1, k_2, k_3, n_4) = H(k_1, k_2, n_4) + H\left(k_1, k_2, n_4 + \frac{N}{8}\right) W_8^{(k_1+2k_2+4k_3)} \tag{9}$$

이고, 결과적으로 3개의 Radix-2 버터플라이 연산자로 구현할 수 있고, 3단계의 분해과정을 진행하는 동안 별도의 복소 승산기가 요구되지 않는다[3].

III. 제안된 Radix-4/2 알고리즘

3.1 Radix-4^2 FFT 알고리즘

식(2)와 같은 방법으로 분해과정을 진행하되 Radix를 4로 선택한다. 3차원 인덱스 맵에 의하여 인덱스 n과 k를 분해하면 다음과 같다.

$$n = \frac{N}{4}n_1 + \frac{N}{16}n_2 + n_3, \quad 0 \le n_1 \le 3,\ 0 \le n_2 \le 1\ \ 0 \le n_3 \le \frac{N}{16} - 1$$
$$k = k_1 + 4k_2 + 16k_3, \quad 0 \le k_1 \le 3,\ 0 \le k_2 \le 1\ \ 0 \le k_3 \le \frac{N}{16} - 1 \tag{10}$$

식(9)를 식(1)에 대입하면,

$$\begin{aligned} X(k) &= X(k_1+4k_2+16k_3) \\ &= \sum_{n_3=0}^{\frac{N}{16}-1} \sum_{n_2=0}^{3} \left[BF_4\left(\frac{N}{16}n_2 + n_3, k_1\right)\right] \times W_N^{\left(\frac{N}{16}n_2+n_3\right)(k_1+4k_2+16k_3)} \end{aligned} \tag{11}$$

이고, 트위들 팩터를 정리하면, W_{16}이라는 트위들 팩터가 발생하고 이것은 비단순 복소 승산을 포함하기 때문에 Radix-4에 비해 복소 승산기 면에서 이득을 얻지 못함을 알 수 있다.

3.2 제안된 Radix-4/2 FFT 알고리즘

Radix-4를 이용하여 인덱스 분해과정을 진행했을 경우에는 승산기 면에서 이득을 얻을 수 없음을 확인하였다. 따라서, 분해과정에서 16/N Factor가 발생하지 않도록 Radix를 4, 2의 순으로 선택하여 3차원 인덱스 분해과정을 수행하는 Radix-4/2 알고리즘을 제안한다.

$$n = \frac{N}{4}n_1 + \frac{N}{8}n_2 + n_3, \quad 0 \le n_1 \le 3,\ \ 0 \le n_2 \le 1\ \ 0 \le n_3 \le \frac{N}{8} - 1$$
$$k = k_1 + 4k_2 + 8k_3, \quad 0 \le k_1 \le 3,\ \ 0 \le k_2 \le 1\ \ 0 \le k_3 \le \frac{N}{8} - 1 \tag{12}$$

분해된 인덱스를 DFT의 정의식에 대입하면, $X(k)$는 식 (13)과 같이 전개된다.

$$\begin{aligned} X(k_1+4k_2+8k_3) &= \sum_{n_3=0}^{\frac{N}{8}-1} \sum_{n_2=0}^{1} \sum_{n_1=0}^{3} \left(x\left(\frac{N}{4}n_1 + \frac{N}{8}n_2 + n_3\right) \times W_N^{\left(\frac{N}{4}n_1 + \frac{N}{8}n_2 + n_3\right)(k_1+4k_2+8k_3)} \right) \\ &= \sum_{n_3=0}^{\frac{N}{8}-1} \sum_{n_2=0}^{1} \left[BF_4\left(\frac{N}{8}n_2 + n_3, k_1\right)\right] W_N^{\left(\frac{N}{8}n_2 + n_3\right)(k_1+4k_2+8k_3)} \end{aligned} \tag{13}$$

트위들 팩터를 정리하면,

$$\begin{aligned} W_N^{\left(\frac{N}{8}n_2+n_3\right)(k_1+4k_2+8k_3)} &= W_N^{Nn_2k_3}\, W_N^{\frac{N}{8}n_2(k_1+4k_2)}\, W_N^{n_3(k_1+4k_2)}\, W_N^{8n_3k_3} \\ &= W_8^{n_2(k_1+4k_2)}\, W_N^{n_3(k_1+4k_2)}\, W_{\frac{N}{8}}^{n_3k_3} \end{aligned} \tag{14}$$

이므로,

$$\begin{aligned} X(k) &= X(k_1+4k_2+8k_3) \\ &= \sum_{n_3=0}^{\frac{N}{8}-1} \left(\left\{ \sum_{n_2=0}^{1} \left[BF_4\left(\frac{N}{8}n_2 + n_3, k_1\right)\right] W_8^{n_2(k_1+4k_2)} \right\} \times W_N^{n_3(k_1+4k_2)} W_{\frac{N}{8}}^{n_3k_3} \right) \\ &= \sum_{n_3=0}^{\frac{N}{8}-1} \left[H(k_1, k_2, n_3)\, W_N^{n_3(k_1+4k_2)} \right] W_{\frac{N}{8}}^{n_3k_3} \end{aligned} \tag{15}$$

여기서, $H(k_1, k_2, k_3, n_4)$

$$H(k_1, k_2, n_3)$$

$$= \sum_{n_2=0}^{1}\left[BF_4\left(\frac{N}{8}n_2+n_3, k_1\right)\right] W_8^{n_2(k_1+4k_2)} \quad (16)$$

$$= BF_4(n_3, k_1) + BF_4\left(n_3+\frac{N}{8}, k_1\right) W_8^{(k_1+4k_2)}$$

으로 하나의 Radix-2 버터플라이 연산자와 하나의 Radix-4 버터플라이 연산자로 구현되고, Radix-4^2와는 달리 W_8을 포함하고 있기 때문에 2단계의 분해과정을 진행하는 동안 별도의 비단순 복소 승산기를 요구하지 않는다. 또한 Radix-4를 기반으로 연산하기 때문에 Radix-2^3 FFT 알고리즘에 비해 2배 빠른 수율을 얻을 수 있다.

Ⅳ. 64-point Radix-4/2 FFT 프로세서의 설계 및 검증

고속 FFT 알고리즘의 구현 방법으로는 파이프라인 방식이 일반적이다. 파이프라인 방식 중에서 MDC (Multipath Delay Commutator)방식은 다중 경로(multi-path)를 통해 데이터를 전달하며, 지연 교환기로 SFG에 해당하는 데이터를 정렬한다. 따라서, 다른 파이프 라인 방식에 비해 가장 큰 수율을 갖고 있지만, 경로마다 각각의 복소승산기를 요구하기 때문에 면적이 커지는 단점이 있다. 그러나, 제안된 Radix-4/2 FFT 알고리즘은 다른 알고리즘에 비해 복소승산기의 수를 크게 줄이는 특징이 있기 때문에, MDC 방식으로 구현하면 면적과 수율 면에서 동시에 이득을 얻을 수 있다. 그림 1은 설계된 FFT 프로세서의 전체 블록도를 보여준다. 표 1은 시놉시스 캐드 툴을 이용하여 0.6 μm공정으로 합성 결과를 바탕으로 Radix-4 FFT 알고리즘과 Radix-4/2 FFT 알고리즘의 성능 비교 결과를 보여준다.

알고리즘	수 율 (입력 데이터율 R 가정)	면 적 (논리합성 결과)
Radix-4/2	$4R$	46,311
Radix-4	$4R$	60,118

표 1. 제안된 Radix-4/2과 Radix-4 알고리즘과의 성능 비교

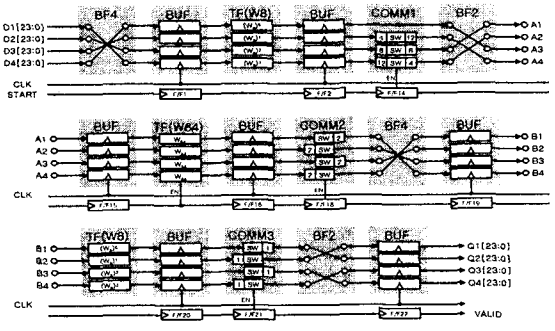

그림 1. MDC 방식의 Radix-4/2 FFT 프로세서의 블록도

Ⅴ. 결론

본 논문에서는 고속 무선 통신의 변조 방법인 OFDM 방식에 적합한 고속 및 면적 효율적인 FFT 알고리즘을 개발하였고, 이를 하드웨어로 구현 및 검증하였다. 제안된 Radix-4/2 알고리즘은 Radix-4 버터플라이 연산자를 기반으로 구현되므로 기존의 Radix-2^3 알고리즘에 비해 2배 빠른 수율을 갖는다. 수율의 비교를 위해, Radix-2 및 Radix-4 알고리즘을 MDC 파이프라인 방식에 적용하여 비교 분석하였고, 이를 통해 제안된 알고리즘이 기존의 기존의 Radix-2^3 알고리즘에 비해 2배 빠른 수율을 갖음을 확인하였다. 면적의 비교를 위해, 제안된 Radix-4/2 알고리즘과 같은 수율을 갖는 Radix-4 알고리즘을 이용한 FFT 프로세서를 설계하였다. 설계 후 논리합성 결과, 기존의 Radix-4 알고리즘을 이용한 FFT 프로세서의 경우, 약 60,118개의 논리 게이트로 구성됨을 확인하였으며, 제안된 알고리즘을 이용하여 구현한 프로세서와 비교한 결과, 약 23% 정도의 면적 감소가 있음을 확인하였다.

참고문헌(또는 Reference)

[1] Lawrence R.Rabiner and Bernard Gold, "Theory and Application of Digital Signal Communication" Printice-Hall, Ins., 1995.

[2] Shousheng He and Mats Torkelson, "A New Approach to Pipeline FFT Processor," Parallel Processing Symposium, pp.766-770, 1996

[3] Shousheng He and Mats Torkelson, "Designing Pipeline FFT Processor for OFDM (de)Modulation," ISSSE 98, Vol.2, pp.945-950, 1998.