Fast Fourier Transform

1조(고종완, 서윤철, 권희식, 장찬원)

Contents

01 Overview

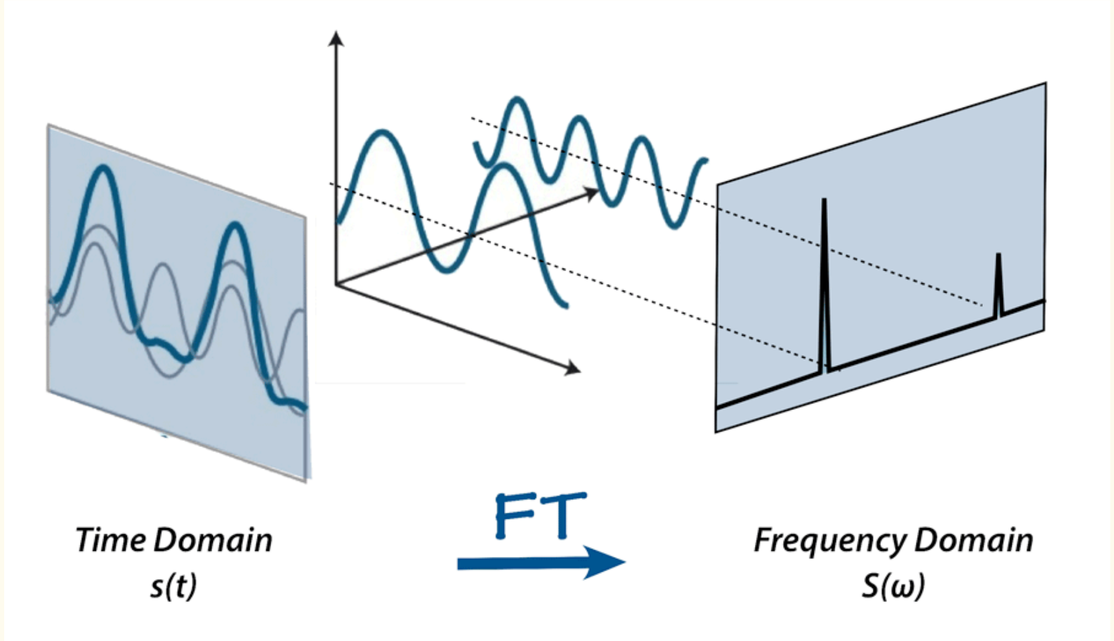
주제 소개

FFT (고속 푸리에 변환)

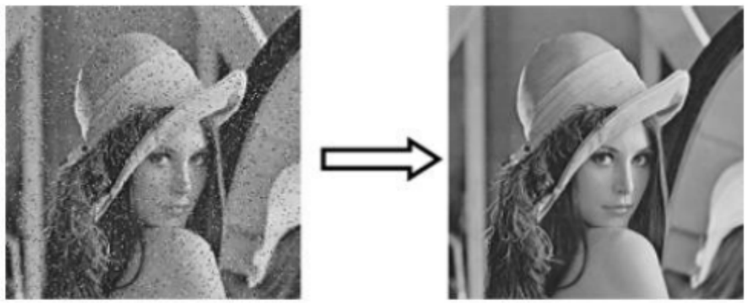
시간 영역 신호를 주파수 영역으로 변환

DFT (이산 푸리에 변환)

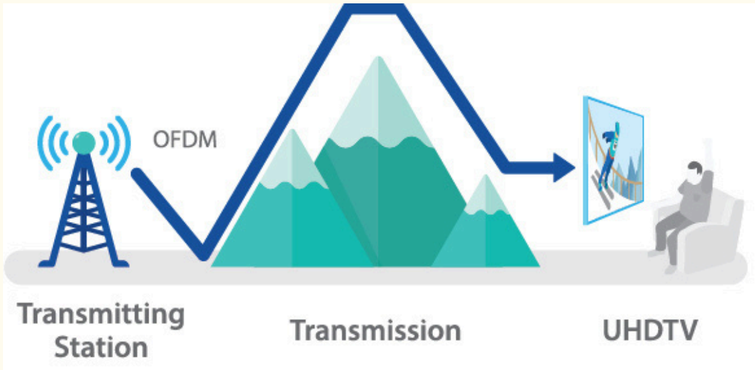
FFT(고속 푸리에 변환)의 계산 복잡도를 획기적으로 줄여 효율적으로 계산하기 위한 빠른 알고리즘



영상 처리



OFDM



영상 처리

01 Overview

주제 소개

Radix

FFT 알고리즘의 기본 연산 단위 크기

- Radix-2, 단순한 버터플라이 구조
- Radix-4, 적은 곱셈기 수
- Radix-2^2, Radix-4의 곱셈 효율성 & Radix-2의 구조적 단순성

N-Point

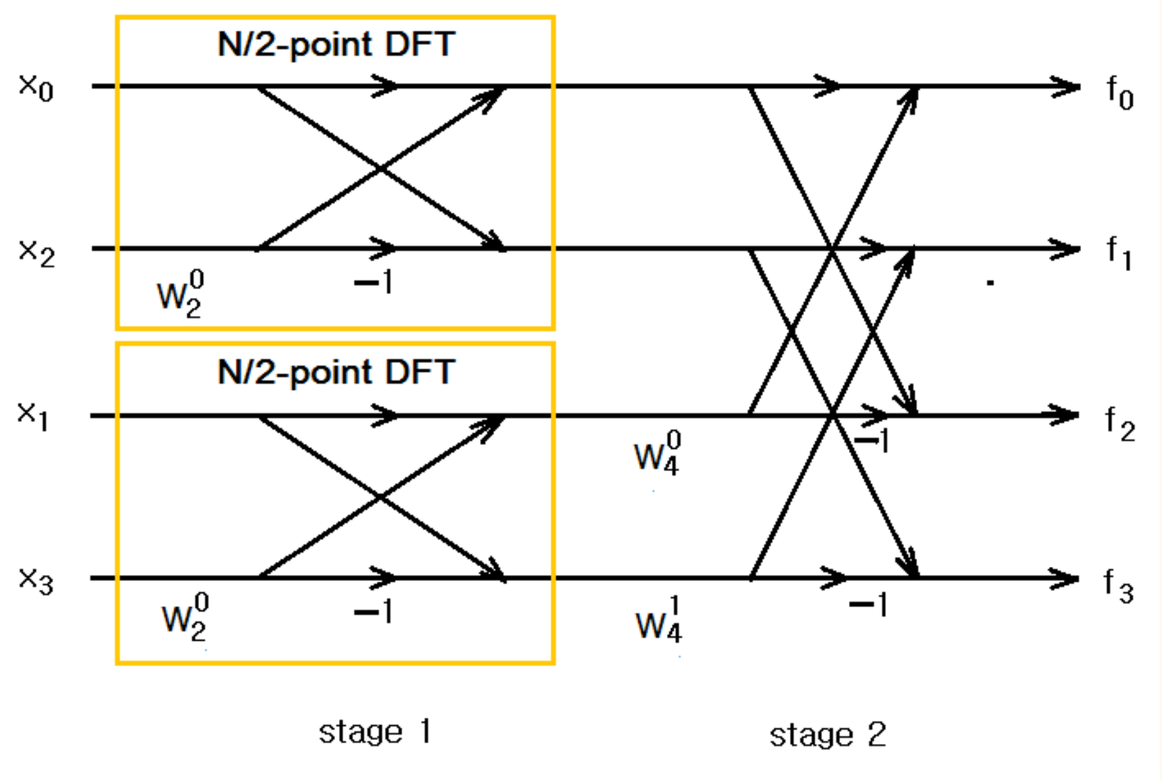
계산에서 처리되는 데이터 샘플의 수

Butterfly

두 개의 입력 데이터를 연산 후 두 개의 출력 데이터를 생성
FFT 기본적인 계산 단위

IFFT

FFT의 역변환, **주파수 신호 → 시간 영역 신호**
FFT 신호: 주파수 분해 분석
IFFT 신호: 주파수 합성 복원



01 Overview

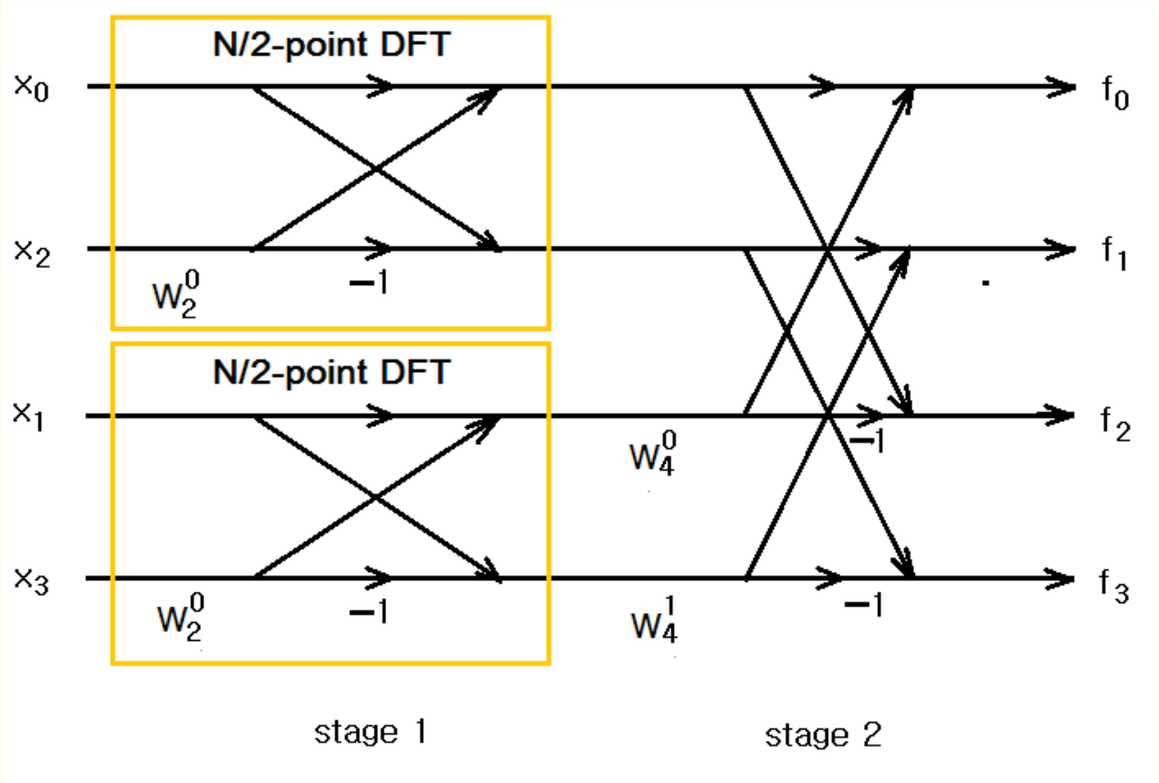
주제 소개

Twiddle Factor

FFT 데이터의 복소수 곱셈기
Radix-2 버터플라이의 단순성과 Radix-4 유사한 곱셈 효율성을 유지

Fixed Point / Floating Point

항목	고정 소수점 (Fixed-point)	부동 소수점 (Floating-point)
소수점 위치	고정	유동적 (Exponent 사용)
계산 속도	빠름 (간단한 하드웨어)	느림 (복잡한 계산)
하드웨어 비용	낮음 (저렴함)	높음 (복잡한 연산기 필요)
표현 범위	제한적	아주 넓음
정밀도	고정된 자리 수	가변적
사용 예	임베디드 시스템, DSP	과학 계산, AI, 3D 그래픽



02 Floating Point Modeling

cos_in_gen.m

Test Vector(Cosine)를 만들어주는 함수를 정의하였습니다.

```
function [data_float, data_fixed] = cos_in_gen(fft_mode, num)
 N = num;

 for i=1:N
    data_float_re(i) = cos(2.0*pi*(i-1)/N);
    data_float_im(i) = 0.0;
    data_float(i) = data_float_re(i) + j*data_float_im(i);
 end
```

Modulated된 신호가 아님으로 순수 Real Number만 입력 됩니다.
따라서 512 point으로 나뉘어진 test vector를 생성합니다.

Fixed Number로 변환하기 위해서 ex) <3.6>

1. <3.6> 에서는 소수점 이하 자리가 6임으로 2^6 = 64의 Scaling Vector를 곱하게 됩니다.

2. Scale된 값을 가장 가까운 정수로 round합니다.

3.Round를 통해 Overflow 및 Saturation상황을 고려해야 합니다.

Fixed Point Real Number Test vector로 만들어주는 부분입니다.

```
for i=1:N
 if (data_float_re(i)==1.0)
  if (fft_mode==1) % FFT
   %data_fixed_re(i) = 127; % <2.7>
   data_fixed_re(i) = 63; % <3.6> % Modified on 2025/07/02 by jihan
  else % IFFT
   data_fixed_re(i) = 255; % <1.8>
   %data_fixed_re(i) = 127; % <2.7> % Modified on 2025/07/02 by jihan
  end
 else
  if (fft_mode==1) % FFT
   %data_fixed_re(i) = round(data_float_re(i)*128); % <2.7>
   data_fixed_re(i) = round(data_float_re(i)*64); % <3.6> % Modified on 2025/07/02 by jihan
  else % IFFT
   data_fixed_re(i) = round(data_float_re(i)*256); % <1.8>
   %data_fixed_re(i) = round(data_float_re(i)*128); % <2.7> % Modified on 2025/07/02 by jihan
  end
 end
```

Fixed Point Imaginary Number Test vector로 만들어주는 부분입니다.

```
if (data_float_im(i)==1.0)
 if (fft_mode==1) % FFT
  %data_fixed_im(i) = 127; % <2.7>
  data_fixed_im(i) = 63; % <3.6> % Modified on 2025/07/02 by jihan
 else % IFFT
  data_fixed_im(i) = 255; % <1.8>
  %data_fixed_im(i) = 127; % <2.7> % Modified on 2025/07/02 by jihan
 end
else
 if (fft_mode==1) % FFT
  %data_fixed_im(i) = round(data_float_im(i)*128); % <2.7>
  data_fixed_im(i) = round(data_float_im(i)*64); % <3.6> % Modified on 2025/07/02 by jihan
 else % IFFT
  data_fixed_im(i) = round(data_float_im(i)*256); % <1.8>
  %data_fixed_im(i) = 127; % <2.7> % Modified on 2025/07/02 by jihan
 end
end
```

Fixed Point Number(Real & Imaginary) Test vector 부분입니다.

```
data_fixed(i) = data_fixed_re(i) + j*data_fixed_im(i);
```

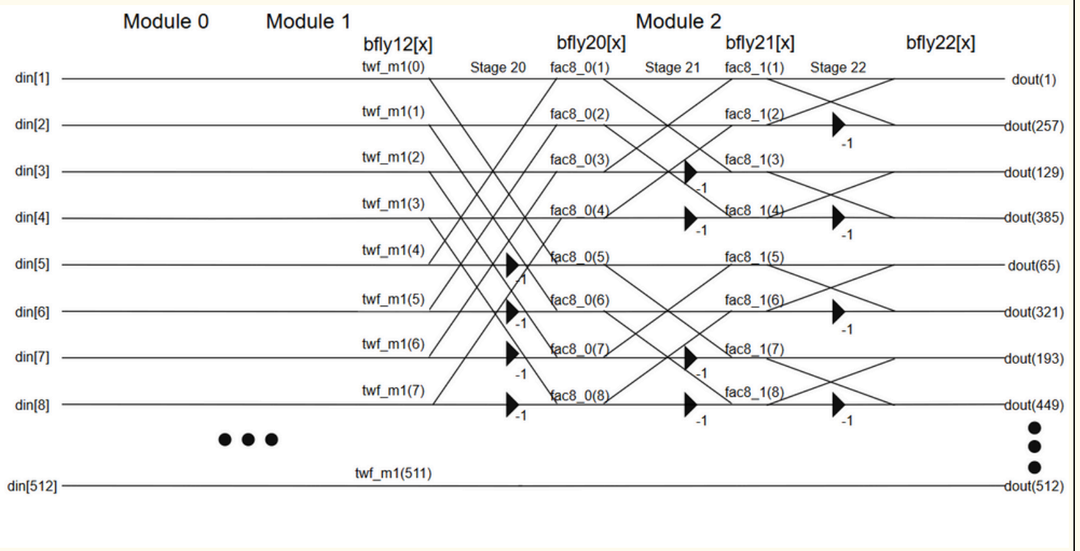
02 Floating Point Modeling

```
% Data rearrangement
K2 = [0, 4, 2, 6, 1, 5, 3, 7];

for kk=1:8
 for nn=1:8
   twf_m1((kk-1)*8+nn) = exp(-j*2*pi*(nn-1)*(K2(kk))/64);
 end
end

for kk=1:8
 for nn=1:64
   bfly12((kk-1)*64+nn) = bfly12_tmp((kk-1)*64+nn)*twf_m1(nn);
 end
end
```

twf_m1 = $W_N = e^{-j\frac{2\pi}{N}}$

bfly12_tmp: stage 12에서 더한 값 (덧셈)
bfly12 = bfly12_tmp * twf_m1 (곱셈)

fft_float.m



$$X[k] = \sum_{n=0}^{N-1} x[n] W_N^{nk},\ k = 0,1,\ldots,N-1 \text{ where } W_N = e^{-j\frac{2\pi}{N}}$$

X = dout, x = din, W = twf, N = 512

02 Floating Point Modeling

```
%-----------------------------------------------------------------------
% Module 2
%-----------------------
% step2_0
for kk=1:64
 for nn=1:4
   bfly20_tmp((kk-1)*8+nn) = bfly12((kk-1)*8+nn) + bfly12((kk-1)*8+4+nn);
   bfly20_tmp((kk-1)*8+4+nn) = bfly12((kk-1)*8+nn) - bfly12((kk-1)*8+4+nn);
 end
end
```

bfly20_tmp: stage 20에서 더한 값 저장 (덧셈)

Radix 2 를 이용한 모습

```
for kk=1:64
 for nn=1:8
   bfly20((kk-1)*8+nn) = bfly20_tmp((kk-1)*8+nn)*fac8_0(ceil(nn/2));
 end
end
```

bfly20 = bfly20_tmp * fac8_0 (곱셈)

```
% step2_1
for kk=1:128
 for nn=1:2
   bfly21_tmp((kk-1)*4+nn) = bfly20((kk-1)*4+nn) + bfly20((kk-1)*4+2+nn);
   bfly21_tmp((kk-1)*4+2+nn) = bfly20((kk-1)*4+nn) - bfly20((kk-1)*4+2+nn);
 end
end
```

Radix 2 를 이용한 모습

bfly21_tmp: stage 21에서 더한 값 (덧셈)

```
for kk=1:64
 for nn=1:8
   bfly21((kk-1)*8+nn) = bfly21_tmp((kk-1)*8+nn)*fac8_1(nn);
 end
end
```

bfly21 = bfly21_tmp * fac8_1 (곱셈)

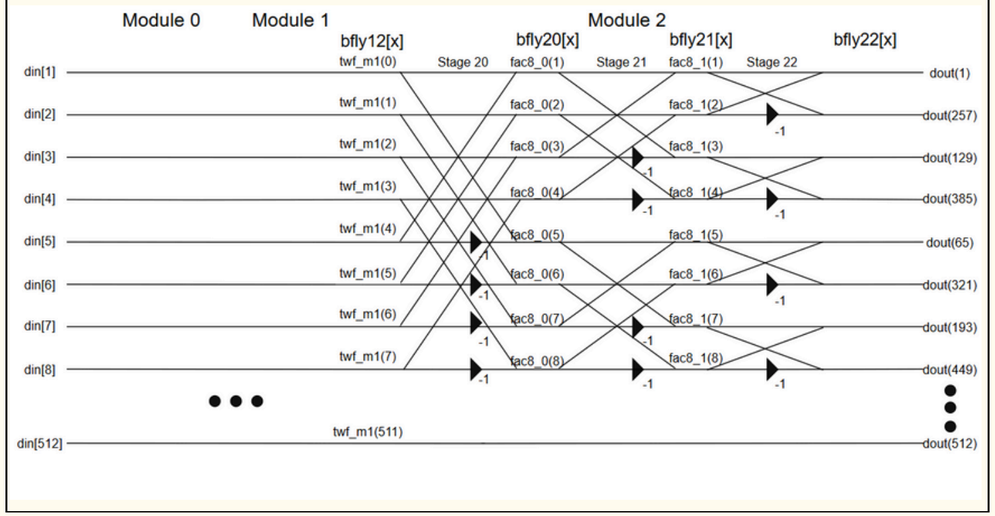
```
% step2_2
for kk=1:256
   bfly22_tmp((kk-1)*2+1) = bfly21((kk-1)*2+1) + bfly21((kk-1)*2+2);
   bfly22_tmp((kk-1)*2+2) = bfly21((kk-1)*2+1) - bfly21((kk-1)*2+2);
end
```

Radix 2 를 이용한 모습

bfly22_tmp: stage 22에서 더한 값 (덧셈)

```
bfly22 = bfly22_tmp;
```

fft_float.m



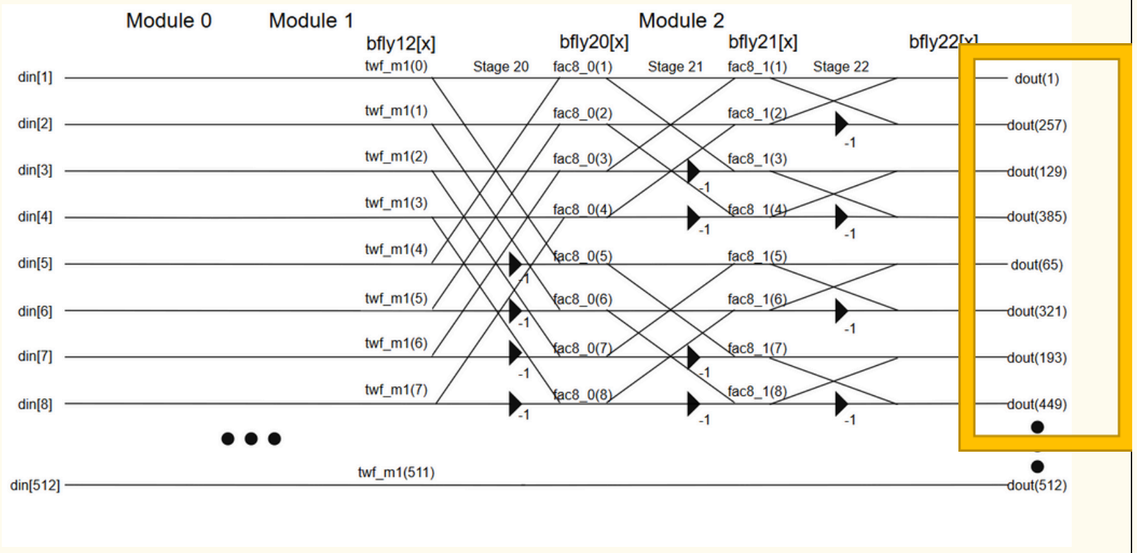
위치	설명
stage12과 stage20사이	bfly20_tmp에 bfly12 더한 값 저장 bfly20에 bfly20_tmp*fac8_0 저장
stage20과 stage21사이	bfly21_tmp에 bfly20 더한 값 저장 bfly21에 bfly21_tmp*fac8_0 저장
stage21과 stage22사이	bfly22_tmp에 bfly21 더한 값 저장 bfly22에 bfly22_tmp*fac8_0 저장

02 Floating Point Modeling

```
for jj=1:512
   kk = bitget(jj-1,9)*1 + bitget(jj-1,8)*2 + bitget(jj-1,7)*4 + bitget(jj-1,6)*8 + bitget(jj-1,5)*16 + bitget(jj-1,4)*32 + bitget(jj-1,3)*64 + bitget(jj-1,2)*128 + bitget(jj-1,1)*256;
   dout(kk+1) = bfly22(jj); % With reorder
   fprintf(fp, 'jj=%d, kk=%d, dout(%d)=%f+j%f\n',jj, kk,(kk+1),real(dout(kk+1)),imag(dout(kk+1)));
end
```

- jj: 1부터 512까지 반복
- jj - 1 -> 9비트 이진수
- 그 비트를 반대로 해석해서 정수로 만든 것이 kk
- dout(kk+1) = bfly22(jj)
- dout → 최종 출력

fft_float.m



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

02 Floating Point Modeling

test_fft_float_stu.m

아래는 test vector 함수와 FFT함수를 연결하여 실행하는 코드입니다.

```
% Test fft function (fft_float)
% Added on 2025/07/02 by jihan
 fft_mode = 1; % '0': ifft, '1': fft
 N = 512;

 [cos_float, cos_fixed] = cos_in_gen(fft_mode, N);

 [fft_out, module2_out] = fft_float(1, cos_float); % Floating-point fft (fft) : Cosine
```

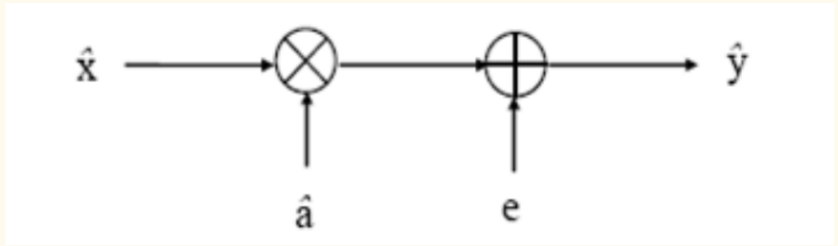
1. 현재 FFT모드는 1로 “FFT 모드“로 동작하고 있습니다.

2. N = 512개의 Point로 설정 되어있습니다.

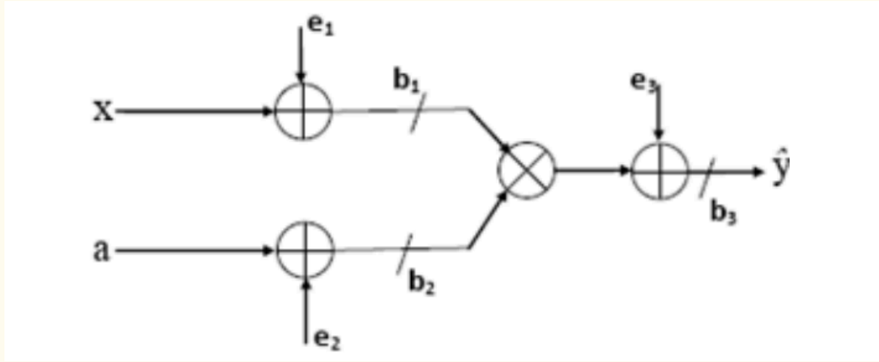
3. cos_in_gen 함수는 float값과 fixed값을 모두 return 하고 있습니다.

4. fft_float함수는 1(FFT_MODE) 과 float값을 입력 받고 fft_out은 bit reversal 적용된 값을, module2_out은 bit reversal을 하기전의 값을 return합니다.

03 Floating Point → Fixed Point 문제점



- 이론상 출력은 $\hat{y} = \hat{x} \times \hat{a}$
- 실제 고정소수점 하드웨어에서는 정밀도가 부족해 양자화 오차 e가 생김
- 따라서 실제 출력은 $\hat{y} = \hat{x} \times \hat{a} + e$로 모델링
- 이 오차 e는 결과 정확도에 누적되어 큰 손실을 유발
- FFT는 반복적으로 곱셈을 수행하므로 오차도 누적됨



- 이론적 계산
  - $y\hat{} = x \times a$
- 실제 계산 (고정소수점):
  - $y\hat{} = (x+e1)(a+e2)+e3 \approx xa+ae1+xe2+e3$
- 오차 발생 위치:
  - e_1, e_2: 입력과 계수 양자화 오차
  - e_3: 출력 양자화 오차
  - (고차항 $e_1 \cdot e_2$도 누적 가능)
- 결과:
  - FFT 연산을 거칠수록 오차가 누적됨
  - 정확도 저하, SNQR 포화 현상 발생 가능

04 입력 스케일링 (Input Scaling)

- 핵심 아이디어
  - FFT의 전체 스케일링 1/N을 각 스테이지에서 1/2씩 분산 적용
  - 매 단계에서 신호를 1/2씩 나눠 오차 누적을 줄임
  - 이로 인해 QE가 기하급수적으로 줄어드는 효과

$$\sigma_q^2 = 8\sigma_n^2 \left(1 - \left(\frac{1}{2}\right)^v\right)$$

- σq2\sigma_q^2σq2: 전체 양자화 오차 분산
- σn2\sigma_n^2σn2: 각 스테이지에서 발생하는 오차 분산
- v: FFT 스테이지 수 (ex. N=1024이면 v=log2(1024)=10v = \log_2(1024) = 10v=log2(1024)=10)

“

- 오버플로우 위험 감소
  - 각 FFT 스테이지에서 신호 크기를 줄여 정수 범위 초과 방지
- 양자화 오차 누적 최소화
  - 스테이지별 오차를 분산시켜 전체 오차 크기 감소
- SQNR 향상 (Signal-to-Quantization-Noise Ratio)
  - 정밀도 유지 및 출력 신뢰도 향상

06 지시형 스케일링 (Directive Scaling)

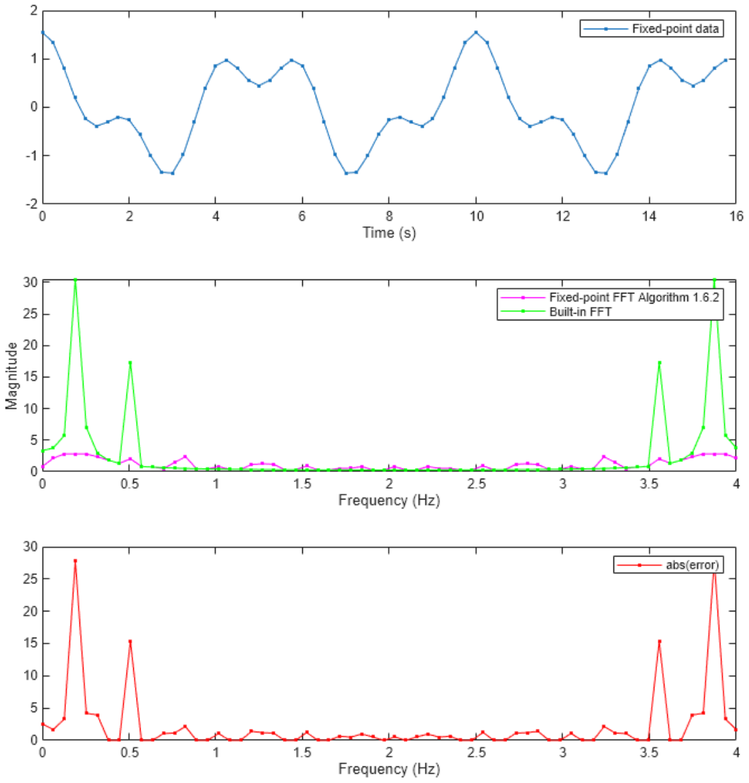
🔢 수식 기반 설명

- 평균 신호 전력 증가율
  - FFT에서 각 스테이지는 평균 신호 전력을 약 2\sqrt{2}2 배 증가시킴 이를 수학적으로 정리하면
  - E[∣y[i]∣2]=2⋅E[∣x[i]∣2]
  - 즉, 스테이지를 지날수록 오버플로우 위험이 커짐

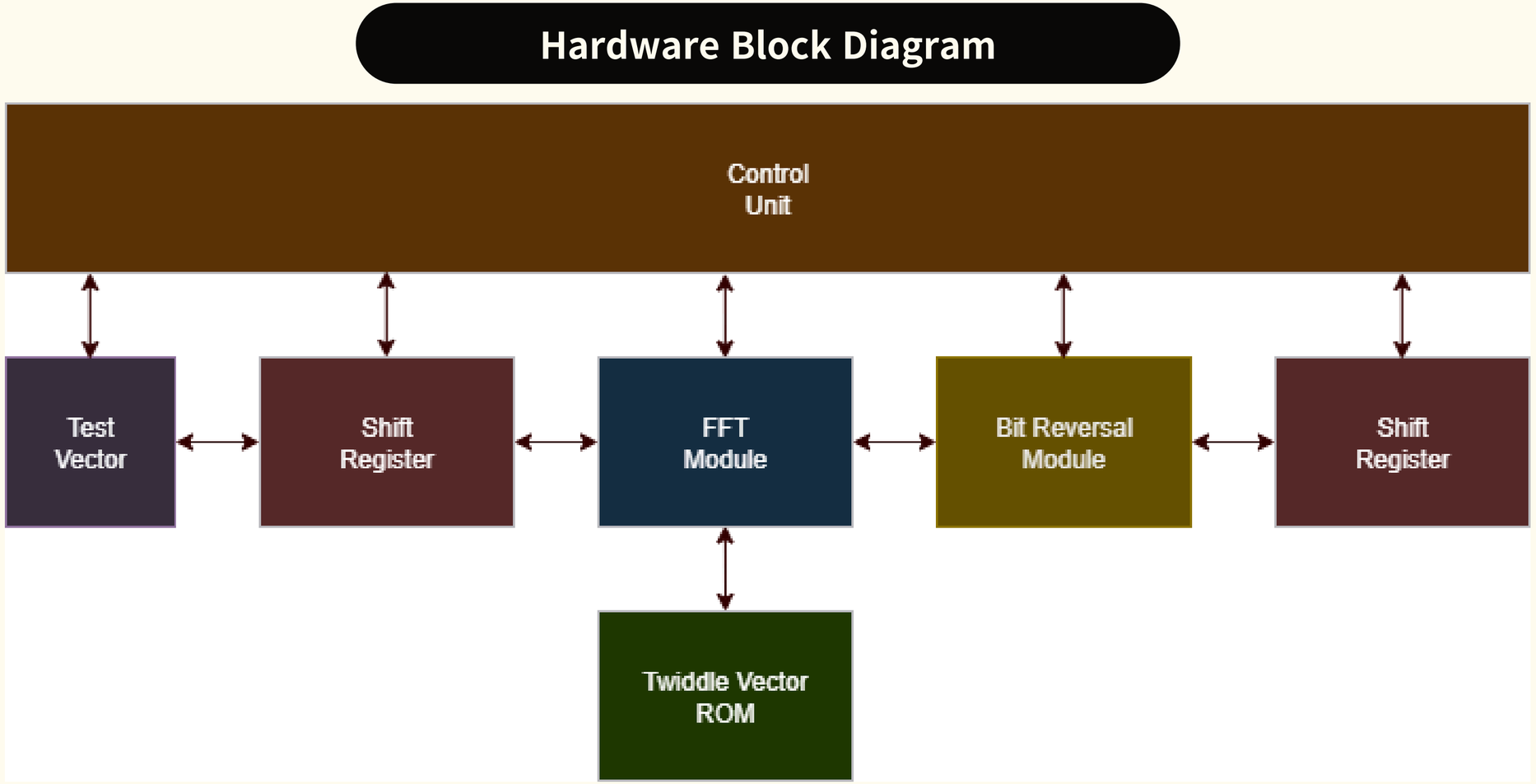
- 스테이지마다 오버플로우 여부를 실시간으로 판단하는 대신, 미리 설계한 스케일링 패턴을 반복 적용
- 조건부 스케일링(CS)는 오버플로우 발생 시에만 1/2 shift 수행 → 런타임 비용 존재
- 지시형 스케일링(DS)는 트라이얼 기반 시뮬레이션 결과를 바탕으로, 오버플로우가 자주 발생하는 스테이지를 사전에 결정
- → 해당 스테이지에만 스케일링 비트 1을 할당하여 사용 (e.g. 1101010100)

06 지시형 스케일링 (Directive Scaling)

- 상단 (시간 영역 입력)
  - Fixed-point data는 고정소수점으로 양자화된 입력 신호를 나타냅니다.
- 중단 (주파수 영역 FFT 결과 비교)
  - 보라색 곡선: 지시형 스케일링을 적용한 고정소수점 FFT 결과
  - 초록색 곡선: IEEE-754 기반의 부동소수점 FFT 결과 (참조 기준)
  - → 두 결과가 거의 일치 → 정확도가 매우 높음
- 하단 (절대 오차)
  - abs(error) 그래프에서 대부분의 주파수에서 오차가 거의 0에 수렴
  - 피크가 있는 부분도 있음 (ex. 고주파), 하지만 전체적으로 매우 안정적

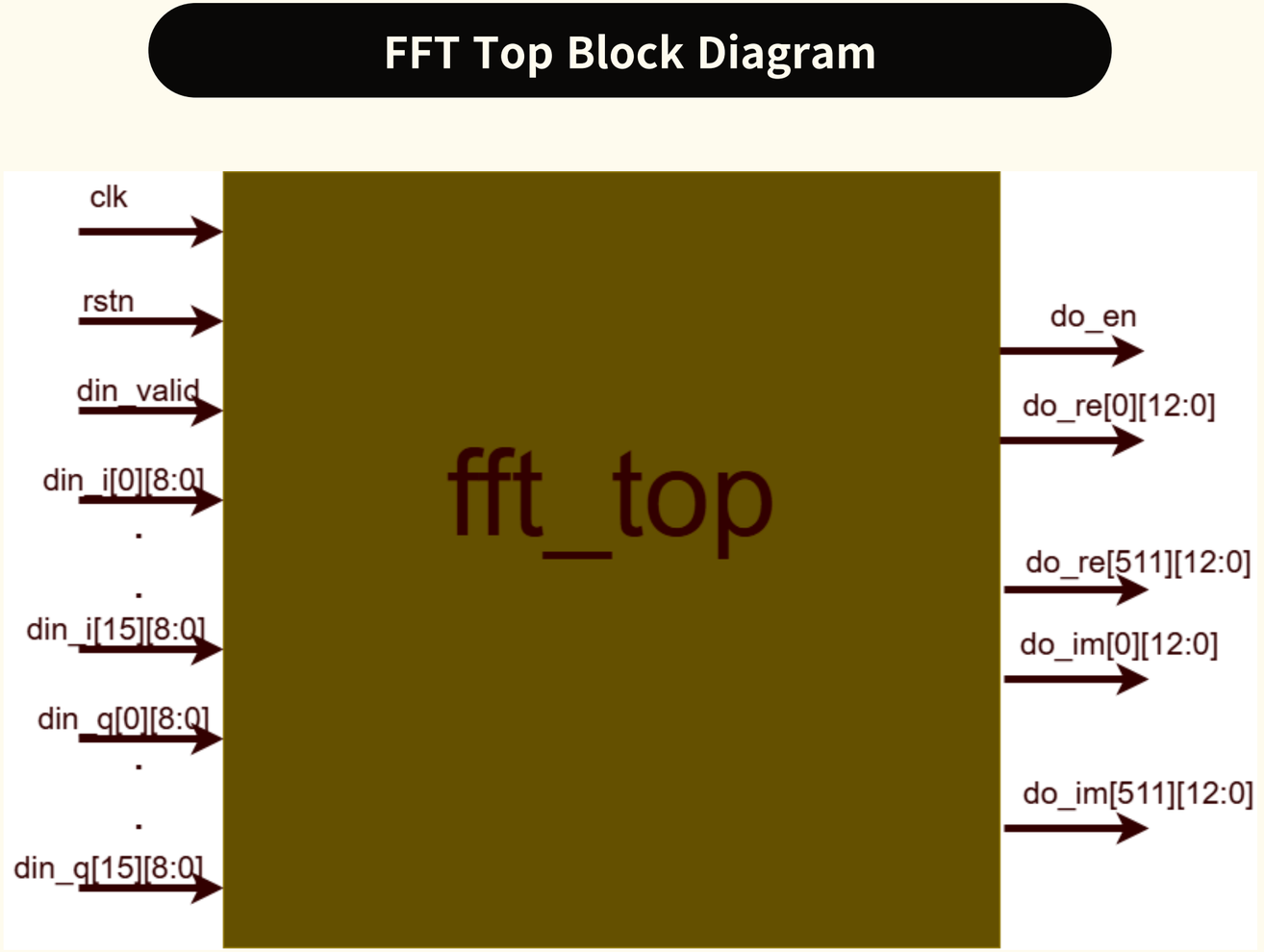


07 Hardware Block Diagram



System Block Diagram

08 FFT Top Block Diagram



fft_top Block Diagram

AI 시스템반도체설계 2기 1조

Q & A

감사합니다