JKIICE Journal of the Korea Institute of Information and Communication Engineering

한국정보통신학회논문지(J. Korea Inst. Inf. Commun. Eng.) Vol. 17, No. 9 : 2121~2126, Sep. 2013

Ultra-long FFT를 위한 Radix-2 기반 구조

강형주*

Radix-2 Based Structure for Ultra-long FFT

Hyeong-Ju Kang*

School of Computer Science and Engineering, Korea University of Technology and Education, Cheonan 330-708, Korea

요 약

본 논문에서는 32768-point FFT에서 radix-2에 기반한 구조들을 비교한다. Radix-2에 기반한 radix-2^k 구조들은 버터플라이가 단순하면서 곱셈기의 수를 줄일 수 있어서 많이 이용되고 있다. 본 논문에서는 근래에 많이 연구되고 있는 ultra-long FFT 중 대표적인 32768-point FFT에 대해 다양한 radix-2^k 구조를 적용하였다. 합성했을 때의 복잡도와 SQNR 성능을 비교한 결과 radix-2^4 구조가 가장 적합함을 보였다.

ABSTRACT

This paper compares radix-2 based structures for 32768-point FFT. Radix-2^k structures have been widely used because the butterfly is simple and the number of multipliers can be reduced in those structures. This paper applied various radix-2^k structures to 32768-point FFT that is representing ultra-long FFT. The ultra-long FFT has been studied much recently. This paper shows that the radix-2^4 structure is the most adequate because it shows the smallest complexity in the synthesis and the best SQNR performance. should be placed here.

키워드 : FFT, Ultra-long FFT, Radix-2^k 구조

Key word : FFT, Ultra-long FFT, Radix-2^k structure

접수일자 : 2013. 05. 15 심사완료일자 : 2013. 06. 12 게재확정일자 : 2013. 06. 27

* **Corresponding Author** Hyeong-Ju Kang(E-mail:hjkang@koreatech.ac.kr, Tel:+82-41-560-1420)
School of Computer Science and Engineering, Korea University of Technology and Education, Cheonan 330-708, Korea

Open Access http://dx.doi.org/10.6109/jkiice.2013.17.9.2121 print ISSN: 2234-4772 online ISSN: 2288-4165

Ⅰ. 서 론

Fast Fourier Transform(FFT)는 가장 널리 사용되는 디지털 신호처리 알고리즘 중 하나이다. FFT의 적용 분야는 매우 넓으며, 많은 현대 통신 표준들에서 사용되고 있는 Orthogonal Frequency Division Multiplexing (OFDM)이 대표적인 예이다. OFDM에서는 신호의 변복조에서 FFT가 중요한 역할을 수행한다.

OFDM 방식의 통신 표준에서는 전송하는 데이터의 전송률이 FFT의 길이에 비례하게 된다. 그러므로 통신 표준에서는 점점 더 긴 FFT를 사용하고 있다. 예를 들어 유럽형 지상파 TV 표준인 DVB-T에서는 최대 8192 point FFT를 사용하도록 되어 있으나, 그 다음 표준인 DVB-T2에서는 32768 point FFT 까지 사용하도록 되어 있다. 이와 같이 기존의 8192 point 이상의 FFT를 ultra-long FFT라고 부르며 이에 대한 연구도 많이 진행되어 왔다[1-3].

FFT의 구조에 대해서는 많이 연구되어 왔으며, 크게 메모리 기반 구조와 파이프라인 기반 구조로 나눌 수 있다. 메모리 기반 구조에서는 처리할 모든 데이터를 메모리에 넣은 뒤, 데이터를 꺼내어 처리한 뒤 다시 저장하는 것을 반복한다. 데이터 처리 유닛을 한 개 사용할 수도 있고, 여러 개를 사용하여 병렬적으로 처리할 수도 있다. 이에 반해 파이프라인 구조에서는 여러 개의 처리 유닛이 직렬적으로 연결되어서 데이터가 여러 단계를 직렬적으로 거치며 처리된다. FFT의 길이가 길수록 파이프라인 구조를 많이 사용한다.

FFT에서의 처리 유닛은 버터플라이 부분과 twiddle factor 곱셈기 부분으로 나눌 수 있다. 버터플라이 부분에서는 여러 개의 데이터를 서로 더하거나 빼는 동작을 수행한다. 버터플라이에서 처리된 데이터는 twiddle factor와 곱해진다.

FFT의 구조는 버터플라이에서 몇 개의 데이터를 처리하느냐에 따라서도 나눌 수 있다. 일반적으로 2개나 4개, 8개의 데이터를 처리하는 구조를 많이 사용한다. 각각의 구조를 radix-2, radix-4, radix-8 이라고 부른다.

Radix-2 구조는 버터플라이 부분의 구조가 단순한 반면에 twiddle factor 곱셈기의 개수가 늘어나는 단점이 있으며, radix가 커질수록 버터플라이 부분이 복잡해지고 곱셈기의 수가 줄어든다. 예전에는 이를 절충하여 radix-4를 많이 사용하였다.

그러나 radix-2 구조와 radix-4 구조의 장점을 취할 수 있는 radix-2^2가 발표된 이후에는 radix-2^k 구조가 많은 관심을 끌고 있다[4,5].

Radix-2^k 구조는 버터플라이 부분은 radix-2 구조와 동일하게 유지하면서 twiddle factor의 배치를 바꾸어 실제로 의미가 있는 곱셈기의 수를 줄이는 방법이다. Radix-2^2 구조가 발표된 이후로 radix-2^3, radix-2^4, radix-2^5 구조 등이 발표되어 적용되었다[6,7].

이와 같이 radix-2 기반의 파이프라인 구조에 대해 많이 연구되어 왔으나, FFT를 구현했을 때 각 구조가 복잡도와 FFT 성능에 어떤 영향을 미치는 지에 대해 종합적으로 비교한 예는 적은 편이다. 특히 최근에 점점 더 많이 이용되고 있는 ultra-long FFT에 대한 비교는 매우 적다고 할 수 있다. 이 논문에서는 ultra-long FFT 중 대표적인 32768-point FFT에 대해 radix-2 기반 구조들의 복잡도와 성능을 비교할 것이다. 이를 통해 ultra-long FFT에 적합한 구조를 찾을 것이다.

본 논문은 다음과 같이 구성되어 있다. 2장에서는 FFT에 대해 고찰하고, 3장에서는 radix-2^k 구조를 ultra-long FFT에 적용한다. 4장에서 실험결과를 보이고, 5장에서 결론을 맺을 것이다.

Ⅱ. Fast Fourier Transform

디지털 푸리에 변환(DFT)는 다음과 같은 식을 이용하여 시간 영역의 디지털 신호를 주파수 영역의 신호로 바꾸는 기법이다[4,5].

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-j2\pi nk/N}, \text{ for } 0 \leq k < N \qquad (1)$$

위 식에서 x(n)은 시간영역의 신호이며 X(k)는 주파수 영역의 신호이다. 이 식에서 $e^{-j2\pi nk/N}$는 간단하게 W_N^{nk}로 적는다. 위의 DFT를 그대로 구현하면 연산 복잡도가 매우 크므로 이를 줄이기 위해 Cooley-Tukey 알고리즘이 발표되었다[8]. 이 알고리즘에서는 DFT 알고리즘을 다음과 같이 변형한다.

$$X(k_1+N_1k_2)=\sum_{n_2=0}^{N_2-1}\left[\left(\sum_{n_1=0}^{N_1-1}x(N_2n_1+n_2)\,W_{N_1}^{k_1n_1}\right)W_N^{k_1n_2}\right]W_{N_2}^{k_2n_2} \quad (2)$$

$$0 \leqq n_1,k_1 < N_1,\ 0 \leqq n_2,k_2 < N_2,\ N=N_1\times N_2$$

일반적으로 N, N_1, N_2는 모두 2의 누승이라고 가정한다. 위 식에서는 안쪽의 합이 N_1-point의 DFT를 의미하며 바깥쪽의 합이 N_2-point의 DFT를 의미한다. 안쪽 N_1-point의 DFT는 총 N_2번 실행되며, 바깥쪽 N_2-point의 DFT는 총 N_1번 실행된다. 즉, Cooley-Tukey 알고리즘에서는 N-point의 큰 DFT를 N_1-point의 DFT와 N_2-point의 DFT의 작은 DFT들을 여러 번 하는 구조로 변형하는 것이다. 그리고 이러한 변형을 재귀적으로 반복하면 DFT의 시간 복잡도를 크게 줄일 수 있으며, 이를 FFT라고 부른다.

FFT를 유도하는 과정에서 하위 DFT의 길이를 재귀적으로 더 작은 값으로 나눌 때, 가장 작은 DFT의 길이를 그 구조의 radix라고 부른다. 즉 radix-2 구조에서는 모든 최하위 DFT의 길이가 2이고, radix-4 구조에서는 모든 최하위 DFT의 길이가 4이다. 그림 1은 radix-2 구조의 16-point FFT이다.

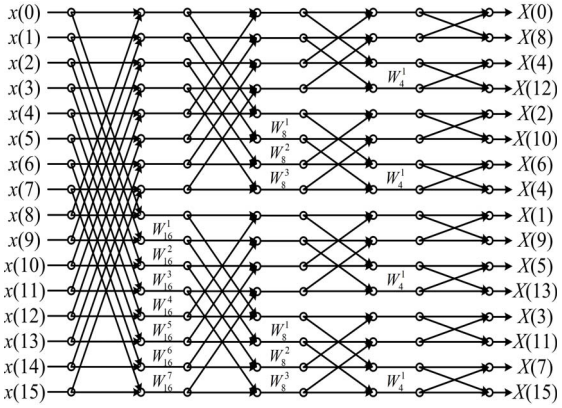

그림 1. Radix-2 16-point FFT
Fig. 1 Radix-2 16-point FFT

최하위 DFT는 보통 하나의 유닛으로 구현하며 이를 버터플라이라고 부른다. 그림 2는 radix-2와 radix-4에 대한 버터플라이 부분이다. Radix-2의 버터플라이는 두 개의 데이터를 서로 더하거나 빼는 동작을 하며, radix-4의 버터플라이는 네 개의 데이터를 서로 더하거나 뺀다. Radix-4의 버터플라이에서 허수 j를 곱하는 부분이 있으나 이는 실수부와 허수부를 바꾸고 새로운 실수부에 -1을 곱하는 것으로 바꿀 수 있다. 그림의 radix-4 버터플라이에서는 그림이 복잡하여 허수 j를 곱하는 등의 기호를 생략하였다.

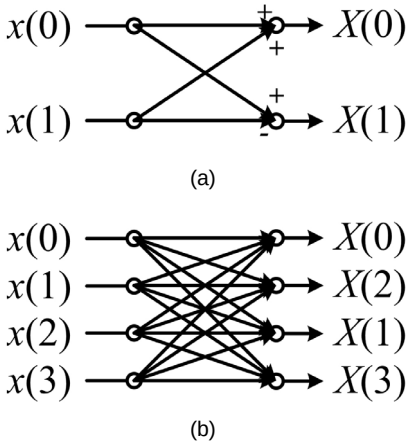

그림 2. Radix-2 버터플라이(a)와 radix-4 버터플라이(b)
Fig. 2 Radix-2 butterfly(a) and radix-4 butterfly(b)

하나의 DFT를 두 개의 하위 DFT로 나누는 과정에서 두 하위 DFT 사이에는 twiddle factor 곱셈을 수행해야 한다. 식(2)의 $W_N^{k_1n_2}$가 twiddle factor이다. 이 twiddle factor는 주로 ROM에 저장을 해 놓았다가 필요한 factor를 읽어서 곱한다.

Ⅲ. Radix-2 기반 구조와 Ultra-long FFT

3.1. Radix-2 기반 파이프라인 구조

FFT는 크게 메모리 기반 구조와 파이프라인 구조로 구현할 수 있다. 메모리 기반 구조에서는 N-point의 데이터를 모두 메모리에 넣은 뒤 하나 씩 읽어서 그림 1의 연산을 수행하고 다시 저장한다. 이에 반해 파이프라인 구조는 그림 의 각 단계 별로 하나의 버터플라이를 배치하고 이 버터플라이들을 직렬로 연결하여 수행하는 방식이다. Ultra-long FFT와 같이 길이가 긴 FFT에서는 주로 파이프라인 구조를 사용한다.

파이프라인 구조에서 한 단은 그림 3과 같이, 버터플라이 한 개와 데이터의 순서를 맞추기 위한 FIFO, twiddle factor ROM, twiddle factor 곱셈기로 이루어진다. Radix-r 구조에서 단계의 수는 $\log_r N$이므로, 버터플라이와 twiddle factor 곱셈기의 총 개수는 $\log_r N$개이다. FIFO의 총 양은 radix에 상관없이 일정하다.

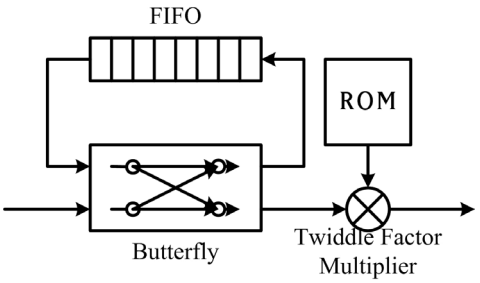

그림 3. 파이프라인 구조의 단
Fig. 3 A stage of pipeline structure

Radix-r에서 r이 커질수록 버터플라이와 twiddle factor 곱셈기의 수는 줄어들지만, 버터플라이 자체의 크기는 늘어나게 된다. 특히 radix-8 부터는 버터플라이 안에 복소수 곱셈기가 들어가게 되므로, radix-4 구조를 기존에는 많이 이용하였다.

Radix-2 구조의 장점인 간단한 버터플라이 구조와 radix-4 구조의 장점인 적은 곱셈기 개수를 결합한 것이 radix-2^2 구조이다. 이 구조에서는 radix-4 구조와 같이 DFT를 분해한 뒤, 최하위 4-point DFT를 다신 두 개의 2-point DFT로 나눈 형태이다. Radix-2^2 구조는 버터플라이 부분을 비롯한 전체적인 파이프라인 구조는 radix-2 구조와 동일하나 twiddle factor의 배치만 다르며, twiddle factor 곱셈기 중 절반이 허수 j를 곱하는 단순 곱셈이어서 곱셈기 개수가 radix-4 구조와 같아진다. Radix-2^2 구조가 제안된 이후로 radix-2^3, radix-2^4, radix-2^5 구조 등으로 확장되어 왔다.

3.2. Ultra-long FFT

요구되는 통신 대역폭의 증가에 따라 최근 이용되기 시작하고 있는 ultra-long FFT는 16384-point 이상의 데이터를 처리한다. 대표적으로 사용되고 있는 32768-point FFT에 대해 twiddle factor의 배치를 정리하면 표1과 같다.

표 1. Radix-2^k 구조의 twiddle factor 배치
Table. 1 Twiddle factor arrangement of radix-2^k structure

stage	radix-2^2	radix-2^3	radix-2^4	radix-2^5
0	W_4	W_4	W_4	W_4
1	W_{32768}	W_8	W_{16}	W_{32}
2	W_4	W_{32768}	W_4	W_4
3	W_{8192}	W_4	W_{32768}	W_8
4	W_4	W_8	W_4	W_{32768}
5	W_{2048}	W_{4096}	W_{16}	W_4
6	W_4	W_4	W_4	W_{32}
7	W_{512}	W_8	W_{2048}	W_4
8	W_4	W_{512}	W_4	W_8
9	W_{128}	W_4	W_{16}	W_{1024}
10	W_4	W_8	W_4	W_4
11	W_{32}	W_{64}	W_{128}	W_{32}
12	W_4	W_4	W_4	W_4
13	W_8	W_8	W_8	W_8
14	-	-	-	-

Twiddle factor $W_N^{k_1 n_2}$ 를 곱하기 위해서는 cos $2\pi\, k_1 n_2/N$ 와 sin $2\pi\, k_1 n_2/N$ 들을 ROM에 저장하고 있어야 한다. 인자 k_1과 n_2의 범위에 따라 ROM에 저장할 상수의 개수가 달라질 것이다. 그러나 몇몇 연구에 따르면 삼각함수의 대칭성을 이용하여 $N/8+1$개의 cos 값과 sin 값들만을 저장하여 모든 필요한 값들을 생성할 수 있음이 밝혀졌다[9]. 이를 이용하여 표1의 각 단에서 필요한 ROM entry의 개수를 계산하면 표2와 같다.

Radix-2 에서 누승이 커질수록 필요한 ROM의 entry 개수가 줄어든다는 것을 알 수 있다. W_8을 구현할 때 ROM을 사용하지 않고 상수 곱셈기로 대체하여 ROM을 제거할 수 있으므로 W_8을 곱하는 단에서도 ROM의 크기는 0으로 하였다.

Radix-2^k 구조에서 k와 곱셈기 개수의 관계는 일정하지 않은 면이 있다. W_4를 곱하는 것은 1 또는 $-j$를 곱하는 것이므로 곱셈기를 사용하지 않고 구현할 수 있다. W_8은 앞에서 논의했듯이 상수곱셈기로 대체가 가능하다. 이를 정리하여 표2의 하단에 제시하였다. 복소수 곱셈기의 개수는 radix-2^3 구조가 가장 적으나 이 구조는 대신 W_8곱셈기의 개수가 가장 많다. 곱셈기의 개수로

만 보면 radix-2^4 구조가 오히려 유리한 면이 있을 것으로 보인다.

표 2. Radix-2^k 구조의 복잡도
Table. 2 Complexity of radix-2^k structure

stage	radix-2^2	radix-2^3	radix-2^4	radix-2^5
0	0	0	0	0
1	4096	0	2	4
2	0	4096	0	0
3	1024	0	4096	0
4	0	0	0	4096
5	256	512	2	0
6	0	0	0	4
7	64	0	256	0
8	0	64	0	0
9	16	0	2	128
10	0	0	0	0
11	4	8	16	4
12	0	0	0	0
13	0	0	0	0
14	-	-	-	-
계	5460	4680	4374	4236
복소수 곱셈기	6	4	5	5
W_8 곱셈기	1	5	1	3

Ⅳ. 실험 결과

앞 절에서의 논의와 같이 여러 radix-2^k 구조는 k의 값에 따라 그 복잡도가 달라지며, twiddle factor 곱셈의 위치가 달라짐에 따라 성능도 달라질 것이다. 이런 구조들을 32768-point FFT에 적용하여 비교하였다.

내부의 데이터 형태는 semi-floating-point 형태로써, 실수부와 허수부가 exponent를 공유하는 형태이다. 입력 데이터에서 mantissa의 폭은 실수부와 허수부가 각각 7bit이고 exponent는 2bit로 하였다. mantissa의 폭은 butterfly를 한 번 지날 때 마다 1bit 씩 증가시켰다. 최종 출력 데이터에서 mantissa의 폭은 실수부와 허수부가 각각 12bit이고 exponent는 3bit로 하였다. Twiddle factor의 폭은 12bit로 하였다.

구현의 복잡도를 비교하기 위하여 radix-2^k 구조들을 RTL 수준에서 구현한 뒤 합성하였다. 합성 라이브러리는 0.18㎛ 공정을 사용하였고 Cadence사의 RTL Compiler로 합성하였다.

표 3에서 첫 번째 행이 RTL Compiler에서 합성결과로 출력하는 면적을 비교한 것이다. FIFO의 면적은 모든 구조에서 동일하므로 제외하였다. 표에 따르면 radix-2^4 구조의 면적이 제일 적다. ROM의 entry 숫자는 radix-2^5 구조가 더 적으나 그 양에는 큰 차이가 없고, 곱셈기의 숫자가 radix-2^4 구조에서 더 적으므로 이와 같은 결과가 나왔다. 그러나 그 차이는 매우 작으므로 두 구조의 복잡도는 별 차이가 없다고 할 수 있다.

표 3. Radix-2^k 구조의 실험결과
Table. 3 Experimental results of radix-2^k structure

	radix-2^2	radix-2^3	radix-2^4	radix-2^5
면적 (μm^2)	1,293,409	1,229,675	1,194,506	1,198,047
SQNR (dB)	46.85	47.19	47.11	47.27

표 3의 두 번째 행은 SQNR을 비교한 것으로서 부동소수점 FFT와 구현된 FFT 사이의 양자화 노이즈를 측정하여 계산하였다. SQNR의 비교에서도 radix-2^4 구조가 다른 구조와 거의 비슷한 성능을 보여주었다. 면적과 성능을 보았을 때, radix-2^2 구조는 다른 구조들에 비해 면적과 성능이 비교적 큰 차이를 보였으나, 나머지 세 개의 구조는 거의 비슷하였다.

Ⅴ. 결 론

본 논문에서는 대표적인 ultra-long FFT인 32768-point FFT에 대해 다양한 radix-2^k 구조를 적용하여 복잡도와 성능을 비교하였다. Radix-2^k 구조는 버터플라이의 구조가 간단하면서도 곱셈기의 수를 줄일 수 있어서 많이 이용되는 구조이다. Radix-2^2 구조부터 radix-2^5 구조까지 제안되어 왔으며, 본 논문에서 비교한 결과 32768-point FFT에는 radix-2^4 구조가 가장 적합하다는 것이 밝혀졌다.

REFERENCES

[1] H. Chen, Q. Wu, Z. Ciao, and H. Wan, “A pipelined memory-efficient architecture for ultra-long variable size FFT processors,” in *Proceedings of International Conference on Computer Science and Informmation*, pp. 357-316, 2008.

[2] S.-Y. Lin, C.-L. Wey, and M.-D. Shieh, “Low-cost FFT Processor for DVB-T2 Applications,” *IEEE Transactions on. Consumer Electronics*, vol. 56, no. 4, pp. 2072-2079, 2010.

[3] M. Turrillas, A. Cortes, I. Velez, J. F. Sevillano, and A. Irizar, “An FFT core for DVB-T2 receivers,” in *Proceedings of International Conference on Electronics*, Circuits and Systems, pp. 120-123, 2008.

[4] S. He and M. Torkelson, “A new approach to pipeline {FFT} processor,” in *Proceedings of International Parallel Processing Symposium*, pp. 766-770, 1996.

[5] S. He and M. Torkelson, “Design and implementation of a 1024-point pipeline FFT processor,” in *Proceedings of IEEE Custom Integrated Circuits Conference*, pp. 131-134, 1998.

[6] S. He and M. Torkelson, “Designing Pipeline FFT Processor for OFDM (de)Modulation,” in *Proceedings of IEEE International Symposium on Signals, Systems, and Electronics*, pp. 257-262, 1998.

[7] A. Cortes, I. Velez, and J. F. Sevillano, “Radix r^k FFTs: matricial representation and SDC/SDF pipeline implementation,” *IEEE Transactions on Signal Processing*, vol. 57, no. 7, pp. 2824-2839, 2009.

[8] J. W. Cooley and J. W. Tukey, “An algorithm for machine computation of complex Fourier series,” *Math. Comp.*, vol. 19, pp. 297-301, 1965.

[9] M. Hasan and T. Arslan, “Scheme for reducing size of coefficient memory in FFT processor,” *Electronics Letters*, vol. 38, no. 4, pp. 163-164, 2002.

강형주(Hyeong-Ju Kang)

1998년 한국과학기술원 전기및전자공학과 학사
2000년 한국과학기술원 전기및전자공학과 석사
2005년 한국과학기술원 전자전산학과 박사
2005년 ~ 2006년 (주)매그나칩반도체 선임연구원
2006년 ~ 2009년 (주)지씨티리써치 선임연구원
2009년 ~ 현재 한국기술교육대학교 컴퓨터공학부 전임강사/조교수
※관심분야 : VLSI설계 및 CAD, 마이크로프로세서 설계, 통신 모뎀 설계