RADIX-22 DIF FFT 설계

고속 디지털 신호처리를 위한 병렬 FFT 아키텍처

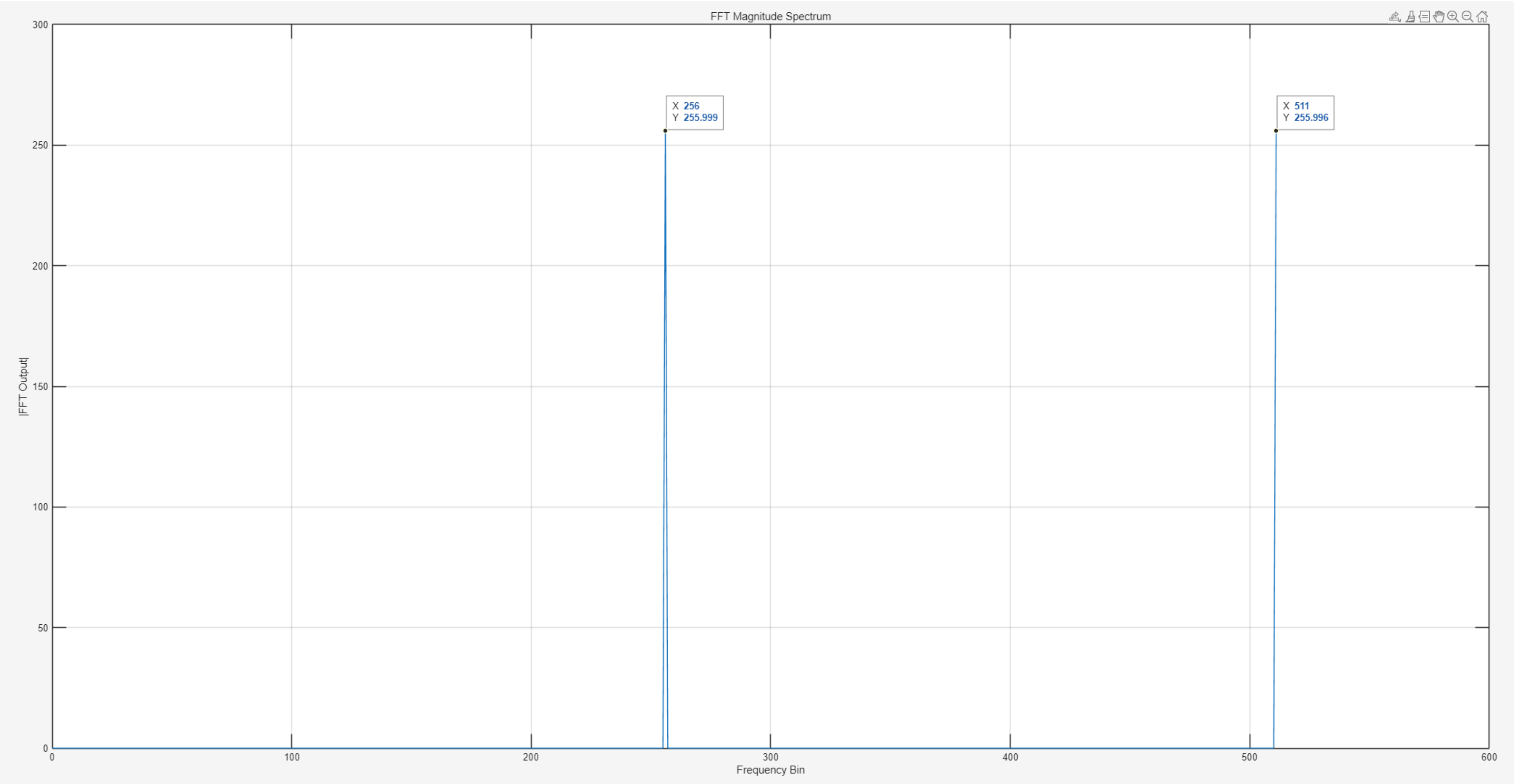
TEAM11 – 정민교 엄찬하 신상학 임재홍

대한상공회의소
서울기술교육센터

- 목차

1. Matlab 결과
2. Module Diagram
3. 논문 수식 설명

1. Matlab 결과

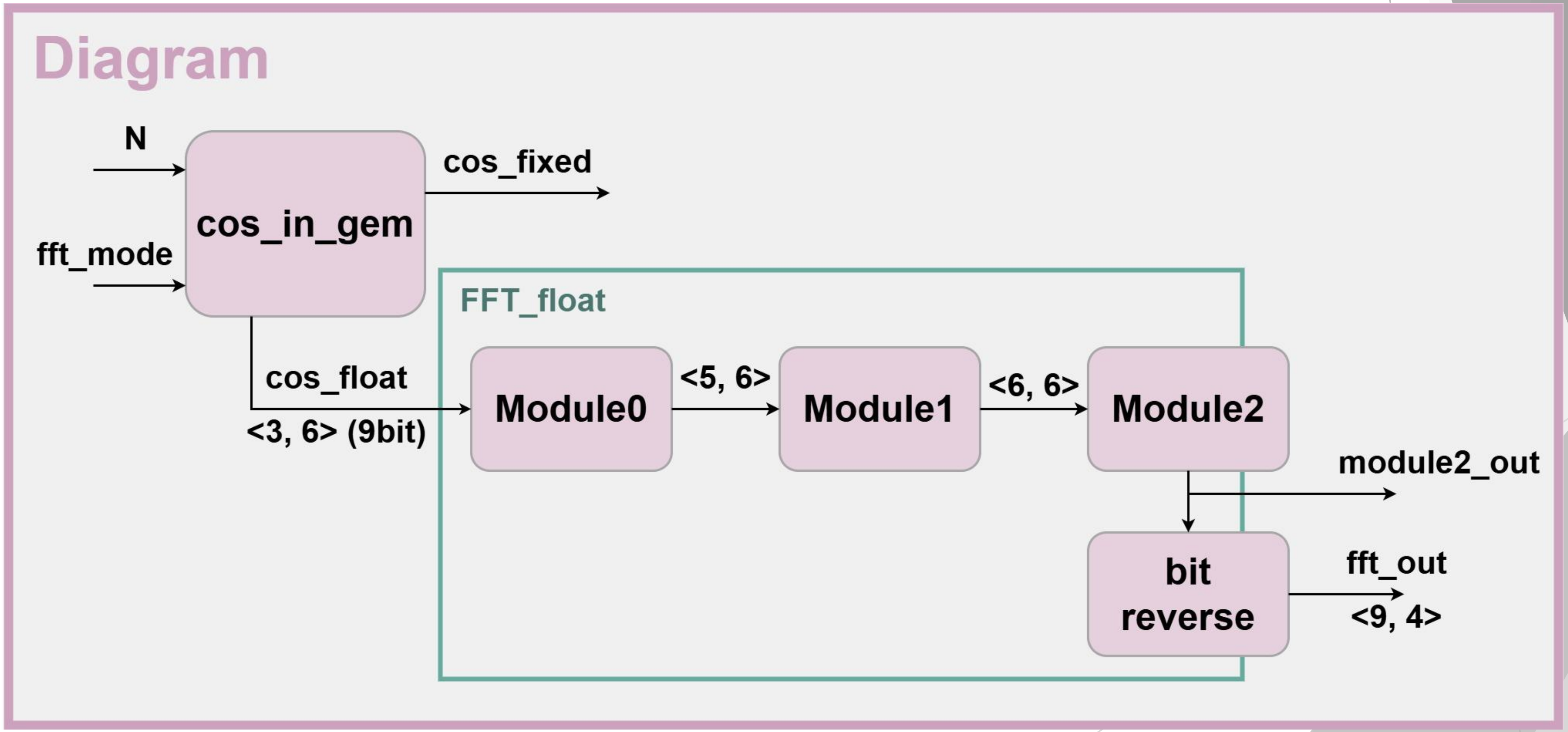


$$\cos(2\pi f_0 n/N) = \frac{1}{2}\left(e^{j2\pi f_0 n/N} + e^{-j2\pi f_0 n/N}\right)$$

-> 중심 주파수 : N/2 -> bin 256 = 256 / bin 511 = 256

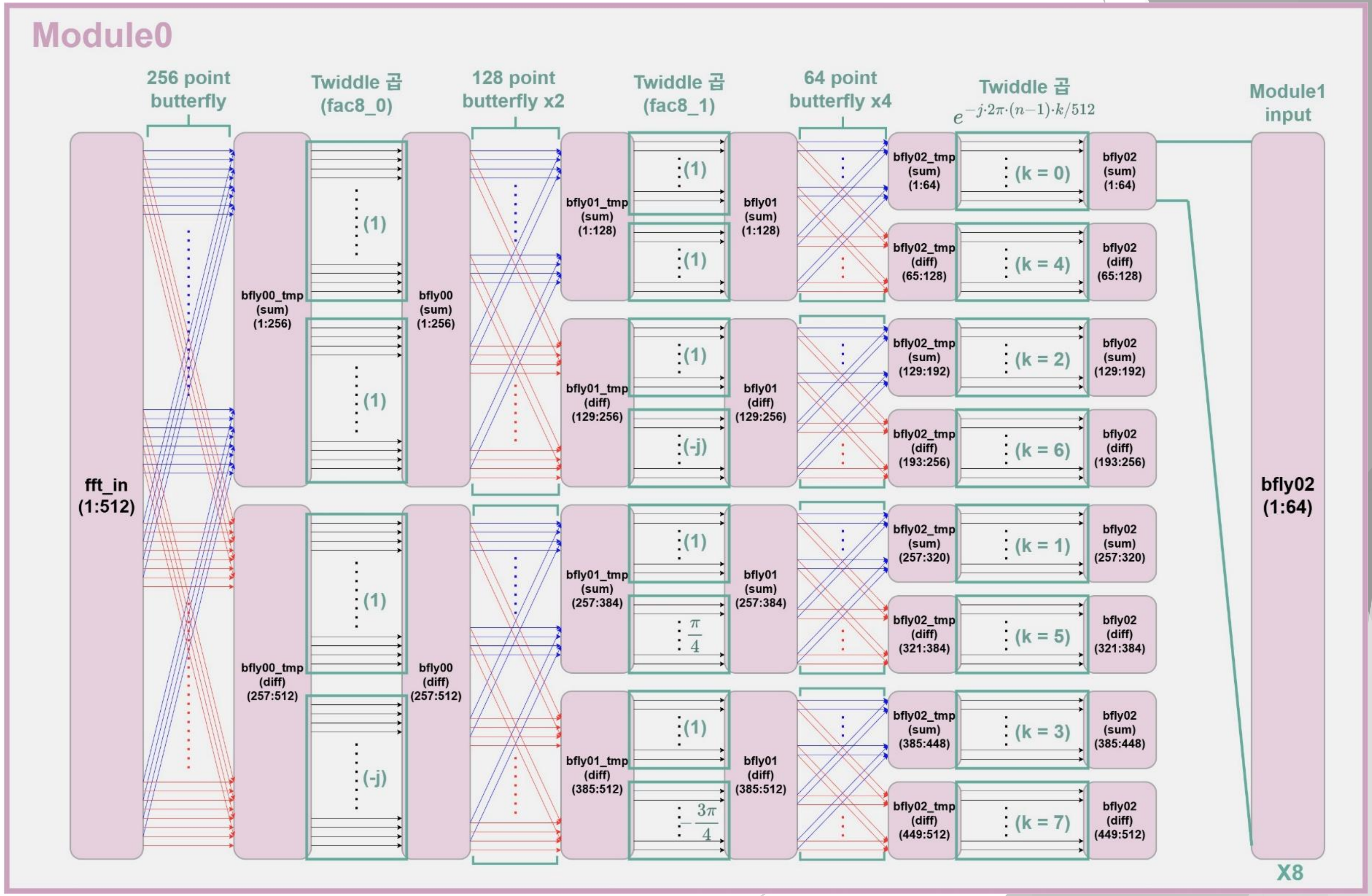
2. 설계 구조

- Module Diagram



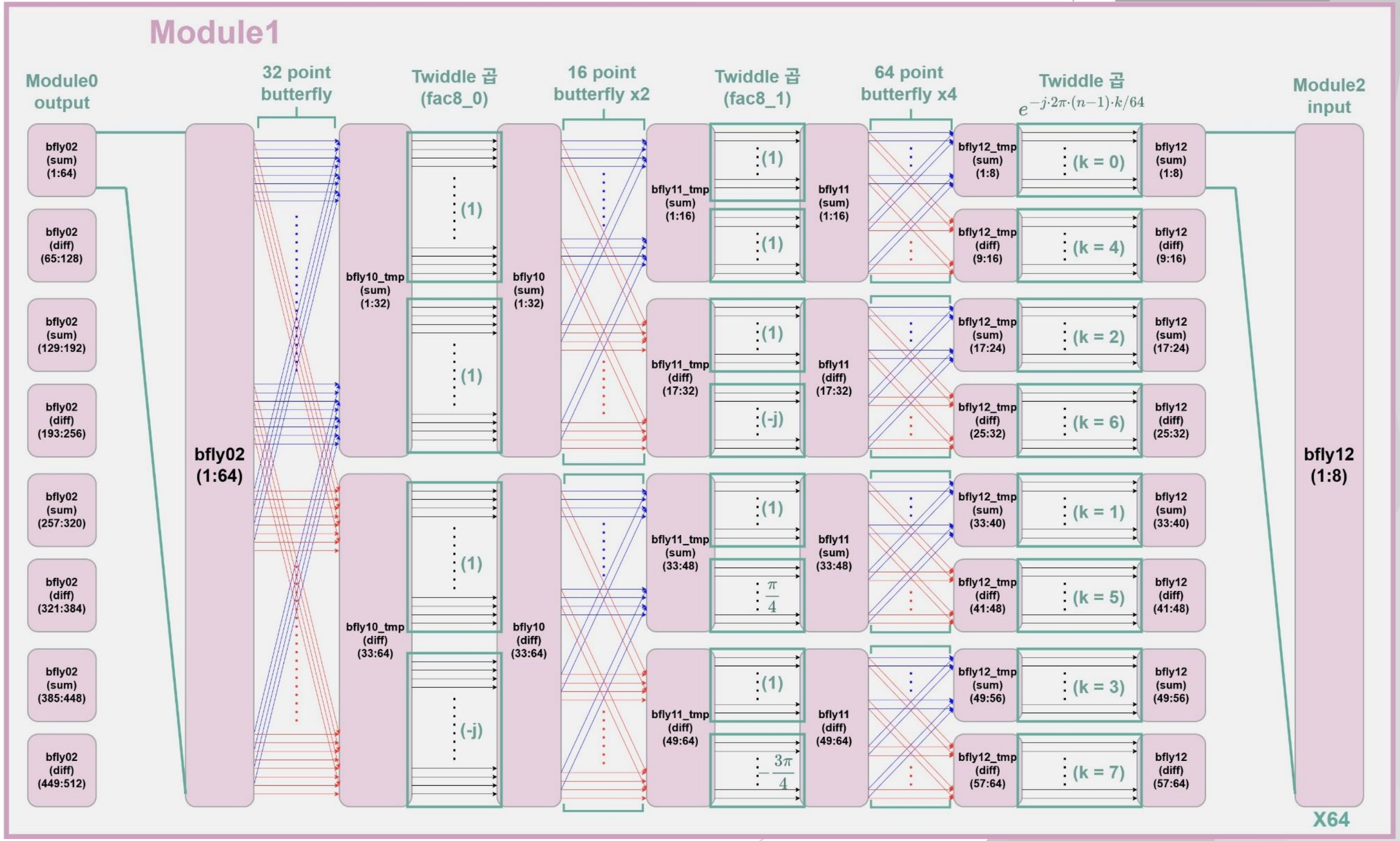
2. 설계 구조

- Module Diagram
  - Module0



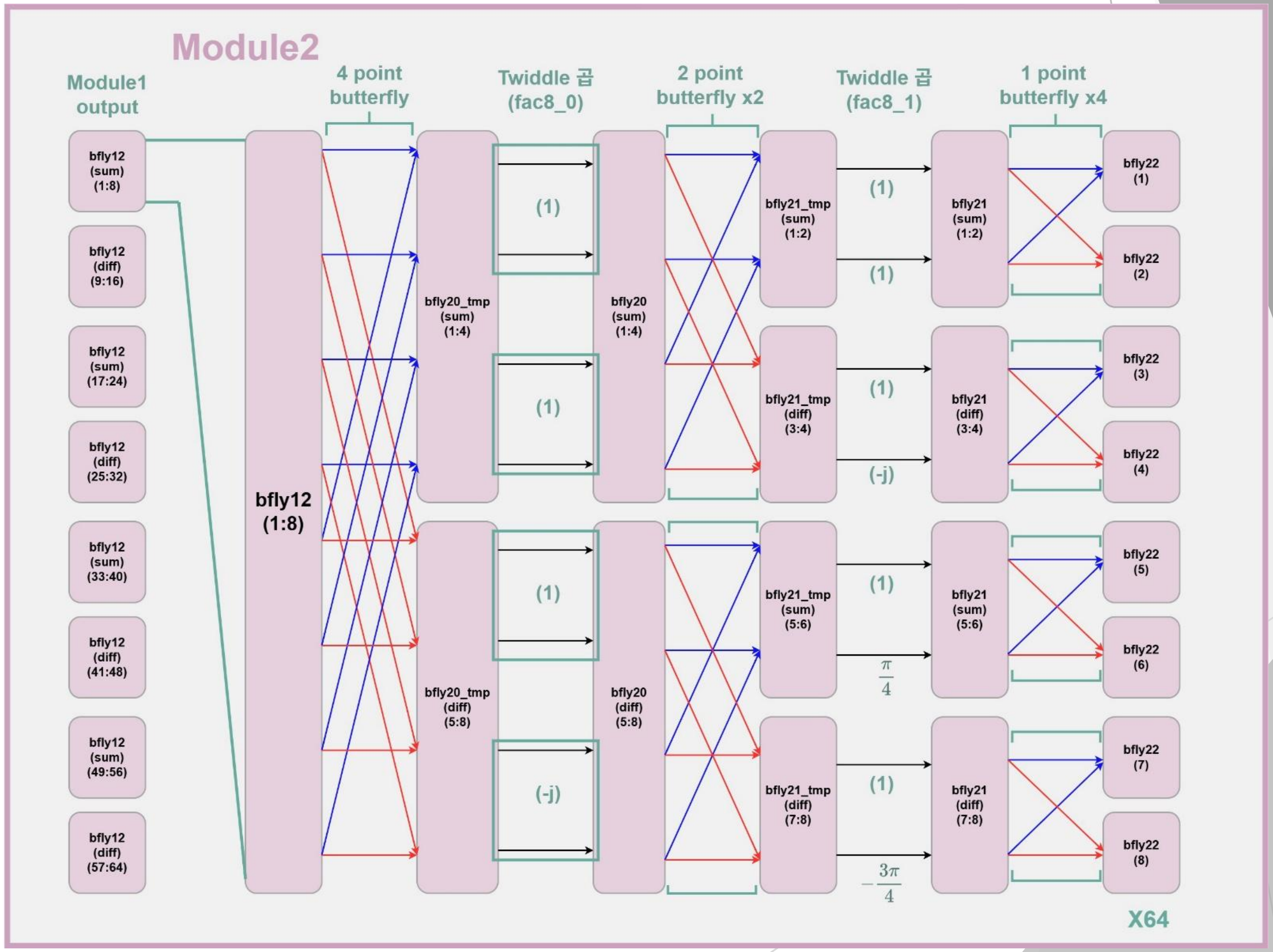
2. 설계 구조

- **Module Diagram**
  - **Module1**



2. 설계 구조

- **Module Diagram**
  - **Module2**



Twiddle Factor

1. DFT 정의식:

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

2. 짝수/홀수로 나눠서 전개:

$$= \sum_{m=0}^{N/2-1} x(2m) W_N^{2mk} + \sum_{m=0}^{N/2-1} x(2m+1) W_N^{(2m+1)k}$$

3. 공통 항 정리:

$$= \sum_{m=0}^{N/2-1} \left[x(2m) W_{N/2}^{mk} + x(2m+1) W_{N/2}^{mk} W_N^{k} \right]$$

$$= \sum_{m=0}^{N/2-1} W_{N/2}^{mk} \cdot \left[x(2m) + x(2m+1) W_N^{k} \right]$$

(여기서 $\alpha = x(2m) + x(2m+1) W_N^k$)

4. 요약 정리된 표현:

$$X_{\text{even}}(k) = \alpha, \quad X_{\text{odd}}(k) = \alpha W_N^k$$

$$X(k) = \alpha + \alpha W_N^k$$

Twiddle Factor

$$W_N^k = e^{-j\frac{2\pi k}{N}}$$ → Twiddle Factor

$$X(k) = \sum_{n=0}^{511} x(n) \cdot e^{-j2\pi kn/512}$$

$$X(k) = \sum_{r=0}^{R-1} \left[\sum_{s=0}^{S-1} x(sR + r) \cdot W_S^{sk} \right] \cdot W_N^{rk}$$

Bit Reverse (K)

원래 인덱스 (10진)	원래 인덱스 (2진)	비트 리버스 (2진)	리버스된 인덱스 (10진)
0	000	000	0
1	001	100	4
2	010	010	2
3	011	110	6
4	100	001	1
5	101	101	5
6	110	011	3
7	111	111	7

Factor 정리

이름	길이	적용되는 위치	설명
fac8_0	4	step0_0, step1_0, step2_0	N=4 FFT 기준 twiddle 상수
fac8_1	8	step0_1, step1_1, step2_1	N=8 FFT 기준 twiddle 상수
twf_m0	512	step0_2	MSB 기준 `K3` 정렬용 위상곱
twf_m1	64	step1_2	MSB 기준 `K2` 정렬용 위상곱

Twiddle Factor

따라서 홀수는 Fac8_1,2를 진행하며 의미 없는 1이 곱해지며, 짝수는 의미 있는 위상 변화가 곱해진다.

fac8_0 = [1, 1, 1, −j];
fac8_1 = [1, 1, 1, −j, 1, 0.7071−0.7071j, 1, −0.7071−0.7071j];

$$n = \left\langle \frac{N}{2}n_1 + \frac{N}{4}n_2 + n_3 \right\rangle_N \quad \text{or} \quad n = n_3 \cdot 4 + n_2 \cdot 2 + n_1$$

$$k = \langle k_1 + 2k_2 + 4k_3 \rangle_N$$

단계	변수	설명	비트위치
Stage1	N3, k3	가장 상위 비트	비트 8~6
Stage2	N2, k2	중간 비트	비트 5~3
Stage3	N1, k1	하위 비트	비트 2~0

Flow_0

Step	역할	입력 크기	출력 크기	Twiddle factor 적용	비고
step0_0	512 → 256×2로 홀짝 분할 (radix-4)	512	512	`fac8_0(ceil(nn/128))` (길이 4)	N=4 FFT, MSB 초기 정렬용
step0_1	256×2 → 128×4 구조로 재조합	512	512	`fac8_1(ceil(nn/64))` (길이 8)	N=8 FFT 중간 구조 설계
step0_2	128×4 → 64×8, 마지막 그룹화	512	512	`twf_m0(nn)` (K3 기반 위상 곱)	**K3 = MSB 정렬 적용**

Flow_1

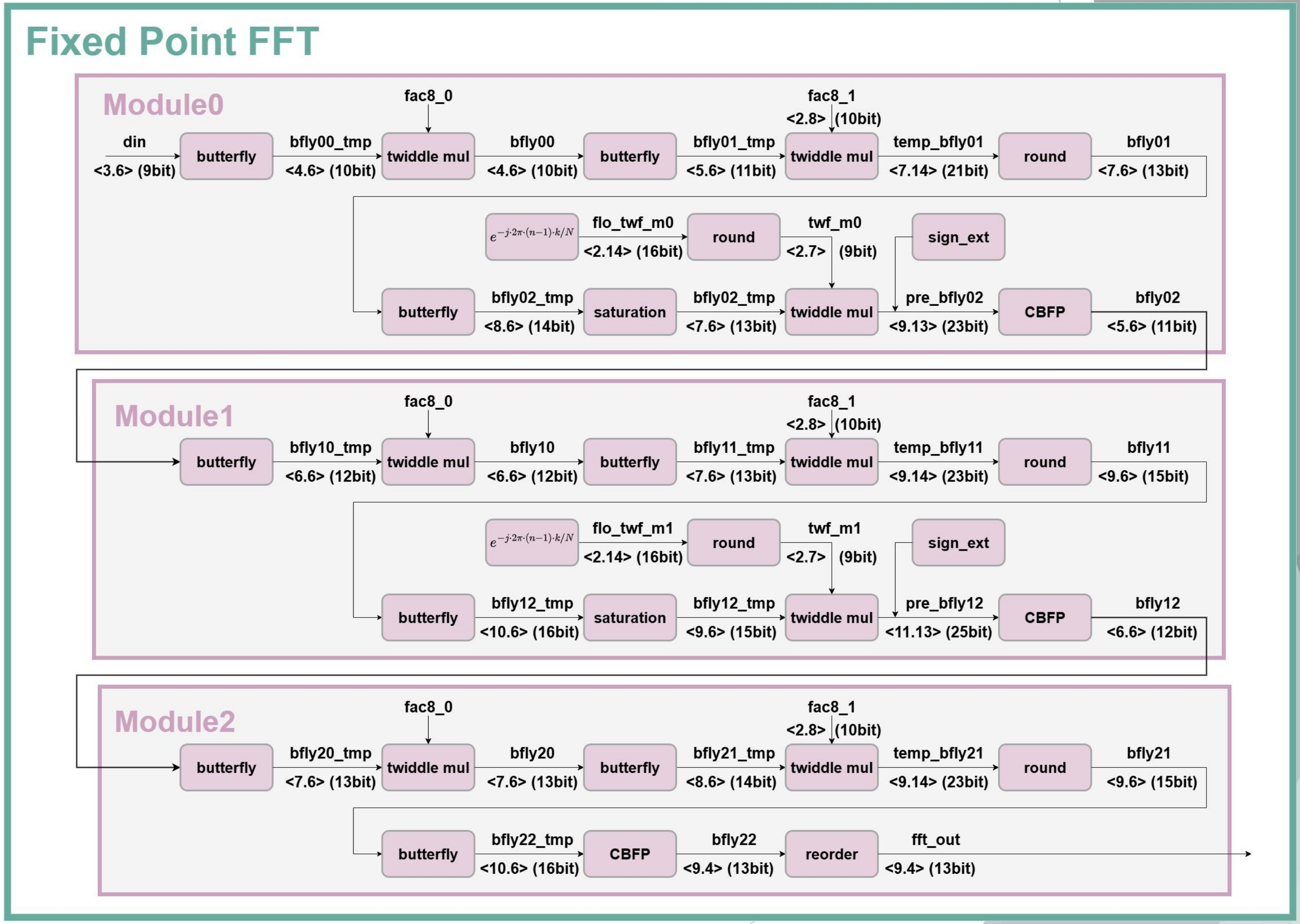
Step	역할	입력 크기	출력 크기	Twiddle Factor 적용	비고
step1_0	64점 그룹 → 32점 butterfly	512 = 8×64	512 = 8×64	`fac8_0(ceil(nn/16))` (길이 4)	
step1_1	32점 그룹 → 16점 butterfly	512 = 16×32	512 = 16×32	`fac8_1(ceil(nn/8))` (길이 8)	
step1_2	16점 그룹 → 8점 butterfly	512 = 32×16	512 = 32×16	`twf_m1(nn)` (K2 기반 위상 곱)	**K2 = MSB 정렬 적용**

Flow_2

Step	역할	입력 크기	출력 크기	Twiddle Factor 적용	비고
step2_0	8점 그룹 → 4점 butterfly	512 = 64×8	512 = 64×8	`fac8_0(ceil(nn/2))` (길이 4)	
step2_1	4점 그룹 → 2점 butterfly	512 = 128×4	512 = 128×4	`fac8_1(nn)` (길이 8, 고정값 적용)	
step2_2	2점 butterfly (최종 단계)	512 = 256×2	512	없음	최종 결과 완성됨

2. fixed point FFT

- **Module Diagram**



3. CBFP

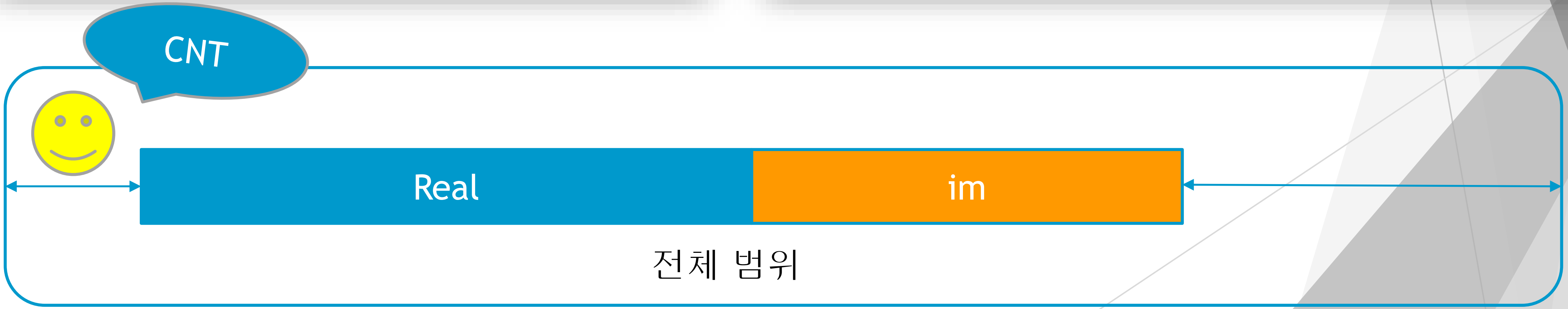
- **Real / Imgn**

```matlab
% CBFP(Convergent Block Floating Point) stage0
cnt1_re = zeros(1,8);
cnt1_im = zeros(1,8);
for ii=1:8
 for jj=1:64
   tmp1_re=mag_detect(real(pre_bfly02(64*(ii-1)+jj)),23);
   tmp1_im=mag_detect(imag(pre_bfly02(64*(ii-1)+jj)),23);
   % 블록 내에서 실수부/허수부 시프트 카운트의 최소값을 계속 갱신
   % (가장 큰 절댓값에 해당하는 가장 작은 시프트 카운트를 찾는 과정)
   temp1_re=min_detect(jj,tmp1_re,cnt1_re(ii));
   temp1_im=min_detect(jj,tmp1_im,cnt1_im(ii));
   cnt1_re(ii)=temp1_re;
   cnt1_im(ii)=temp1_im;
 end
end
```

```matlab
% 각 블록에 대해 실수부와 허수부의 시프트 카운트 중 더 작은 값으로 통일
for ii=1:8
 if (cnt1_re(ii)<=cnt1_im(ii))        % 실수부 시프트 카운트가 더 작거나 같으면 그대로 유지
   cnt1_re(ii)=cnt1_re(ii);
 else
   cnt1_re(ii)=cnt1_im(ii);           % 허수부 시프트 카운트가 더 작으면 실수부 시프트 카운트를 허수부로 맞춤 (더 보수적으로)
 end
end

for ii=1:8
 if (cnt1_im(ii)<=cnt1_re(ii))        % 허수부 시프트 카운트가 더 작거나 같으면 그대로 유지
   cnt1_im(ii)=cnt1_im(ii);
 else
   cnt1_im(ii)=cnt1_re(ii);           % 실수부 시프트 카운트가 더 작으면 허수부 시프트 카운트를 실수부로 맞춤 (더 보수적으로)
 end
end

for ii=1:8
 for jj=1:64
   % 결정된 블록별 시프트 카운트를 전체 512개 데이터에 대해 복사
   index1_re(64*(ii-1)+jj)=cnt1_re(ii);
   index1_im(64*(ii-1)+jj)=cnt1_im(ii);
 end
end
```

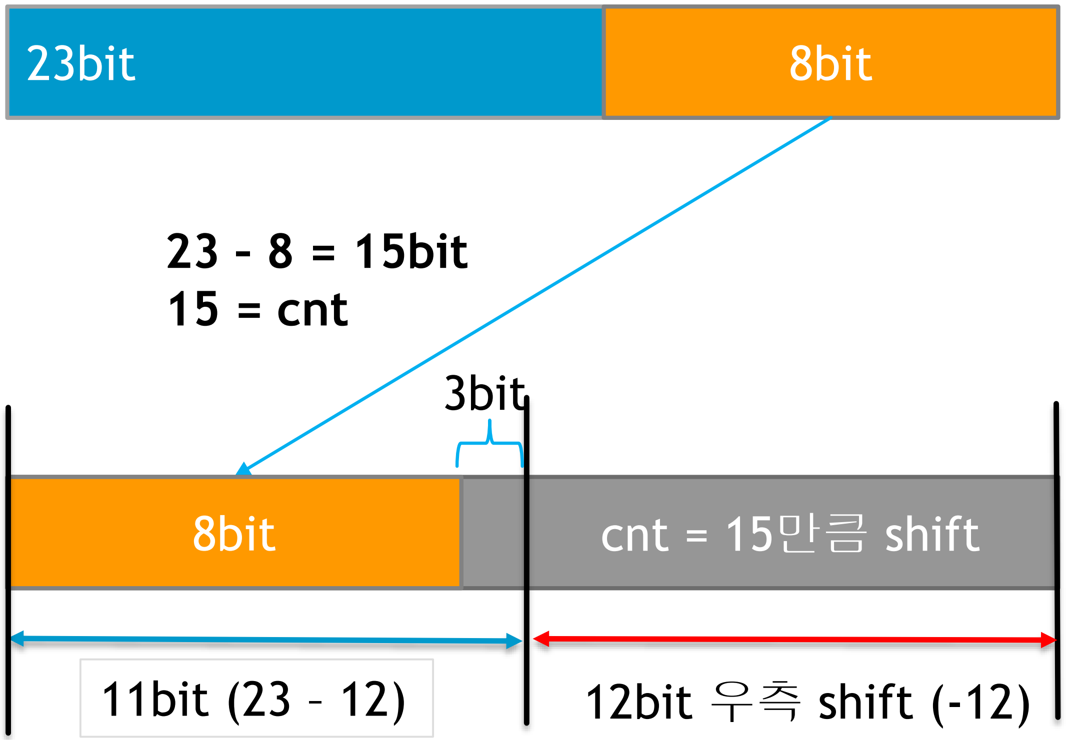


3. CBFP

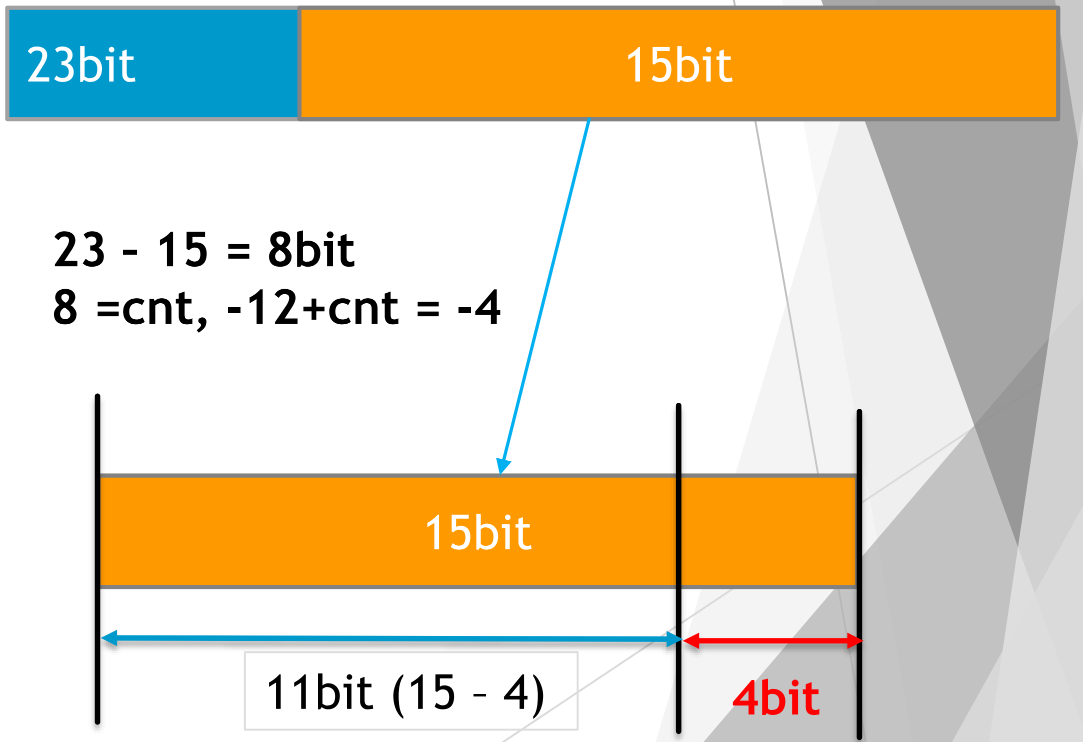
- **bit shift**

```
for ii=1:8
 for jj=1:64
  if (cnt1_re(ii)>12)  % 경우 1: 시프트 가능 비트 수(cnt1_re(ii))가 12보다 큰 경우, 여유 공간 존재
   re_bfly02(64*(ii-1)+jj)=bitshift(bitshift(real(pre_bfly02(64*(ii-1)+jj)),cnt1_re(ii), 'int32'),-12, 'int32');
   % 23중 12보다 작으면 ex)8bit면, cnt_re = 23 - 8 > 12
   % 왼쪽으로 cnt_re=15bit 하여 23bit 만듬 그 후에 12bit를 빼서 다시 11bit으로 만듬
   re_bfly02(64*(ii-1)+jj)=bitshift(real(pre_bfly02(64*(ii-1)+jj)),(-12+cnt1_re(ii)), 'int32');
   % 23중 12보다 크면 ex)15bit이면 cnt_re = 23 - 15 < 12
   % -12+cnt_re = -4 따라서 음수 4bit를 오른쪽 쉬프트 하여 짤라내어 11bit로 만듬(범위를 안전하게 줄임)
  end
 end
end
```

Cnt1_re>12

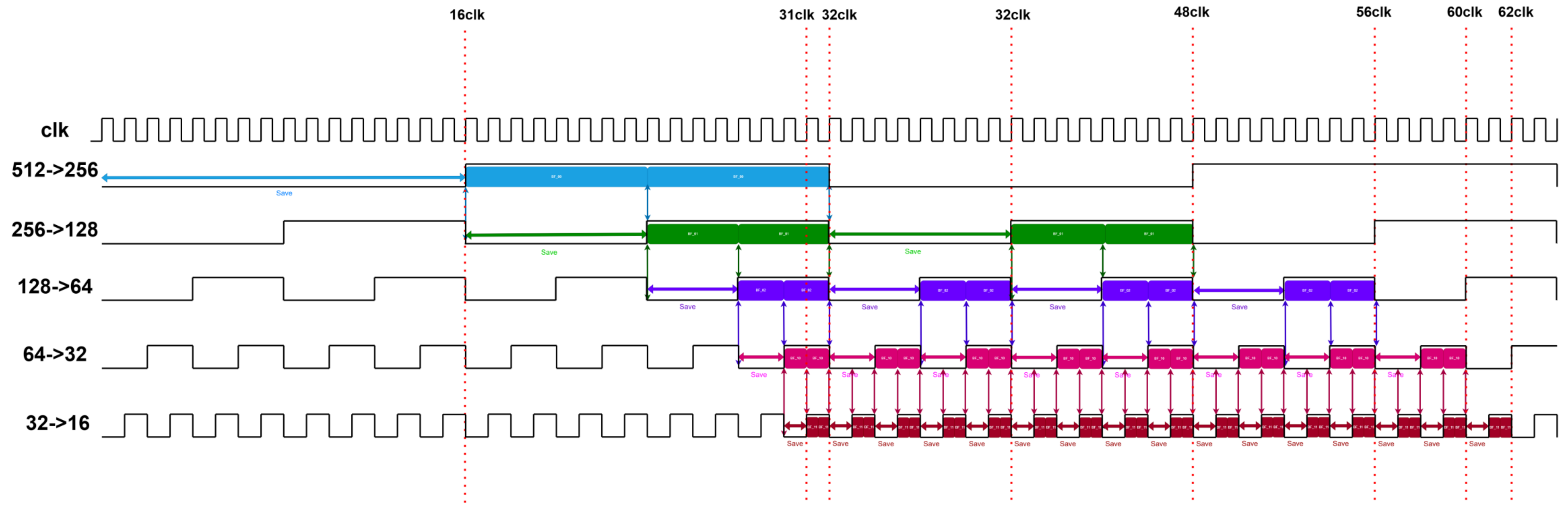


Cnt1_re<=12

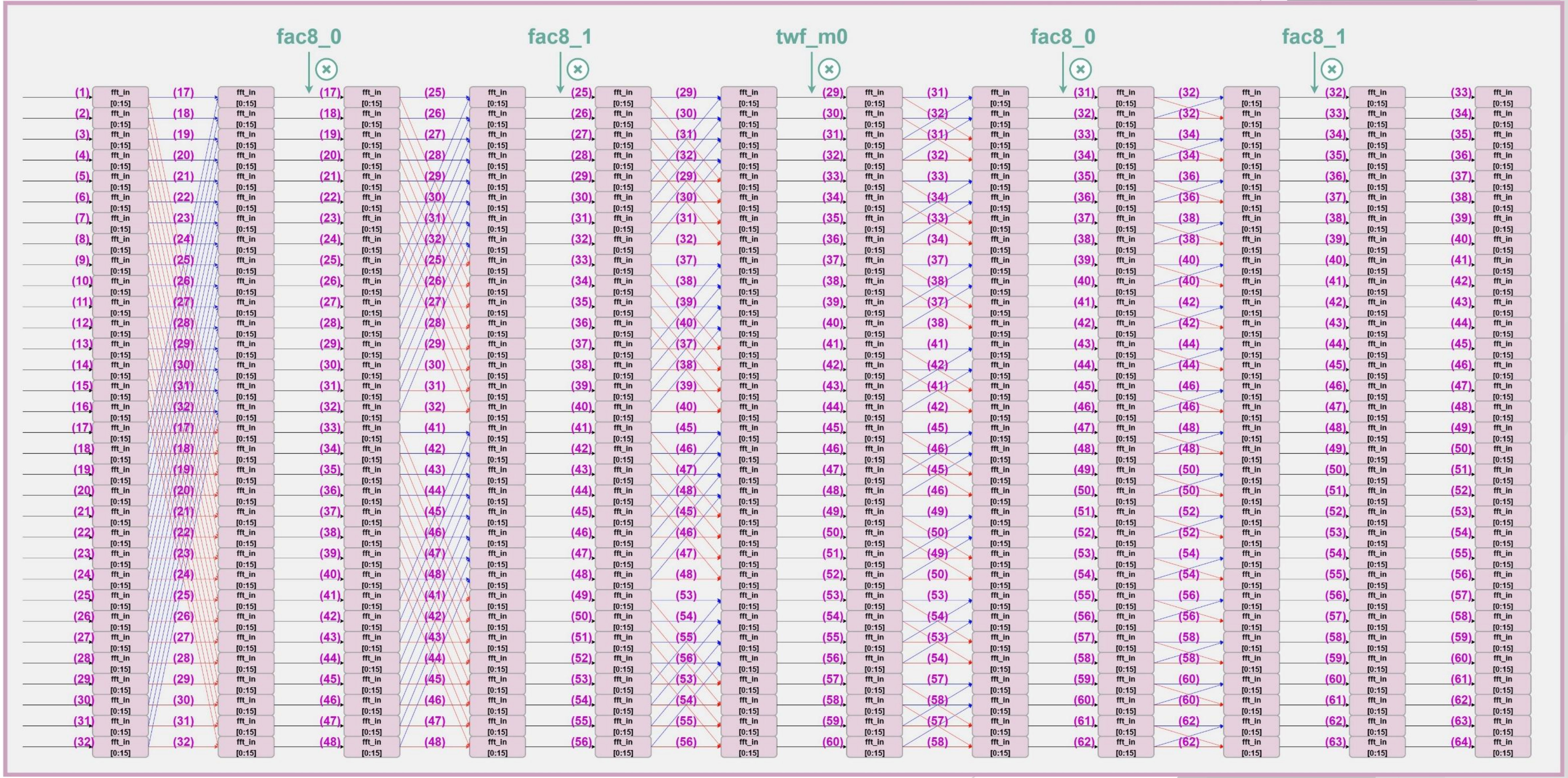


3. Timing

- timing graph



3. Timing



3. FFT architecture

- **module 0**