논문 2007-44TC-11-2

Mapping 기법을 이용한 효율적인 IFFT 설계

(Efficient IFFT Design Using Mapping Method)

장 인 걸*, 김 용 은*, 정 진 균**

(In-Gul Jang, Yong-Eun Kim, and Jin-Gyun Chung)

요 약

FFT(Fast Fourier Transform) 프로세서는 WiBro, DAB, UWB와 같은 OFDM 시스템의 구현에 있어 중요한 블록 중 하나이다. 현재, FFT 프로세서의 구현에 관한 연구는 계속 이루어지고 있으며 대부분의 연구들은 곱셈기의 수나 면적감소, 메모리 사이즈 감소, 제어회로를 감소시키는 것에 초점을 두어 진행되고 있다. 본 논문에서는 IFFT(Inverse Fast Fourier Transform)에서 요구되는 메모리를 감소시키기 위하여 mapping 방법을 토대로 한 새로운 IFFT 설계방법을 제안한다. WiBro를 위한 1024포인트 IFFT를 설계하기 위해 Radix-2^4 SDF(Single-Path Delay Feedback) 구조를 사용하여 시뮬레이션하였으며 제안된 IFFT 설계방법으로 구현했을 시 기존의 방법으로 설계했을 때와 비교하여 메모리가 차지하는 면적에서 60%이상의 감소와 입력비트별로 다양한 SQNR(Signal-to-Quantization-Noise Ratio) 이득을 보였다.

Abstract

FFT(Fast Fourier Transform) processor is one of the key components in the implementation of OFDM systems such as WiBro, DAB and UWB systems. Most of the researches on the implementation of FFT processors have focused on reducing the complexities of multipliers, memory and control circuits. In this paper, to reduce the memory size required for IFFT(Inverse Fast Fourier Transform), we propose a new IFFT design method based on a mapping method. By simulations, it is shown that the proposed IFFT design method achieves more than 60% area reduction and much SQNR(Signal-to-Quantization-Noise Ratio) gain compared with previous IFFT circuits.

Keywords: IFFT, Mapping, Quantization, Memory, Radix-2i

I. 서 론

최근 유무선 OFDM 방식은 WiBro, DMB, DVB-T, DSL 등 많은 통신시스템의 표준기술로 채택되고 있다[1~2]. 광대역 OFDM 무선 전송방식의 FFT(Fast Fourier Transform) 구현에 면적과 전력 면에서 효율적인 알고리즘을 적용해야하며 파이프라인 FFT 방식이 널리 쓰이고 있다[3~7].

다양한 파이프라인 FFT 구조가 제안되었으며 메모리나 제어회로 측면에서 효율적인 구조 중 하나가 R2^2SDF(Radix-2^2 Single-Path Delay Feedback)구조이다. R2^2SDF 구조의 기반이 되는 Radix-2^2 FFT 알고리즘은 기존의 Radix-2 알고리즘의 구조를 개량한 것으로 Radix-4 구조의 곱셈량과 Radix-2 구조의 간단한 연산구조를 복합 수용하여 복잡한 곱셈을 효과적으로 줄이고, 제어를 쉽게 만든 구조이다[8]. 이와 관련해서 다양한 Radix-2i SDF 구조와 복합적인 구조가 연구되었다[9].

OFDM 방식에서 광대역과 이동성에 대한 요구가 높아짐에 따라 고성능, 큰 포인트 사이즈의 FFT 프로세서에 대한 요구가 증가하고 있다. 그러나 포인트 수가 증가할수록 FFT에서 메모리가 차지하는 면적이 크게 증가하는 문제점을 가지고 있다. 본 논문에서는 이동성과 광대역의 특징을 지닌 OFDM 시스템에 적용 가능한

* 학생회원, ** 정회원, 전북대학교 전자정보공학부
(Div. of Electronic & Information Engineering Chonbuk University)

※ 이 연구에 참여한 연구자는 2단계 BK21사업의 지원비를 받았음, This work was supported by the second stage of Brain Korea 21 Project.
접수일자: 2007년8월14일, 수정완료일: 2007년11월18일

Radix-2^i SDF IFFT에서 IFFT에 입력되는 변조신호의 종류가 정해져 있다는 사실에 기반하여 메모리 사이즈를 감소시키는 방법을 제안한다.

Ⅱ장에서는 변조방식에 따른 IFFT내 butterfly의 출력에 대해 설명하고, Ⅲ장에서는 제안하는 방법으로 입력을 mapping하는 것을 설명한다. Ⅳ장에서는 제안하는 방법으로 Radix-2^4 SDF 연산방식을 가진 WiBro용 1024포인트 IFFT를 설계하여 결과를 비교하고, 끝으로 Ⅴ장에서 결론을 맺는다.

Ⅱ. 변조방식과 Butterfly 출력

OFDM 시스템에서 기저대역의 data들은 그림 1과 같이 다양한 형태로 변조되어 IFFT에 입력된다. 이때 변조된 data들은 일반적으로 -1과 1사이로 정규화 된다. 또, IFFT로 입력되는 신호는 복소수로서 변조된 신호의 실수와 허수는 그 종류가 정해져 있다. 이 신호들을 살펴보면 서로 등 간격을 이루며 실수와 허수축을 중심으로 대칭이라는 특징이 있다.

기존의 IFFT에서는 이 정규화 된 값을 IFFT의 첫 번째 stage에서 butterfly연산을 거치기 전에 각각의 시스템에서 요구하는 성능에 따라 다양한 비트로 양자화

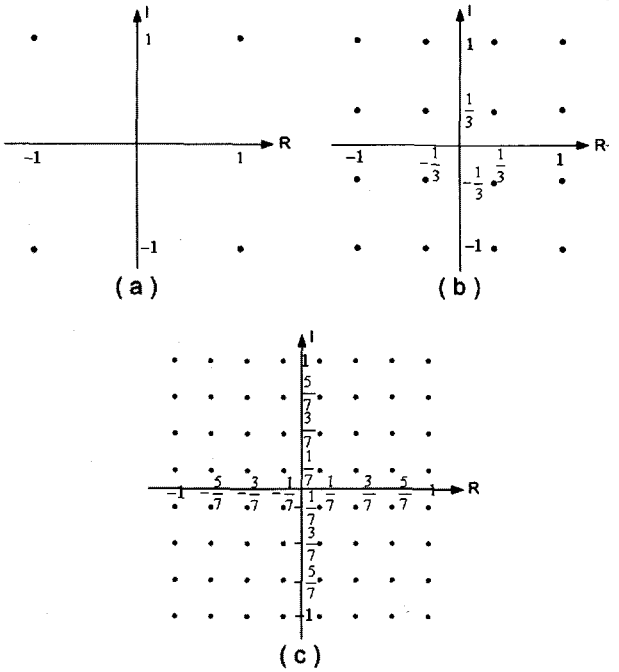


그림 1. 다양한 변조방식 :
(a) QPSK, (b) 16-QAM, (c) 64-QAM

Fig. 1. Various modulation techniques :
(a) QPSK, (b) 16-QAM, (c) 64-QAM.

하여 연산이 이루어진다. 그 결과 높은 SQNR(Signal-to-Quantization-Noise Ratio)을 요구하는 시스템에서는 양자화로 생기는 에러를 줄이기 위해 양자화 비트를 증가시키고 그 결과, 양자화 된 값을 저장하기 위한 메모리가 커지게 된다.

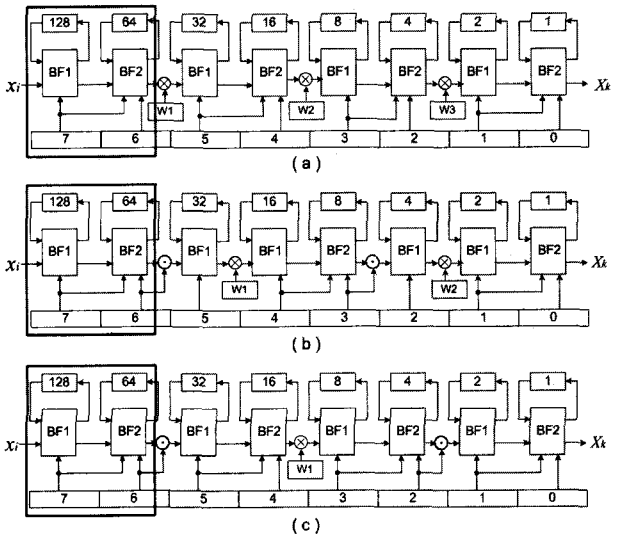


그림 2. Radix-2^i SDF 256포인트 IFFT 구조 :
(a) Radix-2^2 SDF 256포인트 IFFT 구조,
(b) Radix-2^3 SDF 256포인트 IFFT 구조,
(c) Radix-2^4 SDF 256포인트 IFFT 구조

Fig. 2. Radix-2^i SDF 256 point IFFT structure :
(a) Radix-2^2 SDF 256point IFFT structure,
(b) Radix-2^3 SDF 256point IFFT structure,
(c) Radix-2^4 SDF 256point IFFT structure.

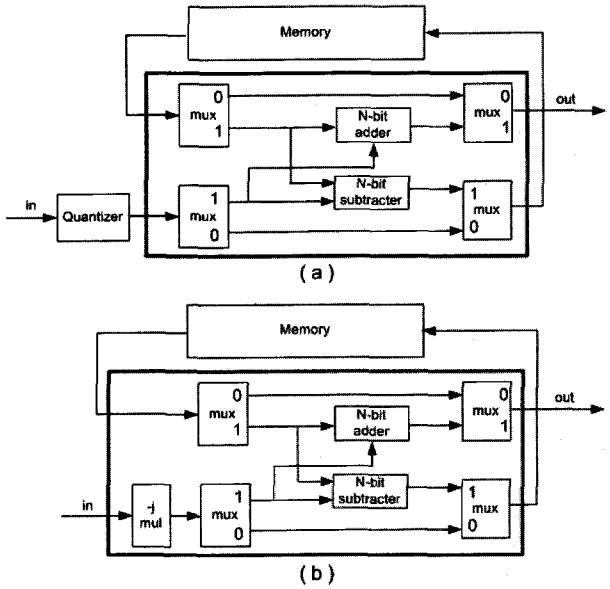


그림 3. BF1 및 BF2의 구조 :
(a) BF1, (b) BF2

Fig. 3. BF1 and BF2 structure :
(a) BF1, (b) BF2.

그림 2는 Radix-2^i SDF 256포인트 IFFT 구조를 나타내며, 그림에서 ⊙는 복소 상수 곱셈기이고 ⊗는 프로그래머블 복소 곱셈기를 나타낸다. 각 그림의 상단에 표시된 숫자들은 레지스터 개수를 의미한다. 그림 3은 BF1(Butterfly type 1)과 BF2(Butterfly type 2)의 구조를 보인다. 그림 2의 각 IFFT 구조는 곱셈기의 개수만 다르다는 것을 알 수 있다. 각 구조에서 박스로 표시된 부분의 메모리/레지스터(192개)가 전체 IFFT의 메모리/레지스터(255개)의 75%를 차지하고 있다. 박스내 BF1 출력이 12비트 일 경우 12 × 128 사이즈의 레지스터가 필요하지만 BF1의 출력을 2비트로 감소시킬 수 있다면 2 × 128 사이즈의 레지스터만 사용하면 되므로 큰 면적을 감소시킬 수 있다. 또한 박스로 표시된 부분에서는 회전인자와의 곱셈연산이 이루어지기 전이므로 입력의 덧셈과 뺄셈만 이루어진다. 따라서 변조되어 IFFT로 들어오는 입력의 종류가 정해져 있으므로 첫 번째 butterfly인 stage 1과 두 번째 butterfly인 stage 2에서 출력되는 값의 종류도 쉽게 알 수 있다. 표 1은 각 변조방식에 따른 입력의 종류와 stage 1(S1)과 stage 2(S2)

표 1. Stage 1과 stage 2에서 나올 수 있는 실제 값의 종류

Table 1. Types of possible actual value going through stage 1 and stage 2.

변조방식	QPSK	16-QAM	64-QAM
입력종류	-1,1 (2개)	-1,-1/3, 1/3,1 (4개)	-1,-5/7,-3/7,-1/7, 1/7,3/7,5/7,1 (8개)
S1 out 종류	-1,0,1 (3개)	-1,-2/3,-1/3 ,0,1/3,2/3,1 (7개)	-1,-6/7,-5/7,-4/7, -3/7,-2/7,-1/7, 0,1/7,2/7,3/7, 4/7,5/7,6/7,1 (15개)
S2 out 종류	-1,-1/2, 0,1/2,1 (5개)	-1,-5/6,-4/6 ,-3/6,-2/6,- 1/6,0,1/6,2/6, 3/6,4/6,5/6,1 (13개)	-1,-13/14,-12/14, -11/14,-10/14,-9/14, -8/14,-7/14,-6/14, -5/14,-4/14,-3/14, -2/14,-1/14,0,1/14, 2/14,2/14,3/14,4/14, 5/14,6/14,7/14,8/14, 9/14,10/14,11/14, 12/14,13/14,1 (29개)

를 거치고 나올 수 있는 스케일 된 출력의 실제 값을 나타낸다.

III. Mapping 기법을 이용한 제안방법

표 1에 보인바와 같이 변조된 신호는 그 종류가 정해져 있고 stage 1과 stage 2에서는 덧셈과 뺄셈연산만 수행하기 때문에 연산의 결과를 쉽게 예측 할 수 있다. 따라서 연산의 결과값을 mapping에 의해 구함으로써 메모리를 줄일 수 있다. 표 2에 간단한 변조방식인 QPSK에 대하여 mapping을 적용한 제안방법 1을 보였다. 괄호 안은 실제 값에 대해 mapping된 표현을 의미한다. 즉, -0.5(0)은 실제 값 -0.5를 이진수 표현 0으로 mapping하였음을 의미한다.

표 2. QPSK로 변조 시 stage 1 덧셈의 mapping

Table 2. Mapping of stage 1 addition(QPSK).

stage 1 덧셈		in_1	
		-0.5 (0)	0.5 (1)
in_2	-0.5 (0)	-1 (00)	0 (01)
	0.5 (1)	0 (01)	1 (10)

1. 제안방법 1

제안방법 1은 다음의 세 가지 step으로 표현할 수 있으며 뺄셈도 동일한 방법으로 처리할 수 있다.

step 1. 입력을 1/2로 scale down 한 후 입력의 종류를 나누어 mapping한다. 입력 종류의 개수 N_{in}이 다음 조건을 만족하면 입력은 W비트로 표현한다.

$$2^{W-1} < N_{in} \leq 2^{W} \quad (1)$$

절대값이 가장 큰 음수부터 시작하여 '00…00', '00…01', '00…10'과 같이 1씩 증가시키며 mapping한다.

step 2. 실제 값의 butterfly 연산이 이루어질 때 발생하는 결과 값을 종류를 나누어 mapping한다. Mapping 방법은 step 1과 같으나 mapping 결과는 W+1 비트를 사용한다.

step 3. 덧셈결과를 $out[W]$, $out[W-1]$,…, $out[0]$로 표시하고 카르노맵을 이용하여 각 out 비트를 in_1과 in_2의 함수로 표시한다.

표 3. QPSK로 변조 시 stage 2 덧셈의 mapping
Table 3. Mapping of stage 2 addition(QPSK).

stage 2 덧셈		in_1			
		-0.5(00)	0(01)	x(11)	0.5(10)
in_2	-0.5(00)	-1(000)	-0.5(001)	x	0(010)
	0(01)	-0.5(001)	0(010)	x	0.5(011)
	x(11)	x	x	x	x
	0.5(10)	0(010)	0.5(011)	x	1(100)

표 4. stage 2의 결과 (| : OR)
Table 4. Output of stage 2 (| : OR).

$out[2]$	$in_1[1] \cdot in_2[1]$
$out[1]$	$(in_1[1] \cdot \overline{in_2[1]}) \mid (in_1[0] \cdot in_2[0]) \mid (\overline{in_1[1]} \cdot in_2[1])$
$out[0]$	$(\overline{in_1[0]} \cdot in_2[0]) \mid (in_1[0] \cdot \overline{in_2[0]})$

표 2에서 보는 것과 같이 변조된 입력(-1, 1)을 양자화하지 않고 1/2로 scale down(-0.5, 0.5)하여 -0.5와 0.5를 각각 '0'과 '1'로 mapping한다. 덧셈결과 값(-1, 0, 1)을 음수인 '-1'부터 시작하여 '00', '01', '10'으로 차례로 mapping한다. 표 2에 step 3을 적용한 결과는 다음과 같다(· : AND, ^ : XOR, ¯ : NOT).

$$\text{덧셈결과 :}\quad out[1] = in_1 \cdot in_2,\quad out[0] = in_1 \wedge in_2 \tag{2}$$

$$\text{뺄셈결과 :}\quad out[1] = in_1 \cdot \overline{in_2},\quad out[0] = \overline{in_1 \wedge in_2} \tag{3}$$

step 1 ~ step 3을 통해서 얻은 *out*은 연산을 해서 얻어지는 것이 아니라 mapping된 입력의 조합에 의해 실제 값의 mapping 결과가 얻어진다.

Stage 2에서도 위와 같은 3가지 step으로 mapping을 통해 *out*을 얻을 수 있다. Stage 1의 *out*이 -1, 0, 1의 세 가지일 때 stage 2에서는 표 3에서 보는바와 같이 먼저 -1, 0, 1을 scale down하여 -0.5, 0, 0.5를 실제 값으로 간주한다. (Mapping된 표현은 '00', '01', '10'으로 동일하다.) 출력은 input보다 1비트가 증가한 3 비트로 표시하며 표 3과 같이 모든 가능한 *out*을 구한다. (카르노맵을 구성 할 때 입력의 종류가 3가지이므로 index '11'은 don't care항이 된다.) 표 3에 step 3을 적용해 얻은 결과는 표 4와 같다.

2. 제안방법 2

제안방법 1에서는 변조된 입력변수를 양자화 시키지 않고 mapping하고 이를 카르노맵을 이용하여 butterfly 출력으로 mapping시키는 조합회로를 얻었다. 하지만 제안방법 1은 입력의 종류가 8이상으로 늘어날 경우 그 복잡도가 매우 높아지며 또한 fan-in과 fan-out 문제를 발생시킨다.

대수적으로 서로 다른 N개의 수를 더할 경우에 덧셈 결과의 종류는 $N(N+1)/2$개 이다. 그러나 서로 등 간격을 이루며 0을 중심으로 대칭인 수의 경우에는 덧셈 결과의 종류는 $2N-1$이 되며 덧셈 결과도 0을 중심으로 대칭이고 등 간격을 유지하게 된다. 덧셈 결과의 종류가 줄어든 이유는 크기가 같고 부호가 반대인 수가 서로 상쇄되어 0이 되기 때문이다. 따라서 표 1과 같은 변조방식을 사용할 경우 입력 종류의 개수를 N_{in}, 덧셈 결과 종류의 개수를 N_{out}이라하면 다음 관계가 성립한다.

$$N_{out} = 2N_{in} - 1 \tag{4}$$

입력 워드길이 W를 가지는 mapping 된 입력을 X_{map}, mapping 된 입력 X_{map}을 양의 정수로 가정했을 때의 값을 X_{int}라 한다면 다음과 같이 표현 할 수 있다.

$$X_{map} = x_{W-1}x_{W-2}\cdots x_1x_0 \tag{5}$$

$$X_{int} = \sum_{i=0}^{W-1} x_i \cdot 2^i = x_{W-1} \cdot 2^{W-1} + \cdots + x_0 \cdot 2^0 \tag{6}$$

표 1과 같이 변조된 신호를 절대값이 가장 큰 음수부터 시작하여 '00…00', '00…01', '00…10'과 같이 1씩 증가시키며 mapping 했을 때의 실제 값 X_a와 X_{int} 사이의 관계는 다음과 같음을 보일 수 있다. (N은 신호의 개수)

$$X_a = \frac{2X_{int}}{N-1} - 1 \tag{7}$$

따라서 입력의 실제값을 $X_{a,in}$, 덧셈 결과의 실제 값을 $X_{a,out}$라 한다면 다음의 관계가 성립한다.($X_{int,in}$는 입력을 양의 정수로 가정했을 때의 값, $X_{int,out}$은 덧셈 결과를 양의 정수로 가정했을 때의 값이다.)

$$X_{a,in} = \frac{2X_{int,in}}{N_{in}-1} - 1 \tag{8}$$

$$X_{a,out} = \frac{2X_{int,out}}{N_{out}-1} - 1 \tag{9}$$

두 입력 신호의 실제 값을 각각 1/2로 scale down하여 더한 값이 원하는 출력 신호의 실제 값이 되므로 이를 식으로 나타내면 다음과 같다.(입력 in_1과 in_2의 실제 값이 $X_{a,in1}$, $X_{a,in2}$이며, 양의 정수로 가정 했을 때의 값이 $X_{int,in1}$, $X_{int,in2}$이다.)

$$\begin{aligned} X_{a.out} &= \frac{1}{2}X_{a,in1} + \frac{1}{2}X_{a,in2} \\ &= \frac{X_{int,in1}}{N_{in}-1} - \frac{1}{2} + \frac{X_{int,in2}}{N_{in}-1} - \frac{1}{2} \\ &= \frac{X_{int,in1}+X_{int,in2}}{N_{in}-1} - 1 \\ &= \frac{X_{int,in1}+X_{int,in2}}{(N_{out}+1)/2-1} - 1 \\ &= \frac{2(X_{int,in1}+X_{int,in2})}{N_{out}-1} - 1 \end{aligned} \tag{10}$$

식 (9)와 식 (10)으로부터 다음 관계가 성립함을 알 수 있다.

$$X_{int,out} = X_{int,in1} + X_{int,in2} \tag{11}$$

식 (11)은 입력된 실제 값을 1/2로 scale down해서 더한 출력값의 mapping 표현이 두 입력의 mapping 표현을 단순히 adder를 통해 더한 결과와 같다는 것을 의미한다. 예를 들어, 표 3에서 in_1과 in_2의 실제 값을 더한 결과의 mapping 표현은 in_1과 in_2의 mapping 표현을 더한 값과 같음을 알 수 있다. 따라서 제안방법 1과 같이 카르노맵을 이용하지 않고 덧셈기를 통하여 mapping 회로를 간단히 구성 할 수 있다.

변조 된 모든 신호는 0을 중심으로 대칭이므로 뺄셈 연산 수행 시 실제값의 반대부호에 해당하는 값의 mapping 값을 할당함으로써 덧셈연산으로 대체 할 수 있다. 표 5는 butterfly내의 뺄셈 연산을 위해서 16-QAM 변조된 입력을 변환한 것을 나타내며 표 6은 16-QAM 변조시 stage 1에서 덧셈의 mapping을 보인다. 덧셈 출력은 3비트로 표현되며 0을 중심으로 서로 대칭임을 알 수 있다.

표 5. 16-QAM의 입력과 뺄셈 변환값의 mapping
Table 5. Mapping of input and the value with opposite sign(16-QAM).

16-QAM 입력	1/2 down scale	변환 전 mapping	뺄셈 변환값	변환 후 mapping
-1	-1/2	00	1/2	11
-1/3	-1/6	01	1/6	10
1/3	1/6	10	-1/6	01
1	1/2	11	-1/2	00

그림 4는 제안방법 2를 나타내는 블록도이다. 입력되는 실제 값을 mapping하는 블록과 뺄셈 변환회로를 나타내는 Map_sub, stage 2에서 나온 mapping 값을 양자화 된 값으로 바꿔주는 mapper 블록이 기존의 구조와는 다른 부분이다. WiBro를 위한 1024포인트 FFT (SQNR = 70 dB, 변조방식 : 64-QAM)에서 필요한 메모리/레지스터는 입력을 10비트로 양자화 했을 경우 첫 번째 stage에서 11 × 512, 두 번째 stage에서 12 × 256 이지만 제안방법 2에서는 각각 4 × 512, 5 × 256이다. 그리고 N-bit adder의 경우 기존구조에서는 10비트, 11비트의 덧셈이 이뤄지지만 제안방법 2는 3비트, 4비트의 덧셈이 이뤄진다.

표 6. 16-QAM 변조 시 stage 1 덧셈의 mapping
Table 6. Mapping of stage 1 addition(16-QAM).

stage 1 덧셈		in_1			
		-1/2(00)	-1/6(01)	1/6(10)	1/2(11)
in_2	-1/2(00)	-1(000)	-2/3(001)	-1/3(010)	0(011)
	-1/6(01)	-2/3(001)	-1/3(010)	0(011)	1/3(100)
	1/6(10)	-1/3(010)	0(011)	1/3(100)	2/3(101)
	1/2(11)	0(011)	1/3(100)	2/3(101)	1(110)

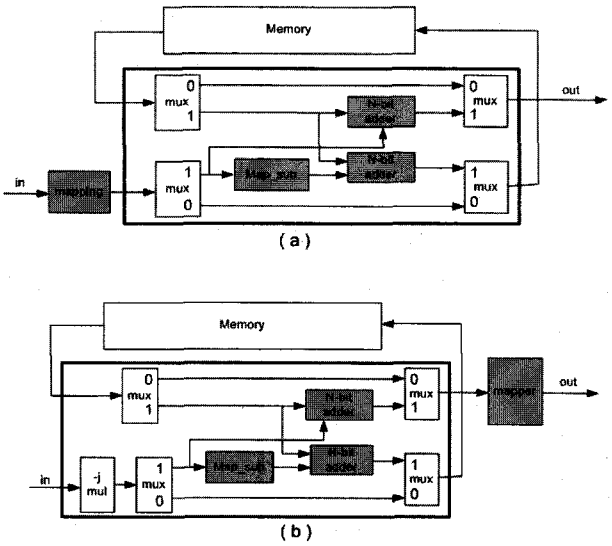


그림 4. 제안방법 2의 구조 :
(a) BF1, (b) BF2
Fig. 4. Structure of proposed algorithm 2 :
(a) BF1, (b) BF2.

제안방법 2는 다음의 세 가지 step으로 표현할 수 있다.

step 1. 입력의 종류를 나누어 제안방법 1의 step 1과 같은 요령으로 mapping한다.

step 2. 실제 값의 butterfly 연산이 이루어질 경우에 뺄셈을 adder를 통해 연산할 수 있도록 부호를 바꾼 뺄셈 변환값을 구한다. 현재 입력되는 mapping값(now_map)으로 뺄셈 변환값의 mapping(mod_map)을 얻어야 하므로 now_map을 변수로 하고 mod_map에서 각 비트별로 '1'을 카르노맵으로 묶어 변환회로를 만든다.

step 3. Adder를 이용하여 out을 구한다.

표 5에서 보는 것과 같이 16-QAM 변조된 입력값(-1, -1/3, 1/3, 1)을 1/2로 scale down(-1/2, -1/6, 1/6, 1/2)하여 가장 절대값이 큰 음수인 -1/2부터 시작하여 '00', '01', '10', '11'로 차례로 mapping 한다. 뺄셈 변환값에 해당하는 값들을 새로 mapping한다. 뺄셈 변환회로는 표 5에서 보는 바와 같이 입력이 실제 값 0을 포함하지 않을 경우 변환 전 mapping 값을 단순히 complement 시킴으로써 뺄셈 변환값의 mapping을 얻을 수 있다. 하지만 입력이 0을 포함하고 있을 경우 단순히 complement를 통해서 가능하지 않고 뺄셈 변환값의 mapping을 가지고 뺄셈 변환회로를 만들어야 한다. 표 7의 경우 뺄셈 변환값에 해당하는 mod_map[0] = now_map[0]이다. 나머지 뺄셈 변환회로를 만들기 위한 카르노맵은 표 8 및 표 9와 같고 정리하면 표 10과 같다.

표 7. 16-QAM IFFT의 stage 2의 입력과 뺄셈 변환값의 mapping

Table 7. Mapping of stage 2 input and the value with opposite sign (16-QAM).

stage1 out	1/2 down scale	now_map [2:0]	뺄셈 변환값	mod_map [2:0]
-1	-1/2	000	1/2	110
-2/3	-1/3	001	1/3	101
-1/3	-1/6	010	1/6	100
0	0	011	0	011
1/3	1/6	100	-1/6	010
2/3	1/3	101	-1/3	001
1	1/2	110	-1/2	000

표 8. 뺄셈 변환회로를 만들기 위한 카르노맵 (mod_map[1])

Table 8. K-map for subtraction-to-addition converter (mod_map[1]).

mod_map[1]	00	01	11	10
0	1	0	1	0
1	1	0	x	0

표 9. 뺄셈 변환회로를 만들기 위한 카르노맵 (mod_map[2])

Table 9. K-map for subtraction-to-addition converter (mod_map[2]).

mod_map[2]	00	01	11	10
0	1	1	0	1
1	0	0	x	0

표 10. 뺄셈 변환회로를 위한 mod_map의 now_map을 이용한 표현

Table 10. Representation of mod_map using now_map.

mod_map[2]	$\overline{now\_map[2] \cdot now\_map[1] \cdot now\_map[0]}$
mod_map[1]	$(\overline{now\_map[1]} \cdot \overline{now\_map[0]}) \mid (now\_map[1] \cdot now\_map[0])$
mod_map[0]	$now\_map[0]$

제안방법 1에서는 각 stage에서의 *out*을 만들기 위해 현재의 입력으로 조합회로를 만들었지만 제안방법 2는 adder로 입력되는 뺄셈 변환값을 얻기 위해 조합회로를 만든다. 실제적인 stage *out*은 adder를 통해 출력된다. Adder를 통해 나온 stage 2의 출력 *out*[3:0]은 butterfly를 통해 나오는 실제 값의 mapping address를 나타내며 시스템에서 요구하는 성능에 따라 양자화 된 값이 저장되어 있어야 한다.

그림 4에 보인 바와 같이 *out*[3:0]은 양자화 한 값을 저장하고 있는 mapper로 입력되어 양자화 된 값을 출력하고 이는 회전인자와 곱해져서 stage 3부터는 기존의 방법과 동일한 연산을 이루게 된다.

Ⅳ. 시뮬레이션 및 비교

1. MATLAB 시뮬레이션

WiBro에 적용할 수 있는 그림 5와 같은 1024-point

SDF Radix-2^4 구조의 IFFT 블록을 제안한 방법과 기존의 방법[9]으로 fixed point 시뮬레이션 하였다. 시뮬레이션 방법은 아래와 같다.

① 1024개의 64-QAM 입력 data 생성
② 기존의 구조로 1024-point IFFT 연산
③ MATLAB의 IFFT 함수로 1024-point IFFT 수행
④ 제안된 구조로 1024-point IFFT 연산 후 동일한 방법을 거쳐 SQNR의 이득비교

기존방법으로 시뮬레이션 한 결과 twiddle factor가 12비트 일 경우 양자화 비트가 하나씩 증가할수록 낮은 비트에서는 약 6 dB SQNR이 개선되다가 60 dB 이상 증가하지 않았다. 또한 제안한 방법과 기존방법의 시뮬레이션 결과를 비교했을 때는 양자화 비트를 약 2비트 적게 해도 같은 결과를 얻었다. 70 dB 이상의 SQNR은 twiddle factor가 14비트 일 경우에 도달하였다. 본 논문에서 제안한 mapping기법을 사용하였을 경우의 비트별 SQNR 및 SQNR 이득을 각각 표 11과 표 12에 나타내었다.

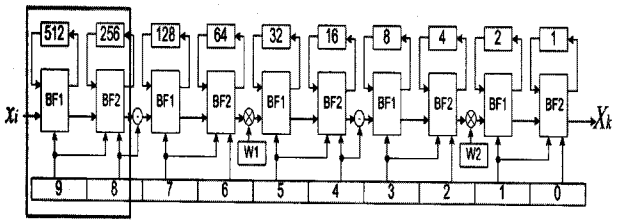


그림 5. Radix-2^4 SDF 1024포인트 IFFT 구조
Fig. 5. Radix-2^4 SDF 1024-point IFFT structure.

표 11. 기존방법과 제안방법의 bit 수에 따른 SQNR 비교 (W_T : twiddle factor 비트수)
Table 11. SQNR comparison of existing and prosed method (W_T : The number of twiddle factor bits).

양자화 비트수	9	10	11	12	13	14	15	16	W_T
기존[9] SQNR	44.6	50.1	55.0	57.8	59.3	59.5	59.6	59.6	12
제안 SQNR	54.5	57.7	59.0	59.5	59.6	59.6	59.6	59.6	12
기존[9] SQNR	44.8	50.7	56.7	62.2	66.5	69.8	70.1	71.3	14
제안 SQNR	56.0	61.6	66.6	69.9	71.1	71.5	71.6	71.6	14

표 12. 양자화 bit 수와 twiddle factor 비트수에 따른 제안방법의 기존방법에 대한 SQNR 이득 결과
Table 12. SQNR gain for quantization bits and twiddle factor bits.

양자화 비트수	9	10	11	12	13	14	15	16
W_T =12 SQNR(dB)	9.9	7.6	4.0	1.7	0.2	0.1	0	0
W_T =14 SQNR(dB)	11.2	10.9	9.9	7.5	4.6	1.7	1.5	0.3

2. Verilog 시뮬레이션

그림 5의 구조와 같은 WiBro용 Radix-2^4 SDF 1024 포인트 IFFT를 제안한 구조와 기존의 구조로 Verilog-HDL로 코딩하였고 Xilinx ISE tool에서 Modelsim을 통해 시뮬레이션을 수행하였다. 또 Synplify Pro를 통해 합성하여 면적을 비교하였다. WiBro에서는 QPSK, 16-QAM, 64-QAM 변조방식을 다 지원해야 하며 SQNR이 70 dB이상이어야 한다. 변조된 신호는 10비트로 양자화되어 첫 번째 stage로 입력된다. 첫 번째 메모리와 두 번째 메모리의 워드길이는 각각 11비트, 12비트이다.

Xilinx의 XC2V3000- 6bf957을 디바이스로 선택하여 Xilinx ISE tool을 이용해 Synplify Pro로 synsynthesis를 수행하였다. 곱셈계수가 저장되기 위한 ROM을 2개 사용하였다. 또한 Bit-reverse를 위해 RAM을 2개 사용하였다. 그림 5에서 표시한 부분을 제안한 방법과 기존의 방법으로 설계하여 표 13에서 비교하였으며 제안한 방법에 의해 61%의 면적 감소를 얻을 수 있음을 알 수 있다.

표 13. 제안한 구조와 기존구조의 결과 비교
Table 13. Result comparison of proposed algorithms and the existing method.

	기존[9]	제안	면적 감소율
면적(gate)	141,887 100(%)	55,723 39(%)	86,164 61(%)

V. 결 론

본 논문에서는 WiBro를 포함한 OFDM 시스템의 핵심 블록인 IFFT에서 큰 면적을 차지는 첫 번째 stage와 두 번째 stage의 메모리를 줄이기 위해 Radix-2^i

SDF 구조에 적용할 수 있는 IFFT의 효율적인 설계방법을 제시하였다. 그리고 WiBro용 Radix-2^4 SDF 1024 포인트 IFFT를 제안방법으로 설계하였다.

제시한 알고리즘을 토대로 MATLAB을 이용해 simulation하여 기존의 양자화 방법과 비교하였다. 양자화비트가 9비트 일 때 약 10 dB정도의 SQNR 성능향상이 있음을 보였다. 양자화비트마다 이득은 다르지만 양자화비트가 작아질수록 SQNR의 이득은 컸다. 이는 mapping을 통해 stage 1과 stage 2에서 양자화 해야 하는 부분을 제거하고 회전계수와 곱해지기 전에 양자화 함으로써 SQNR이 높아지기 때문이다.

면적의 관점에서는 기존의 IFFT 구조로 synthesis 한 경우와 비교하여 약 61%의 감소를 얻었다.

본 논문에서 제안한 방법은 높은 정밀도를 요구하는 큰 포인트의 IFFT에서 더 큰 효과를 가져 올 수 있다. 또한 입력의 종류가 정해져 있고 곱해지는 종류의 수가 적은 곱셈에서도 응용 될 수 있을 것이다.

참 고 문 헌

[1] TTAS.KO-06.0064R1, 2.3GHz 휴대인터넷 표준(물리계층), 한국정보통신기술협회, Dec. 2004.

[2] John A. C. Bingham, "Multicarrier Modulation for Data Trans- mission: An Idea Whose Time Has Come," IEEE Commun. Mag., Vol.28, no.5, pp 5~14, May 1990.

[3] E. H. Wold, A. M. Despain, "Pipeline and Parallel Pipeline FFT Processors for VLSI Implementation," IEEE Trans. Comput., C-33(5), pp.414-426, May 1984.

[4] J. Melander, "Design of SIC FFT Architectures, Linkoping Studies in Science and Technology," Thesis No. 618, LinKoping University, Sweden, 1997.

[5] Thoms Lenart and Viktor Owall, "A Pipelined FFT Processor using Data Scaling with Reduced Memory Requirements", In Proc. NORCHIP, 2002.

[6] S. Yu and E. E. Swartzlander, Jr., "A pipelined architecture for the multidimensional DFT," IEEE Transactions on Signal Processing, vol. 49, pp. 2096-2102, 2001.

[7] El-Khashab, A. M., Swartzlander, E. E., Jr. "A modular pipelined implementation of large fast Fourier transforms," IEEE Transactions on Signals, Systems and Computers, vol. 2, pp. 995-999, 2002.

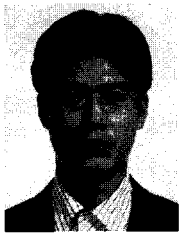
[8] S. He and M. Torkelson, "Designing pipeline FFT processor for OFDM (de)Modulatioin," in Proc. IEEE URSI Int. Symp. Signals, Syst., Electron., pp. 257-262, 1998.

[9] J. Y. Oh and M. S. Lim, "New radix-2 to the 4th power pipeline FFT processor," IEICE Trans. Electron., vol.E88-C, no. 8, pp.1740-1746, Aug. 2005.

저 자 소 개

장 인 걸(학생회원)
2005년 전북대학교 전자공학과 학사 졸업.
2007년 전북대학교 정보통신 공학과 석사 졸업.
<주관심분야 : 통신, 신호처리, 반도체>

정 진 균(정회원)
1985년 전북대학교 전자공학 학사 졸업
1989년 미국 미네소타 주립대학 전기공학 석사 졸업
1991년 미국 미네소타 주립대학 전기공학 박사 졸업
<주관심분야 : 통신, 컴퓨터, 신호처리, 반도체>



김 용 은(학생회원)
2005년 전북대학교 전자공학과 학사 졸업
2007년 전북대학교 정보통신 공학과 석사 졸업
2007년~현재 전북대학교 전자정보공학부 박사과정
<주관심분야 : 통신, 신호처리, 반도체>