OFDM 무선 LAN 시스템에 적용할 FFT/IFFT 프로세서의 설계

권 병 철, 고 성 찬
안동대학교 정보통신공학과
전화 : 054-820-5162 / 핸드폰 : 017-803-1276

Design of FFT/IFFT processor that is applied to OFDM wireless LAN system

Byoung-Chul Kwon, Sung Chan Ko
Dept. of Information Communication, Andong University
E-mail : mirage97@hananet.net

Abstract

In this paper, we are designed and verified a FFT/IFFT processor that is possible from the wireless LAN environment which is adopted international standard of the IEEE802.11a. The proposed architecture of the FFT/IFFT has Radix-2 64point SDF(single-path delay feedback) Pipeline technique and DIF(Decimation in Frequency) structure. The FFT/IFFT processor has each 8 bit complex input-output and 6 bit Twiddle factor. we used Max+PlusII for simulation and can see that the processor is properly operated

I. 서론

앞으로의 통신 서비스는 이동이 자유로운 무선 멀티미디어 서비스 형태를 가지며 노트북과 PDA와 같은 이동성을 갖는 컴퓨터에 대한 요구가 급격히 증가하여 무선 LAN 시스템은 전세계에 걸쳐 시장을 형성하고 있다. 이에 미국 전기전자공학회(IEEE)가 지난 97년 무선LAN에 대한 최초의 국제적 표준인 IEEE802.11을 마련한 후 99년 802.11b와 함께 추가로 IEEE802.11a를 무선LAN 표준으로 확정했다. 802.11b는 2.4GHz대역을 사용해 11Mbps의 속도를 제공하지만 급격히 증가하는 무선 멀티미디어 서비스나 고속 무선 데이터 통신을 지원하기에는 미흡한 반면 802.11a는 5GHz대역을 사용하여 최고 54Mbps 속도를 제공한다. 또한 802.11a는 복수의 반송파를 사용해 주파수 이용효율을 높이는 OFDM 방식의 변조기술을 사용해 고속 데이터 전송에 적합한 것으로 평가받고 있으며 현재 시장에 공급된 무선 LAN 장비의 주를 이루는 802.11b 지원제품을 대체하는 차기 무선 LAN의 표준으로 떠오르고 있다.

따라서 본 논문은 IEEE802.11a의 OFDM 방식에서 핵심부분인 FFT/IFFT를 VHDL언어로 설계하고 시뮬레이션 하는 과정을 논의하고자 한다. FFT는 입력되는 방법에 따라 DIT(Decimation in Time), DIF(Decimation in Frequency)로 나누어지며 Radix를 증가시키면 성능은 향상되지만 하드웨어 복잡도와 용량이 증가한다. 따라서 Radix-2 DIF FFT 알고리즘을 이용한 FFT 프로세서를 설계 및 검증하고 이를 구현하였다.

II. FFT 알고리즘 개요

FFT(Fast Fourier Transform)은 DFT(Discrete Fourier Transform)을 보다 빨리 수행하기 위한 알고리즘이다. Radix-2 N-point Fourier transfer의 경우에, DFT는 N^2의 복소수 덧셈과 곱셈의 연산이 필요하지만 Radix-2 N point FFT 방식의 경우는 $N\log_2 N$의 연산만 필요로 한다. N-point DFT의 식을 DIF(Decimation-in-Frequency) FFT의 형태로 변형하는 과정을 식 (1)에 보

였다[1,3].

$$\begin{aligned} X[k] &= \sum_{n=0}^{N-1} x[n] W_N^{nk}, \qquad k=0,1,\ldots,N-1, \\ &= \sum_{n=0}^{\frac{N}{2}-1} x[n] W_N^{nk} + \sum_{n=\frac{N}{2}}^{N-1} x[n] W_N^{nk} \\ &= \sum_{n=0}^{\frac{N}{2}-1} x[n] W_N^{nk} + \sum_{n=0}^{\frac{N}{2}-1} x[\frac{N}{2}+n] W_N^{k(\frac{N}{2}+n)} \\ &= \sum_{n=0}^{\frac{N}{2}-1} x[n] W_N^{nk} + W_N^{k\frac{N}{2}} \sum_{n=0}^{\frac{N}{2}-1} x[\frac{N}{2}+n] W_N^{nk} \end{aligned} \tag{1}$$

여기서 $W_N = e^{-j(2\Pi/N)}$ 이므로 $W_N^{k\frac{N}{2}} = e^{j(2\Pi/N)\cdot\frac{N}{2}} = (-1)^k$ 로 나타낼 수 있다.

$$X[k] = \sum_{n=0}^{\frac{N}{2}-1} x[n] W_N^{nk} + (-1)^k \sum_{n=0}^{\frac{N}{2}-1} x[\frac{N}{2}+n] W_N^{nk} \tag{2}$$

식(2)에서 먼저 k가 짝수인 경우는 k=2r, 홀수인 경우 k=2r+1로 대입하면 다음과 같다.

$$\begin{aligned} X[2r] &= \sum_{n=0}^{\frac{N}{2}-1} x[n] W_N^{2rn} + \sum_{n=\frac{N}{2}}^{N-1} x[\frac{N}{2}+n] W_N^{2rn} \\ &= \sum_{n=0}^{\frac{N}{2}-1} (x[n] + x[\frac{N}{2}+n]) W_N^{2rn} \\ &= \sum_{n=0}^{\frac{N}{2}-1} (x[n] + x[\frac{N}{2}+n]) W_{\frac{N}{2}}^{rn} \end{aligned} \tag{3}$$

$$\begin{aligned} X[2r+1] &= \sum_{n=0}^{\frac{N}{2}-1} x[n] W_N^{2rn} - \sum_{n=\frac{N}{2}}^{N-1} x[\frac{N}{2}+n] W_N^{2rn+n} \\ &= \sum_{n=0}^{\frac{N}{2}-1} (x[n] - x[\frac{N}{2}+n]) W_N^{n} W_N^{2rn} \\ &= \sum_{n=0}^{\frac{N}{2}-1} (x[n] - x[\frac{N}{2}+n]) W_N^{n} W_{\frac{N}{2}}^{rn} \end{aligned} \tag{4}$$

식(3)과 식(4)의 경우를 묶어서 간단히 정리하면 FFT 알고리즘은 다음과 같이 나타낼 수 있다.

$$\begin{aligned} X[k] &= \sum_{n=0}^{\frac{N}{2}-1} (x[n] + x[\frac{N}{2}+n]) W_{\frac{N}{2}}^{rn} \qquad k=2r \text{ 일때} \\ &= \sum_{n=0}^{\frac{N}{2}-1} (x[n] - x[\frac{N}{2}+n]) W_N^{n} W_{\frac{N}{2}}^{rn} \quad k=2r+1 \text{일때} \end{aligned} \tag{5}$$

III. FFT/IFFT 프로세서의 구조

Radix-2 DIF FFT 알고리즘을 이용한 FFT/IFFT 프로세서를 구성하면 그림 1과 같이 나타낼 수 있다. FFT/IFFT 프로세서는 지연기, 파이프라인된 버터플라이, 제어신호, Bit-reverse된 순서로 재배열하여 출력하는 램 블록(RAM block)등 4개의 모듈로 구성된다[5].

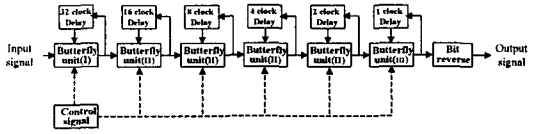

그림 1. IFFT 프로세서의 블록다이어그램

3.1 Butterfly unit(I)의 블록 구조

Butterfly unit(I)은 그림 2와 같이 Upper와 Lower로 구성되어 있으며 multiplier는 사용하지 않으므로 처리 속도의 지연이 적으며 H/W 용량을 줄일 수 있다.

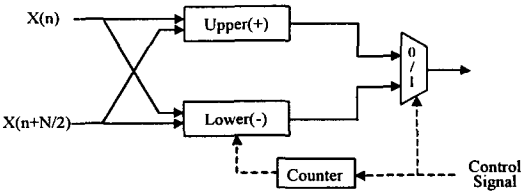

그림 2. Butterfly unit(I)

Upper에서는 $X_m[p] = X_{m-1}[p] + X_{m-1}[q]$에 해당하는 연산을 수행한다. 그림 3에서 보여지는 알고리즘과 같이 Upper에서는 $X_{m-1}[p]$과 $X_{m-1}[q]$의 값을 읽어들여서 각각의 실수값과 허수값들을 비교한다. 그리고 각각의 실수값을 비교해서 값이 "11"이면 출력되는 다음 단계의 $Real(X_m[p])$은 6bit의 twiddle factor에 해당하는 31의 두 배의 값을 가지게 되고 "00"이면 -31의 두배의 값을 가지며 나머지 경우는 0이 된다. 그리고 허수값을 각각 비교해서 "11"이면 다음단계의 $Img(X_m[p])$는 31의 두 배의 값을 가지고 "00"일 경우에는 -30의 두 배의 값을, 나머지 경우는 0의 값을 가진다.

Lower는 $X_m[q] = (X_{m-1}[p] - X_{m-1}[q]) W_N^r$ 에 해당하는 연산을 수행하게 된다. 여기서 $Real(X_{m-1}[p]) - Real(X_{m-1}[q])$를 A로 $Img(X_{m-1}[p]) - Img(X_{m-1}[q])$를 B로 치환하면 $X_m[q]$의 값은 식(6)과 같이 나타낼 수 있다.

$$\begin{aligned} Real(X_m[q]) &= A \cdot Wr - B \cdot Wi \\ Img(X_m[q]) &= A \cdot Wi + B \cdot Wr \end{aligned} \tag{6}$$

Lower에는 $X_{m-1}[p]$과 $X_{m-1}[q]$의 값이 각각 입력되며 twiddle factor의 값은 Control signal 1에 의해서 0~31까지 순서대로 제어된다. 각각의 입력된 실수값을 비교하여 "1,0"이면 $A \cdot Wr = Wr \times 2$, $A \cdot Wi = Wi \times 2$, "0,1"이면 $A \cdot Wr = (-Wr) \times 2$, $A \cdot Wi = (-Wi) \times 2$의 값을 가진다. 그리고 허수값도 순서도에 따라서 구할 수가 있다.

다음 단계의 $Real(X_m[q])$은 $A \cdot Wr$과 $B \cdot Wi$의 차로 $Img(X_m[q])$은 $A \cdot Wi$과 $B \cdot Wr$의 합으로 나타낼 수 있으며 여기서 Wr(weight_real)는 Twiddle factor의 실수값을, Wi(weight_img)는 허수값을 나타낸다.

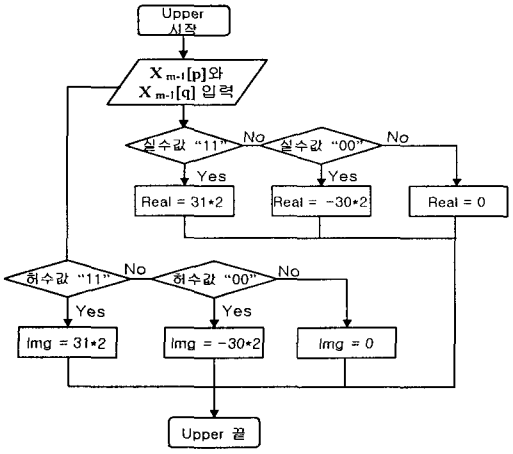

그림 3. Upper의 순서도

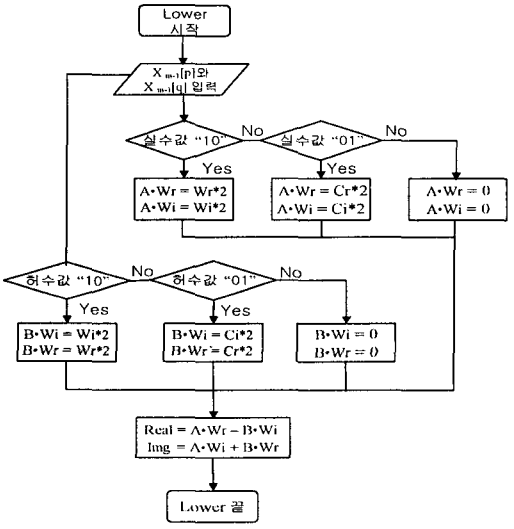

그림 4. Lower의 순서도

Upper과 Lower은 각각 독립적으로 연산을 계속하며 결과값은 N/2 마다 스위칭 되는 MUX신호에 의해 하나의 path로 출력된다.

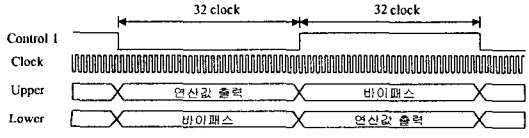

그림 5. 제어신호에 의한 Butterfly unit(I)의 출력

3.2 Butterfly unit(Ⅱ)의 블록 구조

Butterfly unit(Ⅱ)는 그림 6과 같이 X(n)과 X(n+N/2)의 입력을 받아서 Butterfly를 취한 후 제어신호에 의해 서로 다른 twiddle factor를 적절하게 곱해준다. 연산 후 나오는 결과값은 각 단계의 제어 신호에 의해 MUX되어 하나의 path로 출력된다.

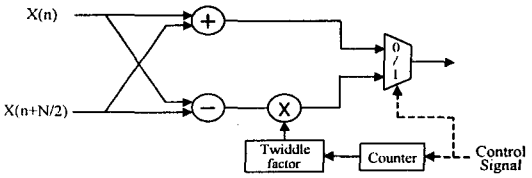

그림 6. Butterfly unit(Ⅱ)

3.3 Butterfly unit(Ⅲ)의 블록 구조

마지막으로 여섯 번째 단계의 해당하는 Butterfly unit(Ⅲ)는 단지 W0와 W16만이 곱하여 진다. 이때 W0는 1이므로 twiddle factor를 곱해줄 필요가 없고 W16은 $W^{16} = e^{-j2\pi \cdot 16/64} = -j$ 임을 알 수 있다. 즉 FFT일 경우는 $-j$, IFFT일 경우는 j를 곱한다. $(a+jb)\times(-j) = b-ja$, $(a+jb)\times j = -b+ja$ 이므로 FFT에서는 입력되는 값에 실수부와 허수부의 값을 치환하고 허수부의 값을 바꾸어준다. IFFT에서는 실수부와 허수부의 값을 치환하고 실수부의 값을 바꾸어주며 결과값은 제어 신호에 의해 MUX되어 하나의 path로 출력된다[2].

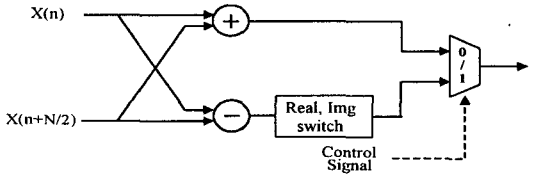

그림 7. Butterfly unit(Ⅲ)

3.4 Butterfly unit(Ⅳ)의 블록 구조

DIF(Decimation In Frequency)FFT 알고리즘에서 순서대로 입력을 받은 프로세서는 bit-reverse의 형태로 출력된다. 그러므로 IFFT/FFT의 경우 마지막 단계에서 나오는 데이터를 정상적인 Address(7bit)로 램에 저장한 후 램에서 bit-reserve형태의 Address로 데이터를 출력한다. 이 때 2개의 램을 이용하여 RAM1에서 64개의 데이터를 출력하는 동안 RAM2에서는 입력되는 64개의 데이터를 램에 저장한다. 이 때 write_enable 신호는 제어신호에 의해 64clock을 주기로 반전된다.

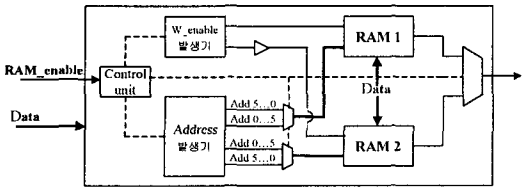

그림 8. Bit-reverse 출력을 내는 RAM 블록도

IV. 시뮬레이션을 통한 성능 평가

본 논문에서는 위에서 논의한 구조를 VHDL로 기술하였으며 RAM_1과 RAM_2는 ALTERA에서 제공하는 LPM(library of parameterized modules)을 이용했다[4].

랜덤한 시퀀스(1,-1)를 만들어서 C언어로 시뮬레이션 한 후 이 시퀀스를 ALTERA의 MAX+plus2에 불러들여 나오는 결과와 C프로그램의 결과를 비교하여 성능을 확인하였다. 그림 9와 그림 10은 C언어로 시뮬레이션 한 결과와 실제 VHDL로 IFFT를 작성하여 나오는 결과값을 비교한 것이다. 양자화 에러에 따른 약간의 차이를 제외하면 신호가 예측한 결과와 거의 같음을 알 수 있다. IFFT에서는 레지스터와 연산에 의해 138클럭의 지연과 RAM에 의한 64클럭의 지연으로 총 202클럭(40ns*202=8.08us)의 지연이 발생하며 FFT에서는 140클럭과 RAM에 의해 발생하는 64클럭으로 모두 204클럭(40ns*204=8.16us)의 지연이 발생한다. FFT에서는 IFFT와는 달리 1단계에서 Butterfly unit(I)의 논리를 사용할 수 없기 때문에 multiplier에 의한 2클럭의 지연이 더 발생한다. 그러므로 H/W의 용량에서 IFFT는 ALTERA사의 EPF10K100ARC240-1의 87%의 게이트를 점유하지만 FFT의 경우 EPF10K130ARC240-1 device의 89%를 점유한다. 그림 11은 FFT에 IFFT의 출력시퀀스를 입력받아 FFT연산 후 나온 출력시퀀스를 IFFT에서의 입력시퀀스와 Exclusive OR를 취하여 실제 데이터에 에러가 없음을 확인할 수 있다.

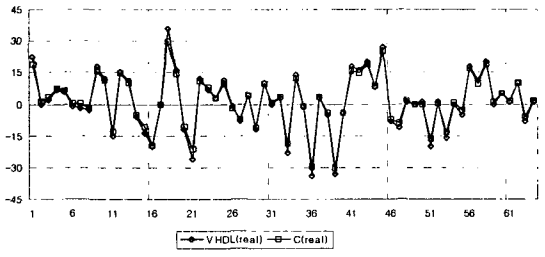

그림 9. C 와 VHDL 시뮬레이션 결과(Real)

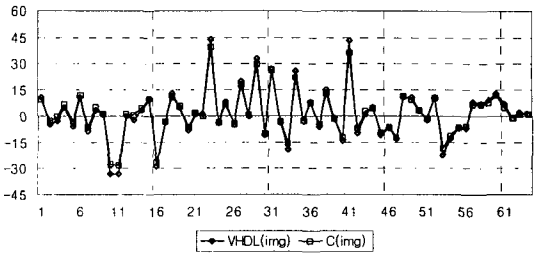

그림 10. C 와 VHDL 시뮬레이션 결과(Img)

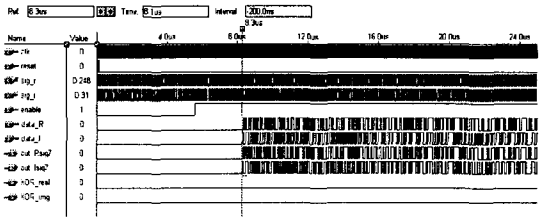

그림 11. IFFT입력 시퀀스와 FFT 출력 시퀀스의 XOR 한 결과

V. 결론

위의 구조를 이용하여 FFT/IFFT Radix-2 64point 프로세서를 설계해 보았다. 입력과 출력되는 데이터는 모두 25MHz(40us)의 클럭을 이용했으며 내부의 제어신호는 기본 클럭을 분주해서 사용했다. 실수와 허수를 각 8bit로, Twiddle factor를 6bit의 양자화 비트로 설계했다. 양자화 비트가 낮을 경우는 각 단계를 거치면서 양자화 에러가 계속 증가하여 심한 오류가 발생하게 된다. 반면 양자화 비트가 크게 되면 용량과 처리시간이 증가하게 되는 단점이 있으므로 적절한 양자화 비트를 고려해야 한다. 구현한 IFFT 프로세서는 입력비트가 1, -1의 경우로 한정지어 설계되었다. 따라서 1 ,0 ,-1 과 같이 비트 입력이 없는 경우를 고려한 IFFT프로세서 구현을 진행중에 있으며 본 논문은 앞으로 IFFT 프로세서 설계의 기초 자료로 활용할 수 있을 것으로 사료된다.

Reference

[1] Alan V. Oppenheim, Ronald W.Schafer "Discrete-time signal processing" pp.646-652

[2] 오정열, 임명섭 "IEEE802.11a 고속무선 LAN에 적용할 FFT/IFFT 프로세서 설계"

[3] Shousheng He, Mats Torkelson "A new approach to pipeline FFT processor"

[4] 박세현 저 "VHDL 기본과 활용" pp.205-221