OFDM 시스템을 위한 radix-8/4/2 가변 FFT 프로세서의 설계

김영진*, 김형호**, 이현수***

Design of a Radix-8/4/2 variable FFT processor for OFDM systems

Young-Jin Kim*, Hyung-Ho Kim, Hyon-Soo Lee*****

요 약 본 논문은 OFDM 시스템을 위한 효율적인 가변길이 radix-8/4/2 FFT 구조를 제안하였다. radix-8/4/2 연산을 수행하기 위해서 제안한 FFT 프로세서는 shared memory 구조를 사용하여 하드웨어가 단순하고 적은 면적을 차지한다. 메모리 사이즈를 줄이고 데이터들 간의 충돌을 피하기 위해 효율적인 In-place 메모리 엑세스 방법을 제안한다. 또한 회전인자(twiddle factor)를 위한 ROM 기반의 lookup 테이블 방식을 대신하여 적은 면적을 차지하는 회전인자 발생기를 제안한다. 제안한 FFT 프로세서는 802.11a, 802.16a, DAB, DVB-T/H 그리고 xDSL에서 요구하는 모든 FFT 샘플링 포인트인 64, 256, 512, 1024, 2048, 4096 그리고 8192 포인트의 FFT 연산을 할 수 있다.

주제어 : OFDM, FFT, 회전인자, 공유 메모리, FPGA

Abstract In this paper, we propose an efficient variable-length radix-8/4/2 FFT architecture for OFDM systems. The FFT processor is based on radix-8 FFT algorithm and also supports radix-4 or radix-2 FFT computation. We are using efficient “In-place” memory access method to maintain conflict-free data access and minimize memory size. Also we replace a very large lookup table with a twiddle factor generator which consumes less area then a ROM-based lookup table. The proposed FFT processor performs variable-length FFT including 64, 256, 512, 1024, 2048, 4096 and 8192 points which cover all the required FFT lengths used in 802.11a, 802.16a, DAB, DVB-T, VDSL and ADSL.

Key Words : OFDM, FFT, Twiddle factor, Shared memory, FPGA

1. 서론

오늘날 OFDM(Orthogonal Frequency division multiplexing) 기술은 고속 데이터 전송에 적합한 방법으로, 고속의 디지털 통신 즉, xDSL(x-Digital Subsciber Line), DAB(Digital Audio Broadcasting), DVB(Digital Video Broadcasting)-T/H 그리고 WLAN(Wireless Local Area Network) 등에 폭넓게 사용되고 있다. OFDM은 고속 전송률을 갖는 데이터 열을 낮은 전송률을 갖는 많은 수의 데이터 열로 나누어 PSK(Phase Shift Keying) 이나 QAM(Quadrature Amplitude Modulation)에 의해 변조된 부반송파를 사용하여 병렬로 전송하는 것이다[9].

OFDM 시스템에서 가장 중요한 연산이 DFT (Discrete Fourier Transform)와 IDFT(Inverse Discrete Fourier Transform)이다. 실시간 처리가 중요한 OFDM에서는 DFT/IDFT의 빠른 연산 처리를 위해서 효과적인 FFT 알고리즘이 필요하다. 또한 OFDM의 어플리케이션

*본 논문은 2009.9 ~ 2010.8 교수연구년 과제로 경희대학교에 의하여 지원되었음
*경희대학교 전자계산공학과 박사과정
**세한대학교 컴퓨터교육과 교수
***경희대학교 컴퓨터공학과 교수 (교신저자)
논문접수 : 2013년 1월 24일, 1차 수정을 거쳐, 심사완료 : 2013년 2월 20일, 확정일 : 2013년 2월 20일

(application)마다 FFT의 샘플링 포인트(Sampling point, N) 기준이 다르기 때문에 어플리케이션에 맞는 FFT 프로세서의 설계가 필요하다. 이러한 문제를 해결 하기위해서 병렬구조 (Parallel architecture)[3], 파이프라인구조 (Pipeline architecture)[12], 메모리 공유 구조(Shared memory architecture)[13]등이 연구되었다.

병렬 구조는 FFT 연산 스테이지를 병렬로 처리하는 구조로써 연산 속도는 빠르지만 샘플링 포인트가 증가할수록 하드웨어 복잡도가 증가하는 단점을 갖는다. 파이프라인 구조는 FFT연산 스테이지 마다 단일의 버터플라이 연산기를 두어 파이프라인 처리함으로써 연산 속도를 빠르게 할 수 있지만 샘플링 포인트 수가 증가할수록 FFT 연산 스테이지 수가 증가함으로 연산기가 추가되어야 하는 단점을 가진다. 반면, 공유 메모리 구조는 연산 결과를 저장하는 메모리와 단일의 버터 플라이 구조로 되어있으므로, 샘플링 포인트가 증가하여도 버터플라이 연산기를 추가 할 필요가 없어 병렬처리나 파이프라인 구조보다는 연산기의 면적은 효율적이지만, 단일의 버터플라이 연산기를 반복적으로 사용하므로 연산횟수가 많아 낮은 처리율을 갖으며, 데이터를 읽고 쓰기위한 메모리의 주소를 제어하기 위한 추가적인 하드웨어가 필요하다. 또한 낮은 처리율의 단점을 해결하기 위해서 높은 radix를 사용하면, 메모리 구조가 복잡해지며, FFT 길이(N)의 제한을 받는다.

본 논문에서는 앞서 언급된 문제들을 해결하기 위해서 첫 번째로 OFDM의 실시간 처리를 위한 다양한 샘플링 포인트에 적용 가능한 FFT 프로세서를 설계하였다. 연산의 수를 최대한 줄이고 프로세서의 면적을 줄이기 위해서 공유 메모리 구조를 사용하였다.

두 번째로 공유 메모리의 낮은 처리율을 보완하기 위해서 radix-8 DIF(Decimation-in-Frequency) FFT 알고리즘을 적용하였다. 그러나 radix-8 DIF FFT는 샘플링 포인트가 8n이 아닌 포인트에 대해서는 처리할 수 없다는 문제점이 있다. 그러므로 2n과 4n의 샘플링 포인트에서도 연산을 수행할 수 있도록 radix-8 DIF FFT상에 radix-2와 radix-4 DIF FFT가 가능한 radix-8/4/2 DIF 버터플라이 연산기를 제안하였다. 세 번째로 다양한 radix를 연산하기 위해서는 메모리 구조가 복잡해지므로 효율적인 메모리 엑세스 방법에 대해서 제안하였으며, 마지막으로 FFT 연산을 위한 회전인자(twiddle factor)의 면적을 줄이기 위해서 Rom 기반의 룩업테이블 방식이 아니라 회전인자를 계산하여 생성하는 방식을 제안하였다.

본 논문의 구성은 2장에서 관련연구로 OFDM과 FFT 알고리즘에 대해서 알아보고, 3장에서는 radix-8/4/2 FFT 구조에 대해서 제안하였으며, 4에서는 구현방법과 기존 구조와의 성능 비교를 하였다. 마지막으로 5장에서는 논문의 결론을 맺었다.

2. 관련기술

2.1 OFDM 기술

OFDM 신호는 PSK나 QAM에 의해 변조된 부반송파의 합으로 구성된다. $N=2^n$의 부반송파를 갖는OFDM 신호의 주파수 영역에서 k번째 부반송파에 실린 복소 데이터를 A_k라고 정의할 때 입력 심볼 시퀀스는 $A=[A_0 A_1 \cdots A_{N-1}]$과 같이 표현된다. OFDM 신호는 $1/NT$의 주파수 만큼 간격을 갖는 N개의 변조된 부반송파를 더하여 전송된다. t가 연속된 시간을 나타낼 때 시간 영역에서는 OFDM 신호는 수식(1)과 같이 표현된다.

$$a_t = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} A_n e^{j2\pi \frac{n}{NT} t}, \quad 0 \leq t \leq NT \tag{1}$$

OFDM 방식에서는 상호 직교성을 갖는 복수의 반송파를 사용하므로 주파수 이용률이 높아진다. 또한 OFDM을 구현하는데 있어서 송·수신부에서 복수의 반송파를 변·복조하는 과정은 IDFT와 DFT를 수행한 것과 같아서 IFFT와 FFT를 사용하면 고속으로 처리 가능한 OFDM을 구현할 수 있다.

2.2 FFT 알고리즘

FFT는 통신, 신호처리, 이미지 처리 등 매우 다양한 곳에 쓰이는 알고리즘이다. 시간 도메인의 신호를 진폭과 주파수의 스펙트럼을 분석할 수 있도록 주파수 도메인으로 변환하는 방법이다. N포인트 DFT를 계산하는데, N포인트 DFT의 주기성과 대칭성을 사용하여 연속된 N보다 작은 DFT로 나누어 계산할 수 있는데 $N/2$포인트로 나누어 계산하는 방법이 FFT 알고리즘이다.

2.2.1 DFT 알고리즘

$x(n)$의 N 포인트 DFT는 식(2)와 같이 정의할 수 있다.

$$X[k] = \sum_{n=0}^{N-1} x(n) w_N^{nk}, \quad k = 0, 1, \ldots, N-1 \tag{2}$$
$$W_N^{nk} = e^{-j2\pi/N \cdot nk} = \cos\left(\frac{2\pi}{N} \cdot nk\right) - j\sin\left(\frac{2\pi}{N} \cdot nk\right)$$

식(2)는 w_N^{nk}와 입력 $x(n)$의 곱으로 정의할 수 있다. DFT를 계산하기 위해서는 N^2 번의 곱셈과 $N(N-1)$번의 덧셈이 필요하다. 따라서 N의 크기가 커질수록 계산량이 급격하게 많이 증가한다. 그러므로 본 논문에서는 이를 해결하기 위한 방법으로 radix-8 FFT를 사용하였다.

2.2.2 DIF FFT 알고리즘

DIF FFT 알고리즘은 입력 신호 $x[n]$ 대신에 출력 신호 $X[k]$를 부분샘플로 연속해서 분리하는 방법으로 계산량을 줄일 수 있다. DIF는 신호의 크기가 $N = 2^n$을 만족한다고 가정한다. DFT의 정의인 식(2)로부터 두 부분으로 나누어 정리하면, 식(3)과 같이 변형 될 수 있다.

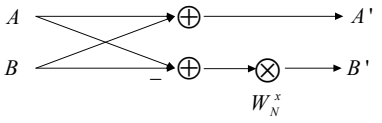
$$X[k] = \sum_{n=0}^{(N/2)-1} \left[x(n) + (-1)^k \times \left(n + \frac{N}{2}\right) \right] W_N^{kn} \tag{3}$$

여기서 $k = 0, 1, \ldots, N$ 이다.

식(3)을 이용하여 짝수 샘플과 홀수 샘플에 대해 radix-2 FFT에 대해서 정의 하면 식(4)와 같다.

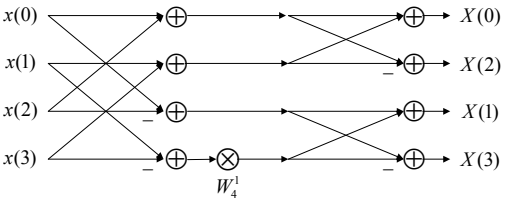
$$X(2k) = \sum_{n=0}^{(N/2)-1} g_1(n) W_{n/2}^{kn} \tag{4}$$
$$X(2k+1) = \sum_{n=0}^{(N/2)-1} g_2(n) W_{n/2}^{kn}$$

식(4)의 radix-2 버터플라이 구조는 [그림 1]과 같다.



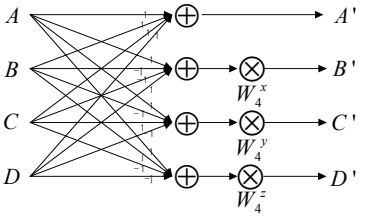
[그림 1] Radix-2 버터플라이 구조

radix-2 버터플라이 구조를 사용하여 N이 2^n인 모든 식을 연산 할 수 있다. [그림 2]는 radix-2 버터플라이 구조를 사용하여 4-point FFT를 실행하는 데이터 플로우를 나타내고 있다.



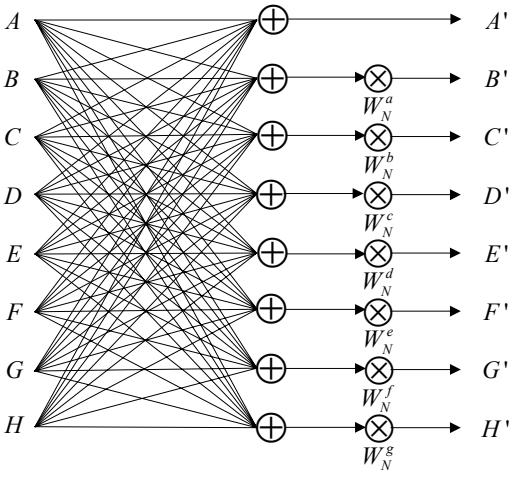
[그림 2] 4포인트 radix-2 FFT의 데이터 플로우

이와 같이 N포인트 FFT는 radix-2 버터플라이 구조로 분해하여 계산할 수 있다. 그러나 radix-2버터플라이 구조를 사용하면 스테이지 수가 증가하게 된다. 스테이지 수를 줄여 연산속도를 빠르게 하기 위한 방법으로 N을 4^n으로 분해하여 처리할 수 있는 방법이 radix-4 버터플라이 구조이다. [그림 3]은 radix-4 버터플라이 구조를 보여준다.



[그림 3] Radix-4 버터플라이 구조

[그림 4]는 N을 8^n으로 분해한 radix-8 버터플라이구조를 보여준다. radix-4보다 스테이지수가 감소함으로써 연산속도가 빠르다.



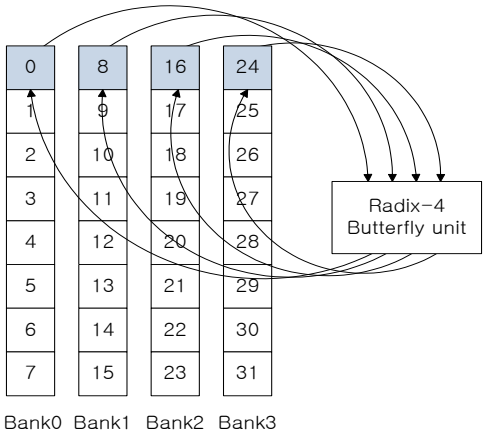
[그림 4] Radix-8 버터플라이 구조

이와 같이 radix-r의 r이 커짐에 따라 연산속도는 빨라진다. 반면 연산복잡도가 증가하며 연산이 불가능한 N 포인트가 존재한다.

본 논문에서는 연산스테이지 수를 줄이고 연산 속도를 증가시키기 위해서 radix-8 DIF FFT를 사용하였으며, 모든 N포인트 FFT 연산을 지원하기 위해서radix-8 뿐만 아니라 radix-4와 radix-2도 지원 가능한 버터플라이 구조를 제안하였다.

2.2.3 공유 메모리 FFT 구조

공유 메모리를 사용하는 FFT 프로세서의 구조는 radix-r에 따라 메모리상에서 r개의 데이터를 불러와 연산한 후 메모리에 r개의 결과값을 메모리에 저장하여야 한다. 이때 r개의 데이터를 동시에 불러와 연산처리 하여야 하므로 메모리를 뱅크(Bank)로 분할하여 처리하고, 연산 결과를 불러온 메모리 주소에 저장하는 In-Place 방법을 사용하여 메모리 크기를 줄일 수 있다[7]. [그림 5]는 radix-4 FFT 프로세서의 경우를 보여주고 있다. 4개의 데이터를 불러와서 FFT 연산을 수행한 후 같은 주소에 저장되는 In-Place 메모리 어드레싱의 예이다.

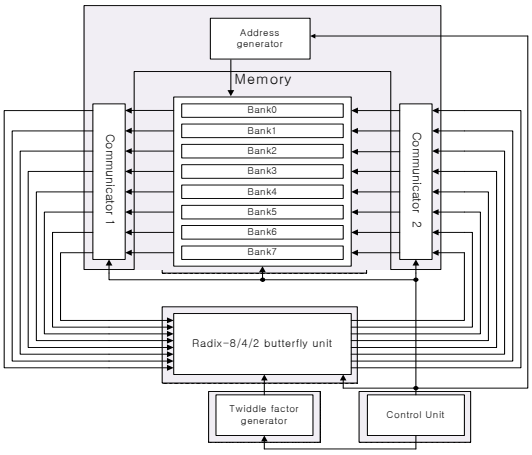


[그림 5] FFT 상에서 In-Place 메모리 어드레싱 구조

In-Place 메모리 어드레싱에서는 동시에 r개의 데이터들을 처리하여야 하므로 메모리간의 충돌을 피하는 것이 중요하여 메모리간의 충돌을 피하는 여러 가지 연구들이 있었다[2][4][6][7][14][15]. 이 방법들의 메모리 할당 방식은 모든 출력을 내보낸 후 새로운 입력을 받을 수 있어 입출력에 시간이 오래 걸린다는 단점을 가지고 있다. Chang[10]이 제안한 방법은 데이터의 입력과 출력에 대한 동작을 overlapping할 수 있어 보다 높은 처리율을 얻을 수 있다. 그러나 이 방법은 r-뱅크 메모리 상에서 고정된 radix-r FFT 알고리즘만을 구현할 수 있다. 그러므로, 본 논문에서는 Chang의 방법을 수정하여 r-뱅크 상에서 radix-8과 radix-4, 그리고 radix-2 FFT가 모두 가능하도록 구현하였다.

3. 제안한 FFT radix 가변구조

제안한 FFT 구조는 크게 3부분으로 구성되어 있다. 첫 번째로 데이터를 저장할 메모리로써 제안한 In-pace 어드레싱을 구현한 부분이며, 두 번째로 가변길이 FFT 연산을 위한 radix-8/4/2 버터플라이 유닛, 마지막으로 회전인자 발생기와 제어 유닛이다. [그림 6]은 제안한 FFT 구조를 나타낸다.



[그림 6] 가변 FFT 프로세서 구조

3.1 제안한 메모리 어드레싱 구조

3.1.1 In-Place 메모리 어드레싱 구조

기존 In-Place 메모리 어드레싱 구조는 고정된 radix-N에 맞추어 구현되어 있다 [10], 본 논문에서는 radix-8/4/2 연산을 모두 수행할 수 있도록 가변 구조로 구현하였다. 메모리 뱅크는 8개로 구성하며 하나의 클럭에 읽고/쓰기가 동시에 가능하다.

처음에는 입력 데이터가 뱅크 0에 순차적으로 저장된

다. 식(5)은 뱅크의 인덱스와 뱅크 안에 할당된 주소를 보여준다.

$$\begin{aligned} bank_address &= (I) \bmod (N/8) \\ bank_index &= \lfloor I \times 8/N \rfloor \end{aligned} \tag{5}$$

여기서 I는 입력데이터의 인덱스이고 0부터 N-1까지의 정수 값을 나타낸다.

첫 번째 연산 스테이지에서 메모리 뱅크의 주소가 0인 8개의 데이터를 불러와 버터플라이 연산기에서 연산한 후 동일한 뱅크의 주소에 데이터를 저장한다. 첫 연산 스테이지에서는 0부터 $N/8-1$까지 8개의 뱅크의 메모리가 동일하게 생성된다. 또한 뱅크의 read 주소와 write 주소는 동일하지만 다음 스테이지에서 불러올 데이터가 하나의 뱅크에 존재하여 충돌하는 것을 피하기 위해서 8개의 데이터를 각각 어떤 뱅크로 할당하는지를 결정해야 한다. 샘플링 포인트 N에 의해 마지막 연산 스테이지에서 radix-8, radix-4 또는 radix-2 FFT 연산을 하는지에 따라서 할당방법이 다르다.

먼저 마지막 연산 스테이지에서 radix-8 FFT 연산을 하는 경우 연산에 필요한 8개 데이터들의 인덱스간의 차가 $a \cdot 8^{n-stage}$이다. 여기서 a는 0부터 7까지의 정수이고 8^n은 샘플링 포인트인 N이다. stage는 연산 스테이지를 나타낸다. 예를 들어 64 포인트 FFT의 경우 첫 번째 연산 스테이지에서 버터플라이 연산기의 입력 데이터들의 그룹은 (0, 8, 16, 24, 32, 40, 48, 56), (1, 9, 17, 25, 33, 41, 49, 57)과 같이 데이터 인덱스들 간의 차가 $a \cdot 8^{2-1}$이다. 마지막 스테이지인 두 번째 연산 스테이지에서 입력 데이터들간의 그룹은 (0, 1, 2, 3, 4, 5, 6, 7), (8, 9, 10, 11, 12, 13, 14, 15)와 같이 데이터들 인덱스 간의 차가 $a \cdot 8^{2-2}$이다. 같은 그룹 안에 속해 있는 데이터들이 같은 메모리 뱅크에 속해 있게 되면 동시에 버터플라이 연산기의 입력으로 들어갈 수 없기 때문에 다른 메모리 뱅크에 속하도록 할당해야 한다. <표 1>은 64포인트 FFT를 위한 뱅크 메모리를 보여준다. 각 스테이지에 맞추어 저장되어 있는 데이터를 보여주고 있다.

마찬가지로 radix-4 FFT 연산과 radix-2 FFT 연산을 마지막 연산 스테이지에서 해야 하는 경우 연산에 필요한 8개의 데이터들의 인덱스간의 차가 각각 $a \cdot 4 \cdot 8^{n'-stage}$ 와 $a \cdot 2 \cdot 8^{n''-stage}$이다($4 \cdot 8^{n'}$과 $2 \cdot 8^{n''}$이 샘플링 포인트 N이다). 이에 해당하는 데이터들의 그룹 역시 같은 메모리 뱅크에 속해 있으면 안 된다.

〈표 1〉 각 스테이지 후 메모리에 저장되어 있는 데이터

a. 입력스테이지

B0	B1	B2	B3	B4	B5	B6	B7
x(0)	x(8)	x(16)	x(24)	x(32)	x(40)	x(48)	x(56)
x(1)	x(9)	x(17)	x(25)	x(33)	x(41)	x(49)	x(57)
x(2)	x(10)	x(18)	x(26)	x(34)	x(42)	x(50)	x(58)
x(3)	x(11)	x(19)	x(27)	x(35)	x(43)	x(51)	x(59)
x(4)	x(12)	x(20)	x(28)	x(36)	x(44)	x(52)	x(60)
x(5)	x(13)	x(21)	x(29)	x(37)	x(45)	x(53)	x(61)
x(6)	x(14)	x(22)	x(30)	x(38)	x(46)	x(54)	x(62)
x(7)	x(15)	x(23)	x(31)	x(39)	x(47)	x(55)	x(63)

b. 스테이지 1

B0	B1	B2	B3	B4	B5	B6	B7
g(0)	g(8)	g(16)	g(24)	g(32)	g(40)	g(48)	g(56)
g(57)	g(1)	g(9)	g(17)	g(25)	g(33)	g(41)	g(49)
g(50)	g(58)	g(2)	g(10)	g(18)	g(26)	g(34)	g(42)
g(43)	g(51)	g(59)	g(3)	g(11)	g(19)	g(27)	g(35)
g(36)	g(44)	g(52)	g(60)	g(4)	g(12)	g(20)	g(28)
g(29)	g(37)	g(45)	g(53)	g(61)	g(5)	g(13)	g(21)
g(22)	g(30)	g(38)	g(46)	g(54)	g(62)	g(6)	g(14)
g(15)	g(23)	g(31)	g(39)	g(47)	g(55)	g(63)	g(7)

c. 스테이지 2

B0	B1	B2	B3	B4	B5	B6	B7
y(0)	y(9)	y(18)	y(27)	y(36)	y(45)	y(54)	y(63)
y(7)	y(8)	y(17)	y(26)	y(35)	y(44)	y(53)	y(62)
y(6)	y(15)	y(16)	y(25)	y(34)	y(43)	y(52)	y(61)
y(5)	y(14)	y(23)	y(24)	y(33)	y(42)	y(51)	y(60)
y(4)	y(13)	y(22)	y(31)	y(32)	y(41)	y(50)	y(59)
y(3)	y(12)	y(21)	y(30)	y(39)	y(40)	y(49)	y(58)
y(2)	y(11)	y(20)	y(29)	y(38)	y(47)	y(48)	y(57)
y(1)	y(10)	y(19)	y(28)	y(37)	y(46)	y(55)	y(56)

메모리 뱅크의 주소를 이진화해서 MSB부터 3비트 마다 분할하면 위의 3가지 경우에 따라 마지막으로 분할된 그룹은 각각 3비트, 2비트, 1비트만 남는다. 마지막 분할된 그룹이 2비트 또는 1비트일 경우 3비트로 만들기 위해 LSB부터 0을 채운다. 3비트로 만들어진 그룹들을 모두 더한 만큼 순환 시프트(circular shift)하여 각각의 메모리 뱅크로 분할하면 데이터들의 충돌을 피할 수 있다. 식 (6)은 메모리 할당 방법을 식으로 표현한 것이다.

$$\begin{aligned} n &= \log_2 N \\ bank_address &:= [a_{n-4}a_{n-5}\ldots a_1 a_0]_2 \\ bank_index &= (i + \sum_{t=0}^{\lceil (n-3)/3 \rceil - 1} (2^2 a_{n-4-3t} + \\ &\quad 2a_{n-5-3t} + a_{n-6-3t})) \bmod 8 \\ &\quad (\text{if } n-j-3t < 0,\ a_{n-j-3t} = 0) \end{aligned} \tag{6}$$

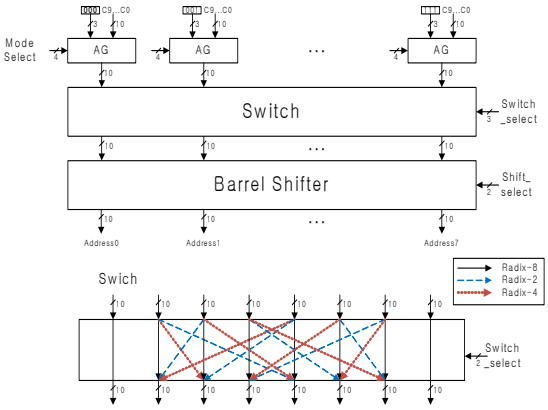
〈표 2〉 메모리 어드레싱

Mode	Address	64	512	4096	128	1024	8192	256	2048
1	$[c_{9...}c_0]_2$	Input, 1stage	Input, 1stage	Input, 1stage	Input, 1stage	Input, 1stage	Input, 1stage	Input, 1stage	Input, 1stage
2	$[n_2n_1n_0c_{9...}c_3]_2$						2stage		
3	$[c_9n_2n_1n_0c_{8...}c_3]_2$			2stage					
4	$[c_9c_8n_2n_1n_0c_{7...}c_3]_2$								2stage
5	$[c_{9...}c_7n_2n_1n_0c_{6...}c_3]_2$					2stage	3tage		
6	$[c_{9...}c_6n_2n_1n_0c_{5...}c_3]_2$		2stage	3tage					
7	$[c_{9...}c_5n_2n_1n_0c_4c_3]_2$							2stage	3tage
8	$[c_{9...}c_4n_2n_1n_0c_3]_2$				2stage	3tage	4tage		
9	$[c_{9...}c_3n_2n_1n_0]_2$	Last stage	Last stage	Last stage	Last stage	Last stage	Last stage	Last stage	Last stage

여기서 i는 버터플라이 연산기에서 나온 결과 값의 인덱스이다. 이와 같은 방법으로 메모리를 제어하기 위해서 메모리의 주소 발생기와 커뮤니케이터가 필요하다.

3.1.2 메모리 주소 발생기

제안한 메모리 어드레싱하는 방법은 N 포인트와 각 스테이지마다 다르고 9가지의 모드로 나눌 수 있다. <표 2>는 N포인트에 따라 각 스테이지가 어떤 모드로 동작하는지를 보여준다. <표 2>에서 c_n는 버터플라이 연산기 카운터의 n번째 비트 숫자이다. 버터플라이 연산 카운터(butterfly counter)는 각 스테이지에서 버터플라이 연산의 수를 세고 각 스테이지가 끝날 때 마다 0으로 리셋(reset)된다. $n_2n_1n_0$은 메모리 뱅크의 인덱스로 000부터 111까지 2진수이다.



[그림 7] 제안한 주소 발생기의 구조

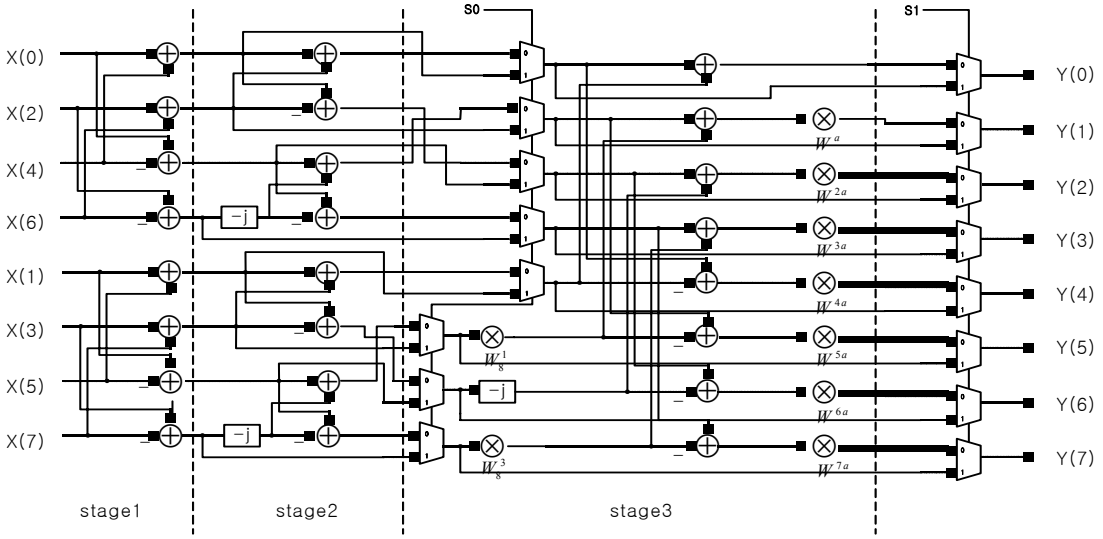
제안한 메모리 주소 발생기는 [그림 7] 과 같다. <표 2>와 같이 9가지의 모드중 하나를 선택하여 각 어드레스 발생기에서 8개의 어드레스가 생성이 된다. 8개의 어드레스는 스위치(switch)와 배럴 시프터(barrel shifter)에 의해 위치가 바뀐다. 스위치는 radix-r에 따라서 출력되는 값의 위치를 결정하며, 배럴 시프터는 버터플라이 연산 카운터의 하위 3비트인 $[c_2c_1c_0]_2$만큼 순환 시프트되어 최종적인 각 뱅크의 어드레스가 생성된다.

3.1.3 커뮤니케이터(Communicator)

제안한 메모리 구조에서 메모리 뱅크의 데이터를 인덱스 순서에 맞추어 읽고 쓰기 위해서 두 개의 커뮤니케이터가 존재한다. 커뮤니케이터는 각각 8x1 MUX 8개로 구성된 배럴쉬프터이다. N 포인트의 각 스테이지에 따라서 시프트의 순서가 바뀌며, 각 MUX의 선택신호가 <표 3>과 같다. 여기서 C_n은 버터플라이 연산 카운터의 n번째 비트 숫자이다.

〈표 3〉 커뮤니케이터 제어 신호

	1 stage	Other stages	Last stage
Communicator 1	000	$c_2c_1c_0$	$c_2c_1c_0$
Communicator 2	64, 512, 4K : $c_8c_7c_6+c_5c_4c_3+c_2c_1c_0$ 128, 1K, 8K : $c_9c_8c_7+c_6c_5c_4+c_3c_2c_1+c_00$ 256, 2K : $c_7c_6c_5+c_4c_3c_2+c_1c_00$	$c_2c_1c_0$	000



[그림 8] 제안한 radix-8/4/2 버터플라이 구조

3.2 Radix-8/4/2 버터플라이 구조

[그림 8]은 제안한 radix-8/4/2 버터플라이 구조를 보여주고 있다. 기본적으로 radix-8 DIF FFT를 위한 버터플라이 구조이다. radix-8을 위해서는 모든 스테이지를 전부 연산하면 된다. 그리고, radix-4와 radix-2를 연산하기 위해서는 MUX의 바이패싱(bypassing)을 사용하여 연산할 수 있다. MUX의 선택신호 S0와 S1을 전부 0값을 선택하면 radix-8 연산이 실행되며, 선택신호를 각각 S0는 0, S1은 1을 선택하면 두 개의 radix-4 연산이 실행되고, 선택신호를 전부 1로 선택하면 네 개의 radix-2 연산을 병렬로 실행할 수 있다.

3.3 회전인자(twiddle factor) 발생기

본 논문에서 회전인자 발생기는 recursive feedback difference 방정식을 사용하였다. 이 방법은 복잡도가 낮은 장점을 갖는다[5]. 식 (7)은 사인함수와 코사인 함수를 발생시키는 recursive함수이다.

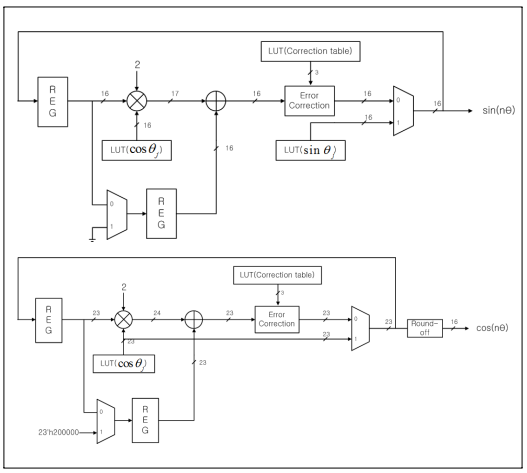
$$\begin{aligned}\sin n\theta &= 2\cos\theta\times\sin(n-1)\theta-\sin(n-2)\theta \\ \cos n\theta &= 2\cos\theta\times\cos(n-1)\theta-\cos(n-2)\theta\end{aligned} \tag{7}$$

식(7)은 누적 오차가 발생하는데 n 비트의 정밀도를 갖도록 설계한다면, 식(8)과 같은 오차가 발생한다.

$$-\frac{3}{2^n}\le error\le+\frac{3}{2^n} \tag{8}$$

오차를 줄이기 위해서 식(9)와 같은 방법으로 정확한 값과 계산된 값의 하위 3비트의 차를 계산한 후, 오차 만큼을 하위 3비트에 더함으로써 오차를 줄이도록 수정하였다.

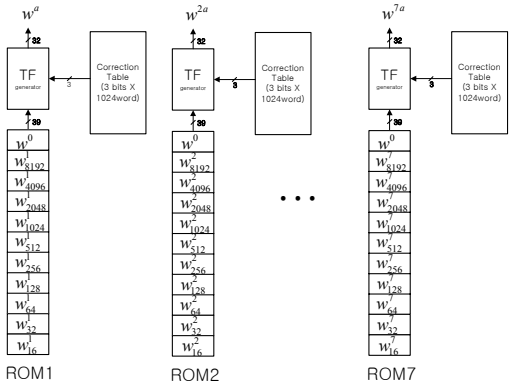
$$\begin{aligned}[z_2z_1z_0]_2 &= unsigned([x_2x_1x_0]_2-[y_2y_1y_0]_2) \\ Correct\ value &= signed(computed\ value+[z_2z_1z_0]_2)\end{aligned} \tag{9}$$



[그림 9] Twiddle factor 발생기의 구조

[그림 9]와 같이 사인값과 코사인 값을 계산하였다. 23비트의 데이터로 이루어져있으며, 2비트의 정수와 21비트의 fraction으로 구성되어 있다. 코사인함수를 구별하기 위해서 위와 같은 비트수를 사용하였다.

본 논문에서는 radix-8 FFT 연산을 실행하여야 함으로 회전인자 발생기, 초기값을 위한 롬, 그리고 오차를 수정하기 위한 테이블, 각각 8개씩이 필요하다. [그림 10]은 제안한 radix-8 FFT를 위한 회전인자 발생기이다.



[그림 10] 제안한 FFT 상에서의 twiddle factor 발생기

4. 구현 및 시뮬레이션

4.1 구현

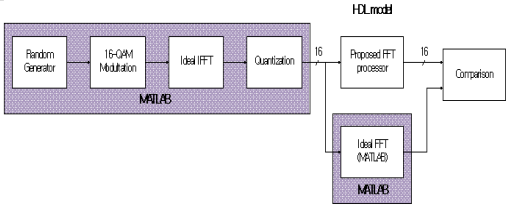
본 논문에서 제안한 FFT를 구현하기 위해서 Verilog-HDL 사용하여 구현하였으며, Xilinx ISE 소프트웨어를 통해 Vertex-4(xc4vlx60) 디바이스에 맞추어 구현하였다. 그리고, Modelsim과 Matlab을 사용하여 검증하였다[16]. 데이터는 실수와 허수부 각각 16비트로 5비트의 정수와 11비트의 소수부를 갖는 고정소수점 데이터이다. <표 4>는 Vertex-4 FPGA의 합성결과이다. 최대 주파수는 42.076MHz 이다[17].

〈표 4〉 Vertex-4 합성결과

Input data width	Twiddle factor width	Slices	Block RAMs	DSP48s	Max. Freq. (MHz)
16	16	3969	30	62	42.076

4.2 시뮬레이션

본 논문에서 구현한 하드웨어를 검증하기 위해서 [그림 11]과 같이 소프트웨어로 구현한 FFT 연산과 하드웨어로 구현한 FFT연산을 비교하였다. 입력 데이터를 변조(16-QAM)한 후 IFFT와 FFT를 각각 실행한 후 하드웨어의 결과와 소프트웨어의 결과의 SQNR(Signal to Quantization Noise Ratio)를 계산하여 비교한다.

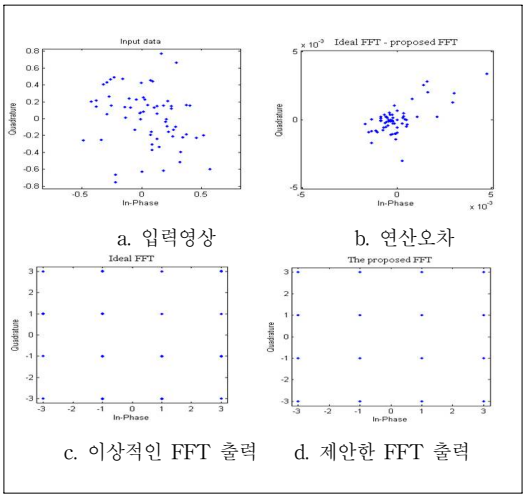


[그림 11] 제안한 FFT의 성능평가 및 설계 검증

식(10)은 SQNR에 대한 수식을 나타내며, A는 MATLAB을 이용한 부동소수점의 결과이며, B는 제안한 FFT 프로세서의 결과이다.

$$SQNR = \frac{\sum(Re(A))^2 + \sum(Im(A))^2}{\sum(Re(A)-Re(B))^2 + \sum(Im(A)-Im(B))^2} \quad (10)$$

예를 들어 64포인트에 대한 FFT 결과는 [그림 12]와 같다.



[그림 12] 시뮬레이션 결과(64-포인트)

모든 주파수 영역에서 54dB 이상의 성능을 보였으며,

식(10)에 의해 SQNR을 구해보면 66.9 dB이 나온다. <표 5>와 같이 포인트 수가 증가할수록 SQNR을 떨어지는 것을 볼 수 있다.

〈표 5〉 각 포인트별 SQNR

Point	64	128	256	512	1024	2048	4096	8192
SQNR(dB)	66.9	63.2	60.3	57.7	55	51.9	48.1	45.3

radix-8/4/2를 이용한 N 포인트 FFT 연산 싸이클을 계산해 보면 아래 식(11)과 같다. FFT 연산을 위해서는 입력 싸이클, 계산 싸이클, 출력 싸이클이 필요하다.

$$C = N + L + (N - \frac{N}{r_{last}}) \qquad (11)$$
$$L = \frac{N}{8} \times \lceil \log_8 N \rceil + \frac{N}{r_{last}}$$

여기서 r_{last}는 마지막 스테이지에서 연산한 radix-r을 의미한다. L은 지연 싸이클로서 현재 프래임의 마지막 입력부터 다음 프로임의 첫 입력까지의 싸이클이다.

〈표 6〉 연산 싸이클의 비교

	Radix	Clock cycles	2048	8192
[1]	Radix-8	N*[log8N]	8201	40969
[11]	Parallel Radix-4	N/8*(log2N-2)+N/4	3328	15360
[10]	Radix-2	N/2*log2N	11264	53248
[16]	Radix-4/2	(5+N/4)*[1+log4(N/2)]	3102	14371
Proposed	Radix-8/4/2	N/8 * ⌈log8N⌉	1024	5120

<표 6>은 2048 포인트 FFT와 8192 포인트 FFT 연산의 경우에 대해 기존 FFT 프로세서와 제안한 FFT 프로세서의 연산 사이클을 비교한 것이다. 연산 사이클이 가장 많은 Radhouane[10]의 FFT와 비교하였을 때 약 90% 향상되었고 연산 사이클이 가장 적은 Moon[11]의 FFT 보다 약 62% 향상되었다.

〈표 7〉 각 포인트에 따른 지연시간

Point	Latency cycles	Latency time(μs)	OFDM Symbol duration(μs)[8]
64	24	0.57	3.2(IEEE 802.11 a/g/n)
128	112	2.67	3.2(IEEE 802.11 a/g/n)
256	160	3.81	31(DAB)
512	256	6.10	62(ADSL)
1K	1024	24.38	128(VDSL)
2K	1536	36.57	248(DAB)
4K	2560	60.95	256(xDSL)
8K	9216	219.43	896(DVB-T)

<표 7>은 Xilinx vertex-4로 합성하여 각 포인트에 따른 지연시간을 나타낸다. 제안한 FFT 연산기는 42.076MHz로 동작 하였을때 여러 가지 OFDM 시스템에 적용이 가능하다.

5. 결론

본 논문은 OFDM 상에서 실시간 처리가 가능하고 다양한 N포인트의 FFT를 빠르게 처리하기 위해서 공유메모리 기반의 가변길이 FFT 프로세서를 제안하였다. 가변길이 FFT 프로세서를 구현하기 위해서 첫째 기존 In-place 메모리 어드레싱 구조의 고정된 radix-N의 구조를 수정하여 가변 구조로 동작하는 공유 메모리를 구현하였으며, 둘째 공유 메모리상에서 N포인트의 FFT를 빠르게 처리하기 위해서 radix-8/4/2 DIF FFT를 제안하였다. 마지막으로 Recursive sin/cos 함수 생성방법을 이용한 회전인자 발생기를 구현하였다.

FFT 프로세서를 Vertex-4 FPGA로 구현함으로써 OFDM시스템에 많이 사용되는 64, 128, 256, 512, 1024, 2048, 4096 그리고 8192 포인트의 FFT 연산이 가능하였으며, 42.076MHz의 주파수로 동작함으로 802.11a, 802.16a, DAB, DVB-T, ADSL, VDSL등 여러 가지 OFDM 시스템에 적용이 가능하다.

본 논문에서 구현한 FFT 프로세서는 기존의 ROM기반 Lookup 테이블 방식의 회전인자 생성기에 비해서 ROM에 저장된 bit수를 약 80% 줄일 수 있었으며, 또한 연산 사이클이 Radhouane[10]의 FFT와 비교하였을 때 약 90%가 그리고 Moon[11]의 FFT 보다 약 62%가 향상되었다. <표 7>에서 보여준 것과 같이 각 포인트 별로 OFDM 표준에 맞추어 동작함을 확인 할 수 있다.

본 논문에서는 radix-8을 기반으로 FFT 연산을 처리함으로써 radix-4나 radix-2보다 곱셈기와 덧셈기가 많이 사용된다. 따라서 향후 연구방향으로 제안한 버터플라이 연산기의 곱셈기와 덧셈기를 줄일 수 있는 연구를 진행할 예정이다.

참 고 문 헌

[1] Chi-Hong Su, Jen-Ming Wu(2006). “Reconfigurable FFT Design for Low Power OFDM Communication

System," IEEE Trans. Circuit Syst. Video Technol, 6(2), 191-199.

[2] Danny Cohen(1976). "Simplified control of FFT hardware,"IEEE Trans. Acoust., Speech, Signal Processing, vol.ASSP-24, 577-579.

[3] Erling H. Wold, Alvin M. Despain(1984). "Pipeline and Parallel-Pipeline FFT Processors for VLSI Implementations," IEEE Trans. Computers, 33(5), 414-426.

[4] H. F. Lo, M. D. Shieh, and C. M. Wu(2001). "Design of an Efficient FFT Processor for DAB System," IEEE International Symposiumon Circuitsand Systems, 4, 654-657.

[5] Jen-Chuan Chi, Sau-Gee Chen,B(2004). "An Efficient FFT Twiddle Factor Generator," EUSIPCO(XII. European Signal Processing Conference).

[6] L. G. Johnson(1992). "Conflict Free Memory Addressing for Dedicated FFT Hardware," IEEE Trans. on Circuit and System-II: Analog and Digital Signal Processing, 39(5), 312-316.

[7] M. C. Pease(1969). "Organization of large scale fourier processors," J.Assoc. Comput. Mach, 16, 474-482.

[8] Nicola E. L'INSALATA, Sergio SAPONARA, Luca FANUCCI, Pierangelo TERRENI(2008). "Automatic Synthesis of Cost Effective FFT/IFFT Cores for VLSI OFDM Systems," IEICE Trans. of Electron, E91-C(4)

[9] Richard Van Nee, Ramjee Prasad(2000). OFDM for Wireless Multimedia Communications.

[10] R. Radhouane, P. Liu, and C. Modlin(2001). "Miniminzing the memory requirement for continuous flow FFT implementation: continuous flow mixed mode FFT (CFMM-FFT)," Proc. IEEE Int. Symp. Circuit Syst. 1, 133-136.

[11] Sang-Chul Moon and In-Cheol Park(2003), "Area-efficient memory-based architecture for FFT processing," Proceedings of the 2003 International Symposium on Circuits and Systems , 5.

[12] Shousheng He, Mats Torkelson(1998). "Designing Pipeline FFT Processor for OFDM (de)Modulation," ISSSE' 98, 257-262.

[13] Shuenn-Yuh Lee, Chia-Chyang Chen, Chyh - Chyang Lee, Chih-Jen Cheng(2006). "A Low-Power VLSI Architecture for a Shared-Memory FFT Processor with a Mixed-Radix Algorithm and a Simple Memory Control Scheme," ISCAS. 157-160.

[14] Youn-Seog Chang, Sin-Chong Park(2004) "An Enhanced Memory Assignment Scheme for Memory-Based FFT Processor," IEICE Trans. on Fundamentals of Electronics, E87-A(11), 3020-3024.

[15] Yutai MA(1999). "An effective memory addressing scheme for FFT processors," IEEE Trans. Signal processing, 47, 907-911.

[16] Xiaojin, Li, Zongsheng Lai, Jianmin Cui(2007). "A Low Power and Small Area FFT Processor for OFDM Demodulator,"Consumer Electronics, IEEE Transactions on, 53, 274-277.

[17] http://www.xilinx.com

김 영 진

- 1999년 2월 : 상지대학교 전산학과(이학사)
- 2001년 8월 : 경희대학교 전자계산공학과(공학석사)
- 2003년 8월 : 경희대학교 전자계산공학과(박사과정 수료)
- 2003년 9월~현재 : 경희대학교 전자계산공학과 박사과정
- 2010년 6월~현재 : 대양전자통신(주)책임연구원
- 관심분야 : 컴퓨터 구조, 병렬처리, SoC, FPGA, 영상처리, 음성처리
- E-Mail : jerryjin@khu.ac.kr

김 형 호

- 1989년 2월 : 경희대학교 전자계산공학과 졸업(공학사)
- 1992년 8월 : 경희대학교 전자계산공학과 졸업(공학석사)
- 1998년 3월 : 일본 게이오대학 계산기과학과(박사수료)
- 1998년~현재 : 세한대학교 컴퓨터교육과 교수
- 관심분야 : 컴퓨터구조, 신경회로망, ICT교육, 물류정보
- E-Mail : hhkim@sehan.ac.kr

이 현 수

- 1979년 2월 : 경희대학교 전자공학과(공학사)
- 1982년 4월 : 일본 게이오대학원 전기공학과(공학석사)
- 1985년 4월 : 일본 게이오대학원 전기공학과(공학박사)
- 2005년~2008년 : 경희대학교 전자정보대학 학장 및 정보통신대학원 원장
- 1985년~현재 : 경희대학교 컴퓨터공학과 교수
- 관심분야 : 컴퓨터구조 및 VLSI, 병렬처리, SoC, 패턴 인식, 신경망, 음성처리
- E-Mail : leehs@khu.ac.kr