ISSN : 1226-7244 (Print)
ISSN : 2288-243X (Online)
논문번호 21-04-15
114

j.inst.Korean.electr.electron.eng.Vol.25,No.4,697~702,December 2021
https://doi.org/10.7471/ikeee.2021.25.4.697

동적 스케일링에 기반한 낮은 복잡도의 2048 포인트 파이프라인 FFT 프로세서

2048-point Low-Complexity Pipelined FFT Processor based on Dynamic Scaling

김 지 훈*★

Ji-Hoon Kim*★

Abstract

Fast Fourier Transform (FFT) is a major signal processing block being widely used. For long-point FFT processing, usually more than 1024 points, its low-complexity implementation becomes very important while retaining high SQNR (Signal-to-Quantization Noise Ratio). In this paper, we present a low-complexity FFT algorithm with a simple dynamic scaling scheme. For the 2048-point pipelined FFT processing, we can reduce the number of general multipliers by half compared to the well-known radix-2 algorithm. Also, the table size for twiddle factors is reduced to 35% and 53% compared to the radix-2 and radix-2^2 algorithms respectively, while achieving SQNR of more than 55dB without increasing the internal wordlength progressively.

요 약

고속 푸리에 변환(Fast Fourier Transform, FFT)은 다양한 응용처에서 널리 사용되는 주요 신호처리 블록이다. 일반적으로 1024 포인트 이상의 긴 FFT 처리의 경우 높은 SQNR(Signal-to-Quantization Ratio)를 유지하면서도 낮은 하드웨어 복잡도의 구현이 매우 중요하다. 본 논문에서는 낮은 복잡도의 FFT 알고리즘과 간단한 동적스케일링 기법을 제시한다. 이를 통해 2048 포인트 FFT연산에 대해서 널리 알려진 radix-2 알고리즘에 비해 곱셉기의 수를 절반으로 줄일 수 있으며, 또한 twiddle factor를 저장하기 위해 필요한 테이블의 크기를 radix-2 및 radix-2^2 알고리즘에 비해 각각 35% 및 53%로 축소할 수 있다. 그리고 내부 데이터의 폭을 점진적으로 늘리지 않고서도 55dB 이상의 높은 SQNR을 달성하는 것을 확인하였다.

Keywords : Fast Fourier Transform (FFT), Pipelined Processor, SQNR, Twiddle Factor, Low-Complexity

Ⅰ. 서론

고속 푸리에 변환(Fast Fourier Transform, FFT)는 통신시스템을 비롯한 다양한 신호처리응용에서 널리 사용되는 핵심적인 블록으로서, 특히 직교 주파수 분할 다중화의 경우에는 1024 포인트 이상의 긴 FFT 연산을 요구하고 있어서 이에 대한 낮은 복잡도의 하드웨어 구현이 매우 중요하다. 이를 위

* Dept. of Electrical and Information Eng., SeooulTech
★ Corresponding author
E-mail : jihoonkim@seoultech.ac.kr, Tel : +82-2-3277-4321
※ Acknowledgment
This study was financially supported by Seoul National University of Science and Technology.
Manuscript received Dec. 20, 2021; revised Dec. 22, 2021; accepted Dec 27, 2021.

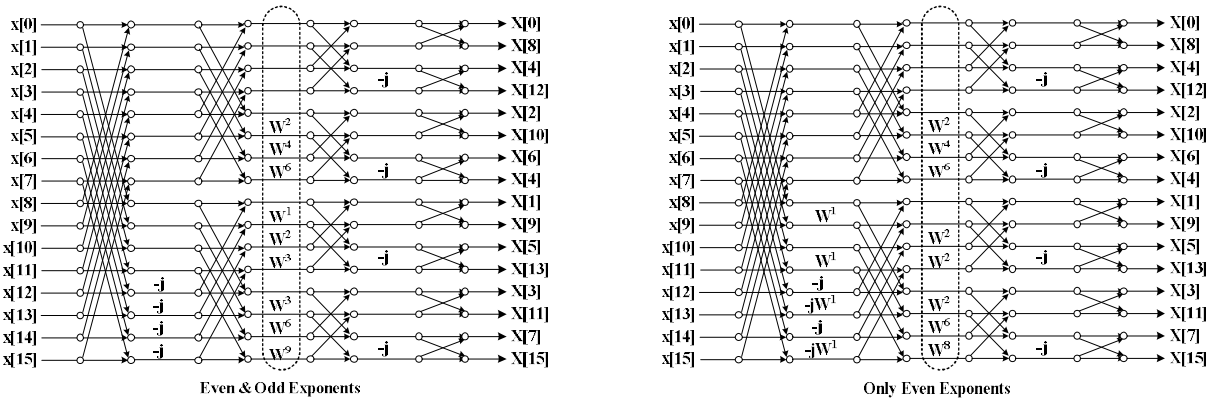


Fig. 1. Signal flow graphs of 16-point FFT. (a) Radix-2^2 algorithm and (b) the proposed algorithm.
그림 1. 16 포인트 FFT에 대한 Signal flow graph 비교 (a) radix-2^2 알고리즘 및 (b) 제안하는 알고리즘

해, 널리 알려진 radix-2 및 radix-4 알고리즘 외에 radix-2^2[1], radix-2^3[2] 등의 다양한 FFT 알고리즘들이 발표되었으며 이를 통해 곱셈기와 덧셈기 등의 연산기 개수를 줄이는 것에는 효과적이었으나 긴 FFT연산에서 새롭게 문제가 되는 twiddle factor를 저장하기 위한 테이블 사이즈에 대한 고려는 많이 다루어지지 않았다. 또한, 이와는 별개로 실제 고정소수점 방식으로 연산을 진행하며 내부 연산으로 인해 지속적으로 비트폭을 늘려가야만 충분한 정확도를 얻을 수 있으며, 이로 인해 높은 SQNR (Signal-to-Quantization Noise Ratio)을 얻기 위해서는 연산기의 하드웨어 복잡도가 증가하게 되는 문제점을 여전히 가지고 있다[3].

본 논문에서는 가장 널리 사용되는 radix-2^2 알고리즘과 유사한 연산기 개수를 가지면서도 twiddle factor를 저장하기 위한 테이블의 크기를 절반가까이 줄일 수 있는 새로운 FFT 알고리즘을 제안한다. 또한, 파이프라인 기반의 FFT 프로세서 구현에서 매 스테이지마다 지속적으로 데이터를 표현하기 위한 비트 폭을 넓혀나가는 방식이 아닌 오버플로우가 발생하는 경우에만 동적으로 크기를 조절하는 방법을 통해서 낮은 하드웨어 복잡도로 높은 SQNR을 달성하는 방법을 적용하였다.

II. 제안하는 FFT 알고리즘

신호 $x(n)$에 대한 N-포인트 Discrete Fourier Transform (DFT)는 다음과 같이 주어진다.

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

여기서 $x(n)$와 $X(k)$는 복소수이며, twiddle factor는 다음과 같이 정의된다.

$$W_N^{kn}=e^{-j\left(\frac{2\pi kn}{N}\right)}=\cos\left(\frac{2\pi kn}{N}\right)-j\sin\left(\frac{2\pi kn}{N}\right) \tag{2}$$

여기서 n과 k를 다음과 같이 3차원 index로 변경하면,

$$\begin{aligned} n&=\frac{N}{2}n_1+\frac{N}{4}n_2+n_3\\ k&=k_1+2k_2+4k_3 \end{aligned} \tag{3}$$

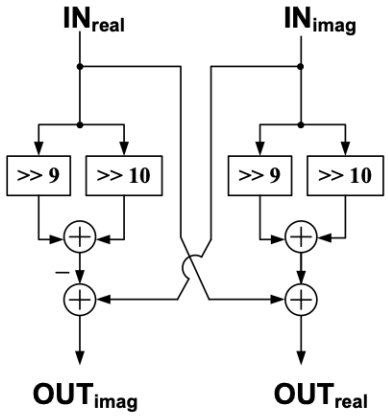
수식 (1)은 다음과 같이 다시 쓰여질 수 있다.

$$\begin{aligned} X(k)&=X(k_1+2k_2+4k_3)\\ &=\sum_{n_3=0}^{N/4-1}\sum_{n_2=0}^{1}\left\{B(\frac{N}{4}n_2+n_3,k_1)\,W_N^{(\frac{N}{4}n_2+n_3)k_1}\right\}\\ &\qquad\bullet\ W_N^{(\frac{N}{4}n_2+n_3)(2k_2+4k_3)}\\ &=\sum_{n_3=0}^{N/4-1}\left\{H(k_1,k_2,n_3)\,W_N^{2m(k_1+2k_2)}\right\}W_N^{4n_3k_3} \end{aligned} \tag{4}$$

여기서 $B(\cdot)$은 다음과 같은 FFT의 butterfly 구조를 표현한다.

$$\begin{aligned} B(\frac{N}{4}n_2+n_3,k_1)&=x(\frac{N}{4}n_2+n_3)\\ &\quad+(-1)^{k_1}x(\frac{N}{4}n_2+n_3+\frac{N}{2}) \end{aligned} \tag{5}$$

이때, n_3의 값이 짝수(=2m)인 경우 $H(\cdot)$은 다음



$$W_{2048}^{1} \simeq 1-j(2^{-9}+2^{-10})$$

Fig. 2. Low-complexity constant multiplier for W_{2048}^{1}.
그림 2. W_{2048}^{1} 곱셈기에 대한 낮은 복잡도 구현

과 같이 주어지며

$$H(k_1,k_2,n_3) = B(n_3,k_1) + (-1)^{k_2}(-j)^{k_1}B(n_3+\frac{N}{4},k_1) \quad (6)$$

n_3의 값이 홀수(=2m+1)인 경우 $H(\cdot)$은 다음과 같이 주어진다.

$$H(k_1,k_2,n_3) = [W_N^{k_1}B(n_3,k_1) + (-1)^{k_2}(-j)^{k_1}W_N^{k_1}B(n_3+\frac{N}{4},k_1)]W_N^{2k_2} \quad (7)$$

k_1은 0 아니면 1의 값을 가지기에, 수식 (6)은 n_3가 짝수인 경우에는 butterfly가 입력쪽에서 $-j$에 해당하는 trivial multiplication만 가지는 것을 의미하며, 이는 복소수에서 실수부와 허수부를 바꿔주는 것으로 손쉽게 구현이 가능하다. 수식 (7)은 n_3이 홀수인 경우 W_N^1을 곱하는 상수곱셈기가 필요한 것을 의미하게 되는데, 대신 그림 1에서 볼 수 있는 것과 같이 twiddle factor들이 기존 방식과는 달리 짝수만 존재하게 되는 것을 알 수 있다. 이는 기존의 홀수와 짝수 모두의 지수를 갖는 twiddle factor를 저장해야 하는 방식대비, 절반의 twiddle factor만 저장해도 되는 것을 의미한다[5]. 이때 추가되는 상수곱셈기는 그림 2에 나타난 것과 같이 4개의 덧셈기만으로 간단하게 구현할 수 있다.

III. 2048 포인트 FFT 프로세서 구조

그림 3은 가장 널리 사용되는 파이프라인 FFT 프로세서의 형태인 SDF(Single-Delay Feedback) 구조를 토대로 한 2048 포인트 FFT 프로세서로서, BF는 수식(5)에 해당하는 butterfly연산을 담당한다[1]. 복잡도가 높은 일반 곱셈기의 개수를 줄일 수 있어서 널리 사용되는 radix-2^2 알고리즘을 기반으로 한 총 11개의 스테이지를 기반으로 한 구조에서, 앞에서 제안한 알고리즘을 twiddle factor를 저장하기 위한 테이블의 크기를 고려하여 처음 4개

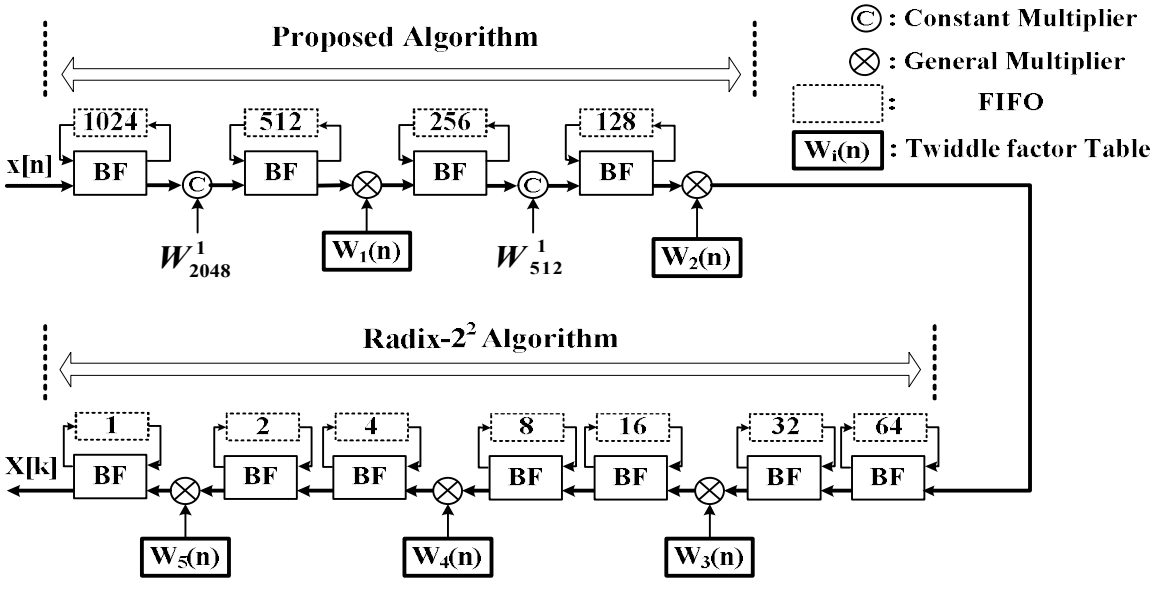


Fig. 3. SDF pipeline architecture for the proposed 2048-point FFT processor.
그림 3. 제안하는 2048 포인트 FFT 프로세서의 SDF 파이프라인 구조

Table 1. Hardware complexity comparison for 2048-point FFT.
표 1. 2048 포인트 FFT에 대한 하드웨어 복잡도 비교

FFT Algorithm	Constant Multiplier	General Multiplier	# of Entries in Table
Radix-2	0	10	1023 (100%)
Radix-2^2	0	5	682 (66.7%)
Radix-2^3	3	4	584 (57.1%)
Proposed + Radix-2^2	2	5	362 (35.4%)

Table 2. Data scaling configuration.
표 2. 데이터 스케일링 기법

Case	Data Scaling Method
I	Scaling-to-Half (for All Stages)
II	Stage 1 ~ 8 : No Scaling Stage 9 ~ 11 : Scaling-to-Half
III	Proposed Data Scaling (for All Stages)
IV	Stage 1 : No Scaling Stage 2 ~ 11 : Proposed Data Scaling

의 스테이지에 적용하고, 새로운 알고리즘으로 인한 twiddle factor 테이블 크기 감소량과 추가 상수 곱셈기의 복잡도를 고려하여 남은 7개의 스테이지에는 기존 널리 사용되는 radix-2^2 알고리즘을 적용하였다. 이 경우 요구되는 W_{512}^{1} 곱셈기 역시 그림 2와 마찬가지로 CSD(canonical signed digit) 방식으로 표현하여 몇 개의 덧셈기로만 손쉽게 구현할 수 있다[6]. 하지만 뒤쪽 스테이지로 갈수록 해당 알고리즘을 적용하는 경우 추가되는 상수 곱셈기의 복잡도가 높아지는 반면 줄어드는 twiddle factor 테이블의 크기가 작아지는 특성이 있기에, 제안하는 알고리즘은 표 1에 나와있는 것처럼 파이프라인 프로세서의 앞부분에 적용했을 때 가장 좋은 특성을 보이게 된다.

전체 2048 포인트 FFT 프로세서에 대해 알고리즘별 하드웨어 복잡도에 대한 비교는 표 1에 나와있으며, radix-2^2 알고리즘을 혼용하는 방식을 통해서 적은 수의 덧셈기로 구성된 상수 곱셈기를 추가하는 제안하는 알고리즘이 twiddle factor를 저장하기 위한 테이블의 요구사항이 전체에서 75%를 차지하는 처음 4개의 stage에서 그 크기를 절반으로 줄임으로써 전체 복잡도를 크게 줄일 수 있게 되는 것을 알 수 있다.

그림 3은 2048 포인트 FFT 프로세서에 제안하는 알고리즘을 앞부분에만 적용하는 하나의 예를 보여준 것이며, 실제 보다 긴 FFT 연산에서는 해당 알고리즘에서 추가되는 상수곱셈기의 복잡도는 더욱 낮으면서도 twiddle factor를 저장하기 위한 테이블의 크기는 더욱 줄일 수 있기에 보다 효과적이다. 입력단에서부터 몇 개의 스테이지에 제안하는 알고리즘을 적용할 것인가는 구현하는 공정에 따라서, 그리고 twiddle factor를 저장하기 위한 테이블을 해당 공정의 메모리 컴파일러를 통해서 구현할 것인지 혹은 조합회로로 구성할 것인지 등에 따라서 달라지게 된다.

Ⅳ. 데이터 스케일링 기법

앞에서 살펴본 바와 같이, 새로운 알고리즘을 통해서 twiddle factor를 저장하기 위한 테이블에서의 entry 개수와 곱셈기의 개수 감소등을 이루어낼 수 있음을 살펴보았다. 하지만, 실제 전체 하드웨어 복잡도는 추가적으로 이와 관련한 각 스테이지에서 사용되는 데이터의 비트폭과 밀접한 관련이 있다[7].

1. 기존 데이터 스케일링 기법

일반적으로 사용되는 기법은 크게 3가지로 나뉜다. 먼저 가장 손 쉬운 방법은 내부 데이터의 비트폭을 각 스테이지마다 1비트씩 늘려주는 'No Scaling'이다. 수식 (6)에 나와있듯이, butterfly연산은 덧셈/뺄셈을 동반하기에 이 과정에서 오버플로우를 고려하여 1비트씩을 늘려주는 것이다. 하지만 1024포인트 이상의 긴 FFT 연산에서는 스테이지의 수가 많아짐에 따라, 이 경우 뒤쪽 스테이지에서 비트폭이 매우 커지는 단점을 갖게 되고 이로 인한 연산기의 복잡도 또한 크게 증가하게 된다. 두 번째로 고려할 수 있는 방법은 이와 같은 비트폭의 증가를 막기 위해서 각 스테이지 연산이 끝나고 얻어진 결과를 오른쪽으로 1비트 쉬프트하여 절반으로 만드는 'Scaling-to-Half'이다. 이 경우에는 해당 스테이지 연산이 끝난 이후에도 데이터의 비트 폭이 늘어나지 않기에 뒤쪽 스테이지에서도 연산기의 복잡도가 늘어나지 않는 다는 장점이 있지만, 제한된 비트폭으로 인한 정확도의 손실로 인해 floating-point 연산대비 오차의 정도를 나타내는 SQNR이 낮아진다는 단점을 지니게 된다.

Table 3. SQNR and internal data bitwidth.
표 3. 각 스테이지별 비트폭 및 SQNR

Case	Stage Number											SQNR (dB)
	1	2	3	4	5	6	7	8	9	10	11	
I	12	12	12	12	12	12	12	12	12	12	12	18.9
II	12	13	14	15	16	17	18	19	19	19	19	56.1
III	12	12	12	12	12	12	12	12	12	12	12	51.2
IV	12	13	13	13	13	13	13	13	13	13	13	55.8

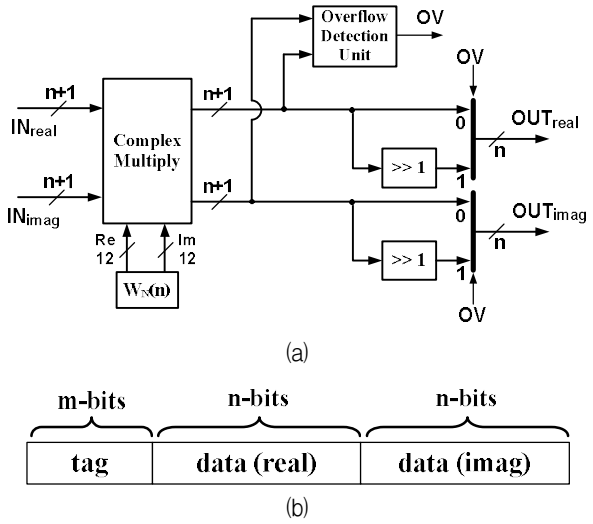


Fig. 4. (a) Multiply unit with simple dynamic scaling and (b) data format with tag value.
그림 4. (a) 간단한 동적 스케일링 기법을 포함한 butterfly 곱셈기 및 (b) tag값을 포함한 데이터 포맷

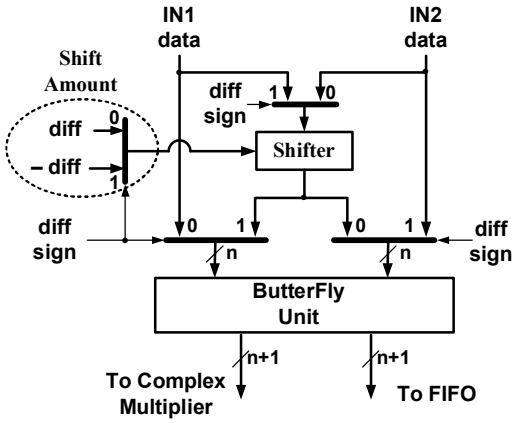


Fig. 5. Butterfly unit with tag processing.
그림 5. Tag값을 고려한 butterfly 연산기

표 2의 ***Case I***와 ***Case II***는 이와 같은 부분을 고려한 구성을 보여주고 있다. 표 3에서 볼 수 있듯이, 모든 스테이지에 Scaling-to-Half를 적용하면 내부 비트폭의 크기는 유지할 수 있지만 SQNR의 값이 20dB를 넘지 못하는 것을 알 수 있다(***Case I***). ***Case II***에서처럼 각 스테이지를 지나감에 따라서 1비트씩 데이터의 비트폭을 늘려주게 되면 55dB이상의 높은 SQNR을 통해서 정확도에서의 장점을 얻을 수 있으니, 뒤쪽 스테이지에서 사용되는 덧셈기 / 뺄셈기, 그리고 곱셈기와 값을 저장하기 위한 FIFO(First-Input First-Output) 등에서의 복잡도 증가를 피할 수 없게 된다.

마지막으로 널리 사용되는 기법중 하나는 BFP (Block Floating Point)기법으로서, 각 스테이지 마다 연산이 모두 끝나고나서 가장 큰 값을 기준으로 하여 정규화하는 방법이 있다[8]. 하지만, 이와 같은 방식은 해당 스테이지의 연산이 끝나기전에 다음 스테이지로 값을 전달하는 파이프라인 방식의 FFT 프로세서에서는 적합하지 않으며, 적용하기 위해서는 추가적인 데이터 버퍼링을 필요로 하게 되어 FFT를 수행하는데 소요되는 시간이 증가한다는 단점을 지니게 된다.

2. 제안하는 동적 데이터 스케일링 기법

그림 4(a)는 본 논문에서 적용한 간단한 동적 데이터 스케일링 기법을 포함한 butterfly 곱셈기이다. *n-bit* 입력을 받아서 butterfly 연산 이후 *(n+1)-bit*의 출력을 토대로, twiddle factor 를 곱한 이후에 해당 연산 결과가 *n-bit*으로 표현될 수 있는지 없는지 여부를 오버플로우를 확인하고, 만약 *n-bit*으로 표현할 수 없다면 해당 값은 그대로 다음 스테이지로 전달된다. 만약 오버플로우가 발생한다면 이는 n-bit으로 표현할 수 없음을 의미하기에, 이때 선택적으로 Scaling-to-Half기법을 적용하고 해당 부분을 그림 4(b)와 같이 tag부분에 표시해서 후속 연산을 진행할 때 Scaling이 몇 번 적용된 실수/허수 pair인지를 알 수 있다.

이 경우, 그림 4(b)의 tag부분이 FIFO에 저장되

면서 오버헤드로 작용하게 되지만, 그림 5에 나타나 있는 것처럼 해당 부분은 각 스테이지에서 연산을 할 때 butterfly연산기의 입력들이 가지고 온 tag값의 차이(=diff)값을 기반으로 자릿수를 맞추는 데만 사용되고 연산기의 복잡도를 높이지 않음을 알 수 있다. 또한, 해당 부분으로 인한 FIFO의 증가치는 표 2의 ***Case IV***와 같이 가장 큰 FIFO를 요구하는 첫 번째 스테이지에서는 해당 기법을 적용하지 않음으로써 복잡도 증거를 피할 수 있다.

표 2 및 표 3에 나타난 것과 같이, 해당 기법을 적용하게 되면 충분히 높은 SQNR을 제공하면서 뒤쪽 스테이지에서도 데이터 비트폭이 크게 증가하지 않음으로써 하드웨어 복잡도를 낮출 수 있음을 알 수 있다. 이와 같은 데이터 스케일링을 통한 연산기 자체의 하드웨어 복잡도를 낮추는 기법은 Ⅱ장 및 Ⅲ장에서 언급한 알고리즘 수준에서의 FFT 연산 복잡도를 낮추는 방식과 함께 적용될 수 있음을 확인할 수 있다.

Ⅴ. 결론

본 논문에서는 긴 FFT연산을 지원하는 파이프라인 FFT프로세서의 하드웨어 복잡도를 낮출 수 있는 알고리즘적인 방법과 동적 데이터 스케일링 기법을 제안하였으며, 이를 통해 2048포인트 SDF구조를 갖는 파이프라인 FFT 프로세서에서 소수의 덧셈기로 구성된 상수곱셈기를 추가함으로써 twiddle factor를 저장하기 위한 테이블의 크기를 radix-2 및 radix-2^2 알고리즘 대비 각각 35% 및 53% 수준으로 줄일 수 있음을 보였다. 또한, 동적 데이터 스케일링 기법을 스테이지에 선택적으로 적용함으로써 내부의 데이터 비트폭의 증가를 피하면서도 55dB 이상의 높은 SQNR을 달성할 수 있음을 확인하였다.

References

[1] S. He and M. Torkelson, “Design and Implementation of a 1024-point pipeline FFT processor,” in *Proc. IEEE Custom Integrated Circuits. Conf.*, pp.131-134, 1998.
DOI: 10.1109/CICC.1998.694922

[2] S. He and M. Torkelson, “Designing pipeline FFT processor for OFDM (de)modulation,” in *Proc. IEEE URSI Int. Symp. Signals, Syst. Electron.*, pp. 257-262, 1998.
DOI: 10.1109/ISSSE.1998.738077

[3] L. Pang, *et al.*, “Design and Implementation of High Performance FFT Processor with Radix-2^k Algorithm,” in Proc. *IEEE International Conference on Signal, Information and Data Processing*, 2019. DOI: 10.1109/ICSIDP47821.2019.9173055

[4] J. W. Cooley and J. W. Tukey, “An algorithm for the machine computation of the complex Fourier series,” *Math. Computation.* vol.19, pp.297-301, 1965. DOI: 10.1090/S0025-5718-1965-0178586-1

[5] I. C. Park, W. H. Son, and J. H. Kim., “Twiddle Factor Transformation for Pipelined FFT Processing,” in Proc. *IEEE International Conference on Computer Design*, 2007. DOI: 10.1109/ICCD.2007.4601872

[6] T. Nguyen and H. Lee, “Shared CSD complex constant multiplierfor parallel FFT processors,” in Proc. *International SoC Design Conference*, 2015. DOI: 10.1109/ISOCC.2015.7401648

[7] Y. W. Lin, H. Y. Liu and C. Y. Lee, “A 1-GS/s FFT/IFFT Processor for UWB Applications,” *IEEE J. Solid-State Circuits*, vol.40, no.8, pp.1726-1735, 2005. DOI: 10.1109/JSSC.2005.852007

[8] David Elam and Cesar Iovescu, “A Block Floating Point Implementation for an N-Point FFT on the TMS320C55x DSP,” *Application Report, SPRA948, Texas Instruments*, 2003.

BIOGRAPHY

Ji-Hoon Kim (Member)

2004 : BS degree in Electrical Engineering and Computer Science, KAIST.
2009 : PhD degree in Electrical Engineering, KAIST.
2009~2010 : Senior Engineer, Samsung Electronics.
2010~2016 : Assistant / Associate Professor, Chungnam National University.
2016~2018 : Associate Professor, SeoulTech.
2018~present : Professor, Ewha Womans University.