IEEE 802.11a 무선랜 모뎀에 적용할 FFT 설계 및 구현

정우철*, 송오영*

*중앙대학교 공과대학원 전자전기공학부

e-mail:song@jupiter.cie.cau.ac.kr

Design And Implementation of A Pipeline FFT for IEEE 802.11a Wireless LAN Modem

Woo Chuel Jung*, Ohyoung Song*

*School of Electrical & Electronics Engineering, Chung-Ang University

요 약

본 논문에서는 IEEE 802.11a 무선 랜 모뎀에 적용할 파이프 라인구조의 FFT 설계 및 구현에 대해서 제시한다. 구현된 FFT의 기본 구조는 radix-4 Single Delay Format 이나 제안된 나비 연산기에 의해서 복잡도는 radix-2방식과 동일하며 저전력을 고려해서 구현하였다. 구현된 FFT는 0.35 μm LG 라이브러리를 이용하여 합성되었고 64-포인트 FFT/IFFT를 4 μs에 수행을 하며 16MHz로 동작을 한다.

1. 서론

현재의 무선 통신망은 음성과 저속 데이터 서비스가 주류였으나 무선 멀티미디어 서비스에 대한 요구가 증가함에 따라 IMT-2000, 무선 ATM 통신망이 연구 및 개발되고 있다. 무선 랜은 무선 전송 기술을 사용하여 배선이 필요없고 단말기의 재배치가 용이하고 이동중에도 통신이 가능하며 빠른 시간에 랜을 구축할수 있는 이동성, 휴대성, 간편성 등의 이점으로 인하여 응용분야가 확산되고 있다. 1-2Mbps 전송속도를 갖는 무선 LAN의 표준안인 IEEE 802.11 규격을 향상시켜 1999년 7월 IEEE 802.11 전체회의에서 NII 대역의 5GHz 대에서 6~54Mbps의 전송 속도를 갖는 OFDM(Orthogonal Frequency Division Multiplexing)방식의 IEEE 802.11a 무선 랜 표준안을 발표하였다[1] 전송 속도를 초고속화함으로서 기존의 유선 랜에 대한 대체 수단 또는 백본망이 없이 자유롭게 망을 구성하여 서비스를 제공하는 것을 목표로 한다. OFDM방식은 송수신단에서 FFT(Fast Fourier Transform)/IFFT(Inverse Fast Fourier Transform) 블록으로 구현이 되어지므로 IEEE 802.11a 무선 랜에서도 FFT/IFFT블록은 필수 요소이다[2].

2. 본론

FFT/IFFT 모듈의 VLSI 구현 방법에는 Single processor 방식[3], Pipeline 방식[4], Parallel processor방식[5] 등이 있다. Parallel processor 방식은 FFT 연산에 필요한 모든 나비 연산기를 구현해

서 FFT 연산을 수행하는 방식으로 속도는 빠르나 크기가 너무 크다. Pipeline 방식은 각 단계마다 나비 연산기를 구현해서 FFT를 병렬 처리하게 된다. 또한 구조의 규칙성으로 인하여 VLSI 설계가 용이하다는 장점이 있다.

2.1 구 조

2.1.1 알고리듬

FFT 연산을 하기 위한 첫 번째 단계는 입력 데이터를 두 그룹으로 분할하는 것이다[6]. 지표 n과 k를

$$n = A\,n_1 + n_2, \quad \begin{cases} 0 \le n_1 \le B-1 \\ 0 \le n_2 \le A-1 \end{cases} \qquad (1)$$
$$k = k_1 + B\,k_2, \quad \begin{cases} 0 \le k_1 \le B-1 \\ 0 \le k_2 \le A-1 \end{cases}$$

로서 나타내자. n_1과 n_2는 지정된 범위에서 모든 가능한 값을 가질 때, n은 0에서 $N-1$까지의 모든 가능한 값을 반복됨 없이 취할 수 있다. 이것은 주파수 지표 k에 대해서도 동일하다. 이런 지표 매핑을 사용하여 DFT를 두 지표 k_1과 k_2의 함수로서 표현할 수 있다. 식(1)을

$$X_k = \sum_{n=0}^{N-1} x_n W_N^{kn}, \qquad 0 \le k \le N-1 \quad (2)$$

에 적용해서 정리하면 다음과 같다.

$$X_k = \sum_{n_2=0}^{A-1} W_N^{Bk_2n_2} W_N^{k_1n_2} \sum_{n_1=0}^{B-1} x_{An_1+n_2} W_N^{Ak_1n_1} \quad (3)$$

$$0 \le k_1 \le B-1,\ 0 \le k_2 \le A-1$$

$N=4^c$인 경우를 고려해보자. N이 4의 승수일 경우에는 radix-4 알고리듬을 적용하는 것이 더 효율적이다. 그리고 IEEE 802.11a에서 $N=4^3=64$이다. 식(1)에서 A와 B를 $A=4,\ B=N/4=4^{c-2}$라고 가정하고 $W^{N/4}=-j$라는 것을 이용하면 식(3)은 다음과 같이 된다

$$X_{(N/4)k_2+k_1} = \sum_{n_2=0}^{3} (-j)^{n_2k_2} W^{n_2k_1} \sum_{n_1=0}^{N/4-1} x_{n_1n_2} W^{An_1k_1} \quad (4)$$

$$0 \le k_1 \le N/4-1,\ 0 \le k_2 \le 3$$

식(4)를 매트릭스 형태로 표시하면 다음과 같다.

$$\begin{bmatrix} X_{k_1} \\ X_{(N/4)+k_1} \\ X_{(N/2)+k_1} \\ X_{(3N/4)+k_1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} \cdot \begin{bmatrix} \sum_{n_1=0}^{N/4-1} x_{4n_1} W^{4n_1k_1} \\ W^{k_1} \sum_{n_1=0}^{N/4-1} x_{4n_1+1} W^{4n_1k_1} \\ W^{2k_1} \sum_{n_1=0}^{N/4-1} x_{4n_1+2} W^{4n_1k_1} \\ W^{3k_1} \sum_{n_1=0}^{N/4-1} x_{4n_1+3} W^{4n_1k_1} \end{bmatrix} \qquad (5)$$

$$0 \le k_1 \le N/4-1$$

2.1.2 FFT/IFFT VLSI 설계

Radix-4 Single Delay Format 구조

그림 1. R4SDF N=64인 경우의 블록도

[그림 1]는 $N=64$인 경우 R4SDF의 전체 블록도이다. 각 단계간은 하나의 데이터 패스로 연결되어 있으며 새로운 데이터나 중간 결과 값을 저장하기 위해서 궤환 딜레이 레지스터(Feedback Delay Register,FDR)가 사용된다. Radix-4구조이므로 모두 3개의 FDR 데이터 패스가 형성이 된다.

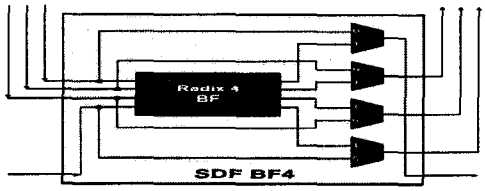

그림 2. R4SDF의 BF4 구조

[그림 2]은 radix-4 SDF의 나비 연산기 구조를 나타낸 것이다. 단계1에서는 식(4)를 만족할 때까지 들어오는 데이터를 FDR로 보내게 된다. 이때는 단계1의 MUX 제어 신호가 0이다. 식(4)를 만족하게 될 때 MUX 제어신호는 1이 되게 되고 모두 4번의 나비연산기 동작을 한다. 이때 하나의 데이터 패스는 단계2로 넘어가고 나머지는 다시 FDR로 저장이 된다. 4번의 나비 연산기 동작이 끝난 후 MUX 제어 신호는 다시 0이 되고 이때 FDR에 저장이 되어 있는 데이터에 격자계수를 곱한 값이 단계2로 넘어간다. 단계 2에서는 단계2의 MUX 제어 신호가 0일 경우 FDR로 값이 저장이 되고 제어 신호가 1인 경우에는 나비 연산을 하게 된다.

나비 연산기 설계

Radix-4 나비연산기는 식(6)과 같다. 모든 데이터는 복소수이다. 식(6)을 VLSI로 구현하기 위해서 일반적으로 16개의 덧셈기/뺄셈기를 필요로 하게 된다. 하지만 제어신호를 첨가하여 다시 적으면 식(7)이 된다.

$$\left.\begin{aligned} Y1 &= X1 + X2 + X3 + X4 \\ Y2 &= X1 - jX2 - X3 + jX4 \\ Y3 &= X1 - X2 + X3 - X4 \\ Y4 &= X1 + jX2 - X3 - jX4 \end{aligned}\right\} \quad (6)$$

$$\begin{aligned} cY1 + \bar{c}Y2 = &[(X1_{re} + cX3_{re} - \bar{c}X3_{re}) + c(X2_{re} + X4_{re}) + \bar{c}(X2_{im} - X4_{im})] \\ &+ j[(X1_{im} + cX3_{im} - \bar{c}X3_{im}) + c(X2_{im} + X4_{im}) - \bar{c}(X2_{re} - X4_{re})] \\ cY3 + \bar{c}Y4 = &[(X1_{re} + cX3_{re} - \bar{c}X3_{re}) - c(X2_{re} + X4_{re}) - \bar{c}(X2_{im} - X4_{im})] \\ &+ j[(X1_{im} + cX3_{im} - \bar{c}X3_{im}) - c(X2_{im} + X4_{im}) + \bar{c}(X2_{re} - X4_{re})] \end{aligned} \quad (7)$$

식(7)을 이용하여 하드웨어로 구현 시에는 8개의 덧셈기/뺄셈기로 구현이 되어진다. radix-4로 구현시 가장 문제점이 나비 연산기가 복잡하다는 것이었다. 그러나 식(7)을 이용하면 전체 구현시 필요한 덧셈기/뺄셈기의 수는 radix-2방식과 동일하다. 식(7)을 이용하면 구현시 일반적인 R4SDF구조보다 절반의 덧셈기와 뺄셈기를 필요로 한다. [그림 3]는 식(7)을 이용한 구현된 radix-4 나비연산기 구조이다.

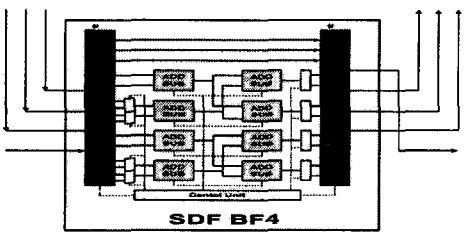

그림 3. 구현된 나비연산기 블록도

2.2 성능 분석 및 합성

2.2.1 Simulation 방법 및 결과

랜덤하게 생성한 데이터를 16-QAM을 한 후 파일롯 톤과 가상 캐리어를 삽입한 64개의 무선 랜의 송신단에서의 실제 입력 신호(24Mbps)를 만들었다. 입력 신호를 구현된 IFFT와 Matlab에서의 시뮬레이션 결과가 [그림 6]과 [그림 7]이다. Matlab 결과와 동일함을 알 수 있다.

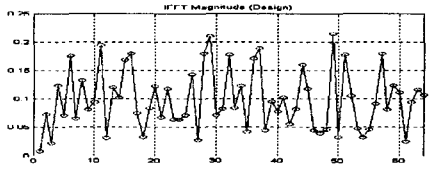

그림 6. 구현된 IFFT의 결과 (크기성분)

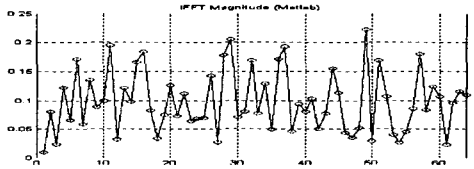

그림 7. Matlab의 결과 (크기성분)

3. 결 론

랜의 표준으로 1999년에 OFDM방식을 채택한 IEEE 802.11a이 확정되었다. 아직은 무선 랜의 시장이 활성화가 되지 않았지만 휴대용 컴퓨터의 보급, 고속 멀티 미디어 서비스에 대한 수요와 무선 랜의 이동성, 휴대성, 간편성 등의 장점으로 인하여 활성화가 될 것으로 예상된다. 본 논문에서는 IEEE 802.11a 무선 랜 모뎀에 적용할 64-포인트 FFT/IFFT를 설계하고 구현하였다. Radix-4 알고리듬의 나비연산기의 복잡도를 줄이기 위해서 본 논문에서 제시한 나비연산기를 사용하여 복잡도를 radix-2 알고리듬과 동일시 했으며 저전력을 고려하여 불필요한 스위칭 횟수를 줄이도록 구현하였다. 본 논문에서는 64-포인트 FFT를 구현했지만 파이프 라인 방식의 규칙성으로 인하여 확장이 가능하여 OFDM 방식을 채택하는 DAB, DVB등에 기본 모듈로서 사용이 가능하다.

참고문헌

[1] IEEE Std 802.11a, "High Speed Physical Layer in the 5 GHz Band", 1999

[2] Richard V. Nee, Ramjee Prasad, "OFDM for Wireless multimedia communications", Artech house publishers, 2000.

[3] Bevan. M. Baas, "A low-power, high-performance, 1024-point FFT processor", IEEE Journal of Solid-State Circuits, pp. 380-387, Mar. 1999.

[4] R. Storn, "Radix-2 FFT-pipeline architecture with reduced noise-to-signal ratio", IEE Proc.-Vis. Image Signal Process., vol. 141, no. 2, pp. 81-86, Apr. 1994.

[5] John E. Whechel, John P. O'Mailey, William J. Rinard, James F McArthur, "The systolic phase rotation FFT - A new algotirhm and parrallel processor architecture", IEEE International Conference on Acoustics, Speech and Signal Processing, vol. 2, pp. 1021-1024, Apr. 1990.

[6] Alan V. Oppenheim and Ronald W. Schafer, "Digital signal processing", Prentice Hall, 1975.