**Department of Electrical Engineering**
**Indian Institute of Technology Kharagpur**

# Digital Signal Processing Laboratory (EE39203)

**Autumn, 2022-23**

**Experiment 7: Fast Fourier Transform**
Slot: **X** Date: **13/10/2023**

Student Name: **P. Manoj Kumar** Roll No.: **21IE10027**

**Grading Rubric**

| | Tick the best applicable per row | | | Points |
|---|---|---|---|---|
| | Below Expectation | Lacking in Some | Meets all Expectation | |
| Completeness of the report | | | | |
| Organization of the report (5 pts) *With cover sheet, answers are in the same order as questions in the lab, copies of the questions are included in report, prepared in LaTeX* | | | | |
| Quality of figures (5 pts) *Correctly labelled with title, x-axis, y-axis, and name(s)* | | | | |
| Understanding of the frequency range of DFT and effects of zero-padding (35 pts) *DFT and DTFT plots, Matlab code (DTFT samples), questions* | | | | |
| Implementation of Divide-and-Conquer DFT and FFT (40 pts) *Matlab codes (dcDFT, fft2, fft4, fft8, fft stages), questions* | | | | |
| Computation time comparison (15 pts) *Runtimes, questions* | | | | |
| | | | TOTAL (100 pts) | |

Total Points (100): TA Name: TA Initials:

# Digital Signal Processing Laboratory (EE39203)

P Manoj Kumar (21IE10027)

**Experiment 7 - Fast Fourier Transform**

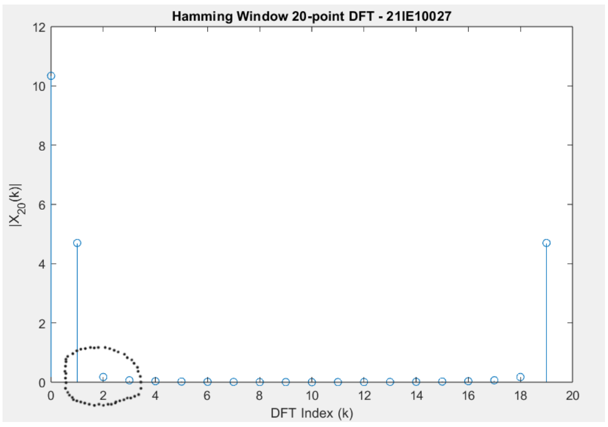
## 1 Introduction

### 1.1 Shifting the Frequency Range

#### 1.1.1 Using DFTSum

The MATLAB Code and the magnitude plot of DTFT of Hamming Window for 20 points using DFTsum is given below.

```
% Create Hamming window
N = 20; % Define the length of the Hamming window as 20
x = hamming(N); % Generate a Hamming window of length N
k = 0:N-1; % Define the index range for DFT
[X] = DFTsum(x);% Compute DFT of the Hamming window
figure;
stem(k, abs(X))
xlabel('DFT Index (k)')
ylabel('|X_{20}(k)|')
title('Hamming Window 20-point DFT - 21IE10027')
```



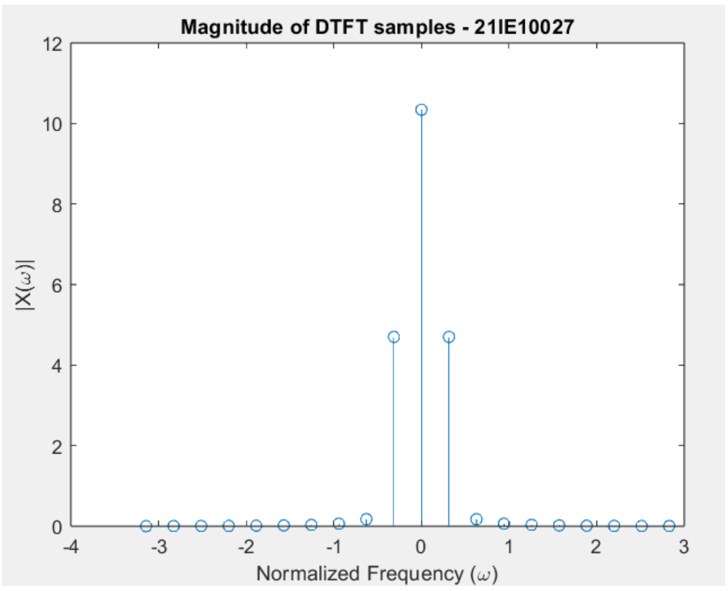
The rounded part in the above figure corresponds to the low frequency range.

#### 1.1.2 Using DTFTSamples

The MATLAB Code and the magnitude plot of DTFT of Hamming Window for 20 points using DTFTsamples is given below.

```
figure;
[X2, w] = DTFTsamples(x);% Compute samples of the DTFT
stem(w, abs(X2))
xlabel('Normalized Frequency (\omega)')
ylabel('|X(\omega)|')
title('Magnitude of DTFT samples - 21IE10027')
```

```
function [X,W] = DTFTsamples(x)
    N = length(x);
    X = DFTsum(x);  % DFT of x
    X = fftshift(X);% Shift the DFT values to be centered around 0
    W = linspace(-pi,pi,N+1);
    W(end)=[];
end
```

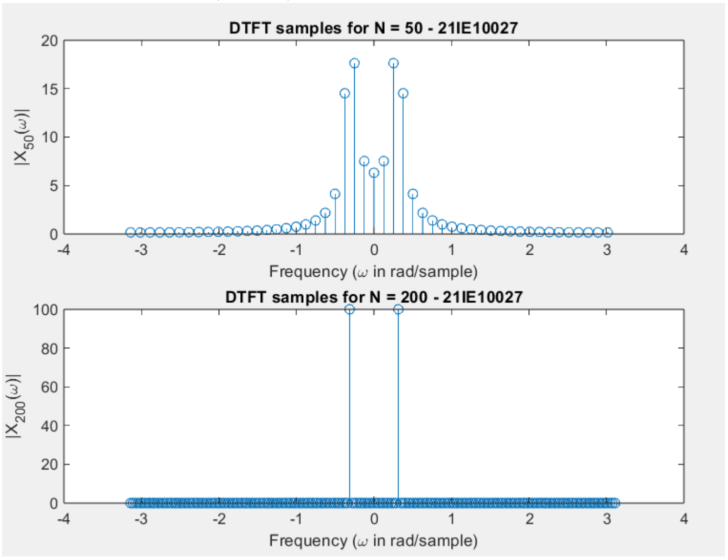


## 1.2 Zero Padding

### 1.2.1 Plot of DTFT Samples for N=50 and N=200

The MATLAB Code and the magnitude plot of DTFT of $sin(0.1\pi n)$ for N=50 and N=200 is given below.

```
% Define parameters
N1 = 50;N2 = 200;
% Step 1: Compute the vector x
n = 0:N1-1;x_n1 = sin(0.1*pi*n);
n = 0:N2-1;x_n2 = sin(0.1*pi*n);
% Step 2: Compute DTFT samples
[X1, w1] = DTFTsamples(x_n1);
[X2, w2] = DTFTsamples(x_n2);
subplot(2,1,1)
stem(w1, abs(X1))
xlabel('Frequency (\omega in rad/sample)')
ylabel('|X_{50}(\omega)|')
title('DTFT samples for N = 50 - 21IE10027')
subplot(2,1,2)
stem(w2, abs(X2))
xlabel('Frequency (\omega in rad/sample)')
ylabel('|X_{200}(\omega)|')
title('DTFT samples for N = 200 - 21IE10027')
```



### 1.2.2 Discussion

- The plot for $N = 200$ looks more like the true DTFT.
- The difference in appearance between the two plots can be attributed to *spectral leakage*, a phenomenon that occurs when the number of points in the Discrete Fourier Transform (DFT) is small. In the case of $N = 50$, the frequency resolution of the resulting DTFT samples is limited. As a result, the DTFT samples may fail to capture fine details in the frequency domain.
- On the other hand, with a larger value of $N$ (such as $N = 200$), the frequency resolution becomes higher. This allows for a more accurate representation of the true DTFT. Additionally, when $N = 200$, *zero-padding* is employed to interpolate additional frequency points. This technique enhances the sampling of the DTFT, resulting in a smoother and more precise representation of the frequency content of the signal.

# 2 The Fast Fourier Transform Algorithm

## 2.1 Implementation of Divide-and-Conquer DFT

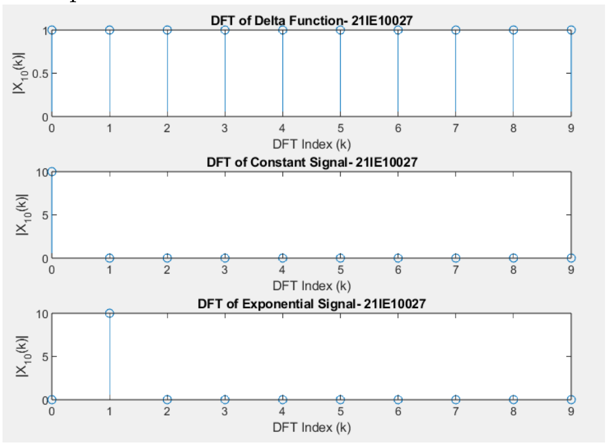
### 2.1.1 MATLAB Code for dcDFT

```
function [X] = dcDFT(x)
    N=length(x);
    j=sqrt(-1);
    % Define indices and split the input signal
    k = 0:N/2-1;
    x0 = x(1:2:N);
    x1 = x(2:2:N);
    % Compute DFT of the even and odd components
    [X0] = DFTsum(x0);
    [X1] = DFTsum(x1);
    % Compute twiddle factors
    W_N = exp(-j*2*pi.*k/N);
    % Combine the DFT results using twiddle factors
    X_k = X0 + W_N .* X1;
    X_k_2 = X0 - W_N .* X1;
    % Concatenate the DFT results
    X = [X_k X_k_2];
end
```

### 2.1.2 Computation of DFT using dcDFT

The MATLAB Code for the magnitude plot of DFT of delta function, constant function and exponential function using dcDFT is given below with the magnitude plots.

```
N = 10;
n = 0:N-1;
k = 0:N-1; % Define the index range for DFT
% Define three different signals
signals = {@(n) (n == 0), @(~) ones(1, N), @(n) exp(1j*2*pi*n/10)};
titles = {'Delta Function', 'Constant Signal', 'Exponential Signal'};
figure;
for i = 1:3
    X = dcDFT(signals{i}(n));
    subplot(3,1,i)
    stem(k, abs(X))
    xlabel('DFT Index (k)')
    ylabel('|X_{10}(k)|')
    title(['DFT of ' titles{i} '- 21IE10027'])
end
```



### 2.1.3 Discussion

1. **Separating samples into even and odd points**:
   - No multiplications are performed in this step.
2. **Computing the two $\frac{N}{2}$ point DFTs** (using `DFTsum`):
   - For each $\frac{N}{2}$ point DFT, there are $\left(\frac{N}{2}\right)^2$ multiplications (since it's a matrix multiplication).
3. **Multiplying by twiddle factors**:
   - There are $\frac{N}{2}$ twiddle factors to compute, each requiring 2 multiplications (real and imaginary parts).
4. **Combining the two DFTs**:
   - For each element of the combined DFT, there are 2 multiplications (real and imaginary parts).

**Total number of multiplications**:

$$2\left(\left(\frac{N}{2}\right)^2+\frac{N}{2}\right)=\frac{3}{2}N^2$$

This analysis assumes that the `DFTsum` function performs the $\frac{N}{2}$ point DFT efficiently. Depending on how `DFTsum` is implemented, the actual number of multiplications may vary.

## 2.2 Recursive Divide and Conquer

### 2.2.1 MATLAB Code for FFT2, FFT4 and FFT8

```
function [X] = FFT2(x)
    X(1)= x(1)+x(2);
    X(2)= x(1)-x(2);
end
```

```
function [X] = FFT4(x)
    N=length(x);
    j=sqrt(-1);
    k=0:N/2-1;
    W_N=exp(-j*2*pi.*k/N);
    x0=x(1:2:N);
    x1=x(2:2:N);
    [X0]=FFT2(x0);
    [X1]=FFT2(x1);
    X_k=X0+W_N.*X1;
    X_k_2=X0-W_N.*X1;
    X=[X_k X_k_2];
end
```

```
function [X] = FFT8(x)
    N=length(x);
    j=sqrt(-1);
    k=0:N/2-1;
    W_N=exp(-j*2*pi.*k/N);
    x0=x(1:2:N);
    x1=x(2:2:N);
    [X0]=FFT4(x0);
    [X1]=FFT4(x1);
    X_k=X0+W_N.*X1;
    X_k_2=X0-W_N.*X1;
    X=[X_k X_k_2];
end
```

### 2.2.2 Output of FFT8 for x[n] = 1 for N = 8

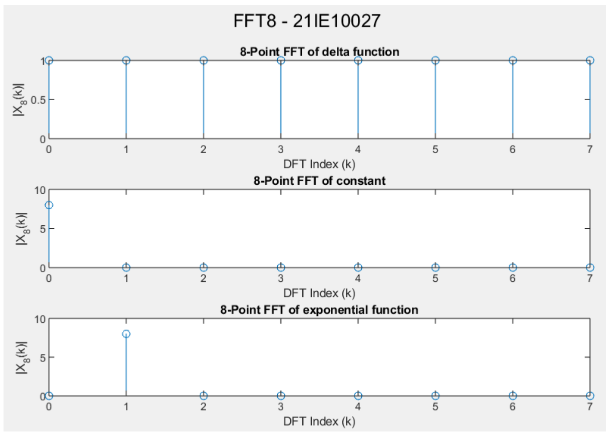
```
>> A = FFT8(ones(1,8));
>> A

A =

     8     0     0     0     0     0     0     0
```

### 2.2.3 Computation of DFT using FFT8

The MATLAB Code for the magnitude plot of DFT of delta function, constant function and exponential function using FFT8 is given below with the magnitude plots.

- $x[n] = \delta[n]$ for $N = 8$.
- $x[n] = 1$ for $N = 8$.
- $x[n] = e^{j\frac{2\pi n}{8}}$ for $N = 8$.

```
N = 8;n = 0:N-1;k = 0:N-1;
signals = {@(n) (n == 0), @(~) ones(1, N), @(n) exp(1j*2*pi*n/8)};
titles = {'delta function', 'constant', 'exponential function'};
figure;
for i = 1:3
    X = FFT8(signals{i}(n));
    subplot(3,1,i)
    stem(k, abs(X))
    xlabel('DFT Index (k)')
    ylabel('|X_8(k)|')
    title(['8-Point FFT of ' titles{i}])
end
sgtitle('FFT8 - 21IE10027')
```



### 2.2.4 Calculations

- **Number of Multiples with Twiddle Factor in FFT8**:
  - In FFT4, there are 2 multiplications for every call. Since FFT4 is called twice, there are a total of 4 multiplications with the twiddle factor.
  - Then, FFT8 is called once, and for each call, 4 multiplications occur. Hence, in total, there will be 8 multiplications with the twiddle factor.
- **Number of Multiples with Twiddle Factor in General FFT (N = $2^p$)**:
  - In the FFT, the number of levels gets reduced to $\log_2(N)$ due to halving at each stage.
  - Each stage (except the last one with 2 elements) will involve $\frac{N}{2}$ multiplications. So, in total, there will be $\left(\frac{N}{2}\right) \cdot (\log_2(N) - 1)$ multiplications with the twiddle factor.
- **For** $N = 2^p$:
  - Number of Multiples $= p \cdot (\log_2(2p) - 1)$
- **For** $N = 2^p$:
  - Number of Multiples $= \frac{N}{2} \cdot (p - 1)$
- **If** $N = 2^{10}$ **(i.e.,** $p = 10$**)**:
  - The number of multiples is $9 \cdot 2^9 = 4608$.

## 2.3 Recursive FFT Algorithm for Power-of-2 Length Signal

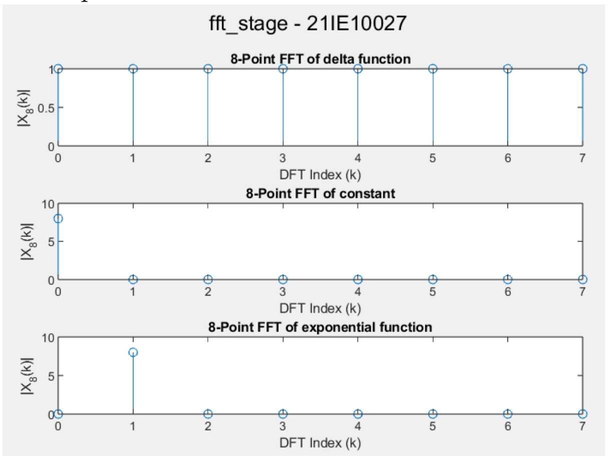
### 2.3.1 MATLAB Code for fft_stage

```
function [X] = fft_stage(x)
    N=length(x);
    if (N==2)
       X=FFT2(x);
       return;
    elseif (N>2)
        j=sqrt(-1);
        % Define indices and twiddle factors
        k = 0:N/2-1;
        W_N = exp(-j*2*pi.*k/N);
        % Split the input signal into even and odd indices
        x0 = x(1:2:N);x1 = x(2:2:N);
        % Recursively apply FFT on even and odd components
        [X0] = fft_stage(x0);[X1] = fft_stage(x1);
        % Combine the FFT results using twiddle factors
        X_k =   X0 + W_N .* X1;X_k_2 = X0 - W_N .* X1;
        X=[X_k X_k_2];
    end
end
```

### 2.3.2 Computation of DFT using fft_stage

The MATLAB Code for the magnitude plot of DFT of delta function, constant function and exponential function using fft_stage is given below with the magnitude plots.

```
figure;
for i = 1:3
     [X] = fft_stage(signals{i}(n));
     subplot(3,1,i)
     stem(k, abs(X))
     xlabel('DFT Index (k)')
     ylabel('|X_8(k)|')
     title(['8-Point FFT of ' titles{i}])
end
sgtitle('fft\_stage - 21IE10027')
```



### 2.3.3 Discussion

- **Matching Results**:
  - The outputs of the recursive FFT implementation (`fft_stage`) are identical to the results obtained from the non-recursive FFT (`FFT8`) when applied to the same input signals.
- **General Solution**:
  - The recursive approach (`fft_stage`) is designed to handle signals of various lengths, not just limited to a specific length like the original `FFT8` function.

The recursive FFT excels in flexibility, effortlessly handling signals of any power-of-2 length. This versatility outshines the need for separate functions for specific lengths, resulting in leaner and more efficient code. Additionally, its modular structure simplifies maintenance and updates, while its scalability effortlessly processes larger lengths by breaking them into manageable steps. Overall, the recursive FFT is a versatile and efficient tool for computing FFTs across varying signal lengths.