# 25 Spring ECEN 610: Mixed-Signal Interfaces

## Lab1: Signal Processing Concepts Review


Name: Yu-Hao Chen

UIN:435009528

Section:601

Professor: Sebastian Hoyos

TA: Sky Zhao

1. DIGITAL FILTERS (20%)

a. Digital filters are broadly classified into FIR and IIR filters. Give an example of an FIR filter and IIR filter (transfer function). Plot the transfer function in Python. Identify the poles and zeros on the plot.

b. Consider the transfer functions,

$$H(z) = 1 + z^{-1} + z^{-2} + z^{-3} + z^{-4}$$

$$H(z) = \frac{1 + z^{-1}}{1 - z^{-1}}$$

Identify the FIR and IIR filter. Plot the FIR filter in (use freqz function in the SciPy signal processing toolbox). Where are the poles and zeros of the filter located? Validate your theory using simulations.

c. Comment on the stability of the FIR and IIR filters. Use simple simulations to explain your ideas.

**FIR (Finite Impulse Response) Filters**

- Impulse response is finite, meaning it settles to zero after a fixed number of samples.
- No Feedback: The output depends only on the current and past input values.
- Always Stable: FIR filters have no poles, meaning they are **inherently stable.**
- y(n)=∑h(k)·x(n−k)

For example:

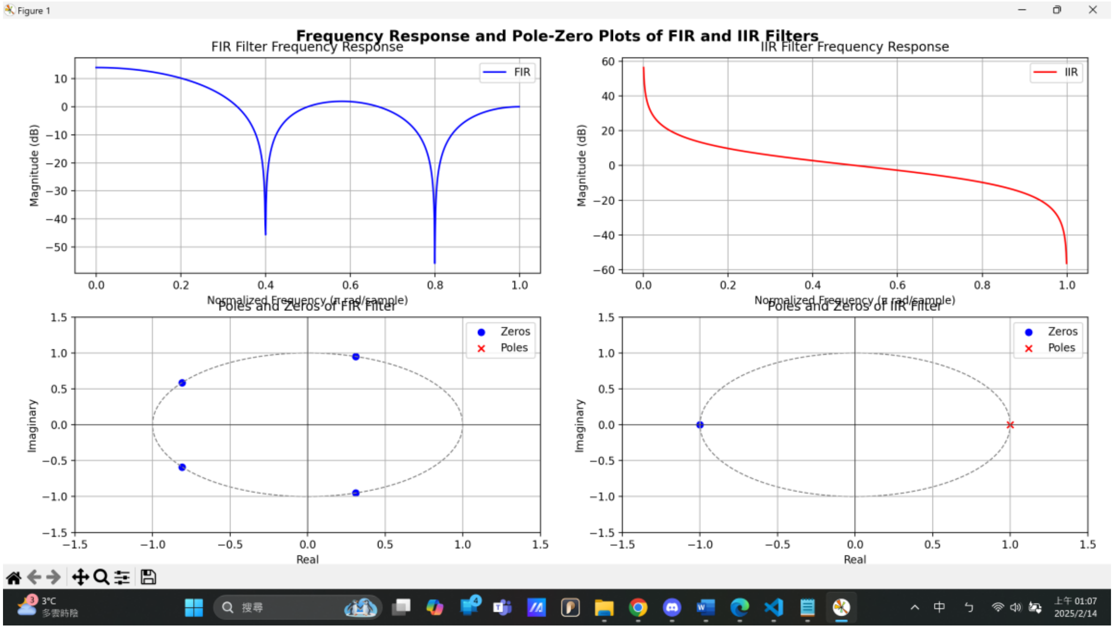
$$H(z) = 1 + z^{-1} + z^{-2} + z^{-3} + z^{-4}$$

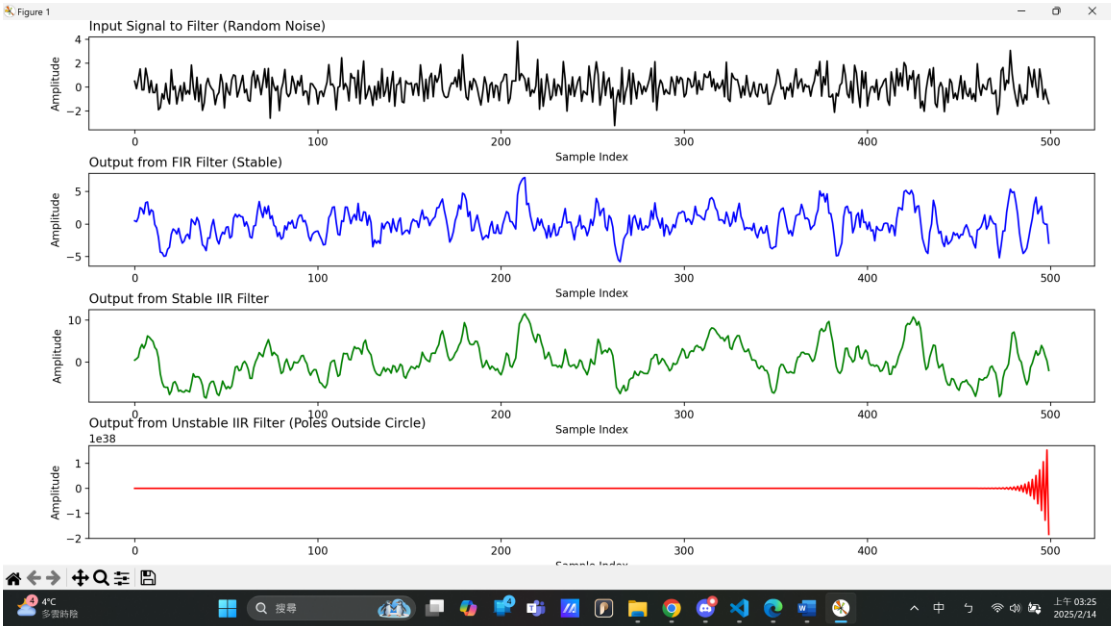
**IIR (Infinite Impulse Response) Filters**

- Impulse response extends infinitely, meaning it never completely settles to zero.
- Feedback Present: The output depends on both past inputs and past outputs.
- IIR filters have poles, which can make them **unstable if placed outside the unit circle** in the Z-plane.
- y(n)= ∑bk·x(n−k)- ∑aj·y(n−j)

For example:

$$H(z) = \frac{1 + z^{-1}}{1 - z^{-1}}$$




## 2. SAMPLING (50%)

a. Consider the two signals $x1(t) = cos(2\pi \cdot F1 \cdot t)$ and $x2(t) = cos(2\pi \cdot F2 \cdot t)$, where F1 = 300MHz and F2 = 800MHz. Both these signals are sampled at the same sampling frequency Fs = 500MHz. What can you say about the sampled data x1(n) and x2(n)? Explain with simulations why this happens.

b. Can you recover the signals x1(t) and x2(t) from x1(n) and x2(n). If not, what is your suggestion to overcome this problem?

c. Find the ideal signal reconstruction (interpolation) equation for a zero-order hold sampling system with pulse width W and sampling rate T. Assume that Nyquist rate criteria is satisfied and the sampling point is at the end of the pulse width.

d. Sample the signal x1(t) using Fs = 800MHz at 0:Ts:T-Ts, where T = 10/F1 (i.e. 10 cycles of the cosine wave) and Ts = 1/Fs. Reconstruct the signal from the samples using the formula,

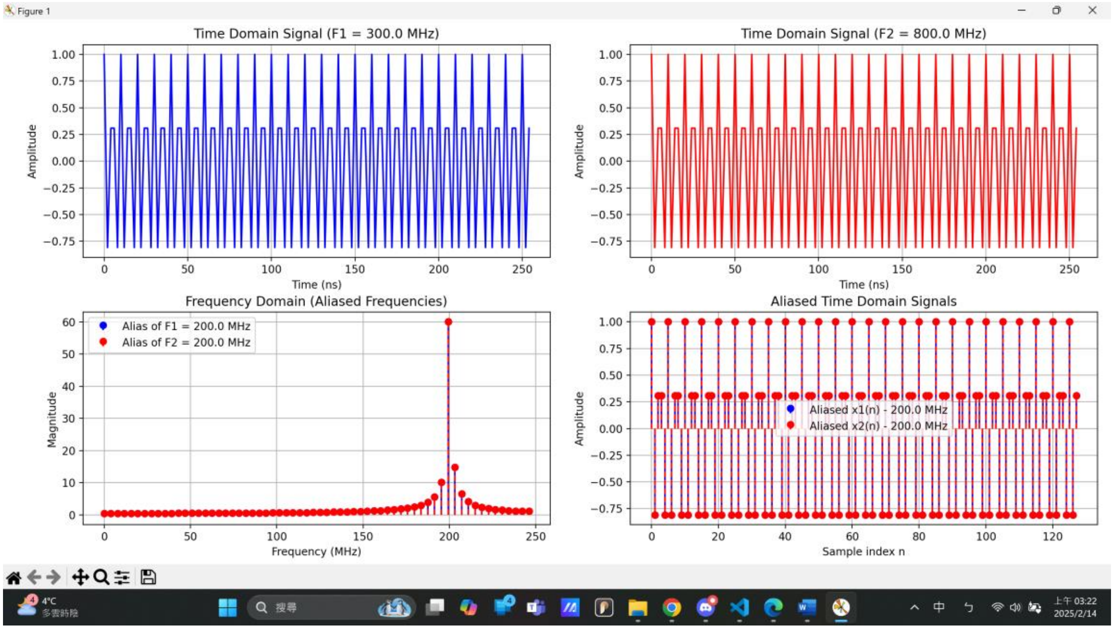
$$x_r(t) = \sum_{n=-\infty}^{n=\infty} x(n) \frac{\sin[\pi(t-nTs)/Ts]}{\pi(t-nTs)/Ts}$$

Now sample the signal at Ts/2:Ts:T-Ts/2 i.e. the samples are shifted by Ts/2. Reconstruct the signal using the same formula. Compute the mean square error (MSE) in the reconstruction in both the cases by using,

$$MSE= mean((x_r(t)-x(t))^2)$$

e. Repeat d. for Fs = 1000MHz and Fs = 500MHz. Report your observations.

a. F Nyquist= Fs/2= 250M Hz
Fa=|F−kFs|, determined k that 0≤ Fa≤ FN
Fa1=|300−1·500|=|300−500|=200 MHz
Fa2=|800−2·500|=|300−1000|=200 MHz
The two Fin after aliasing will appear on the same frequency 200M Hz.



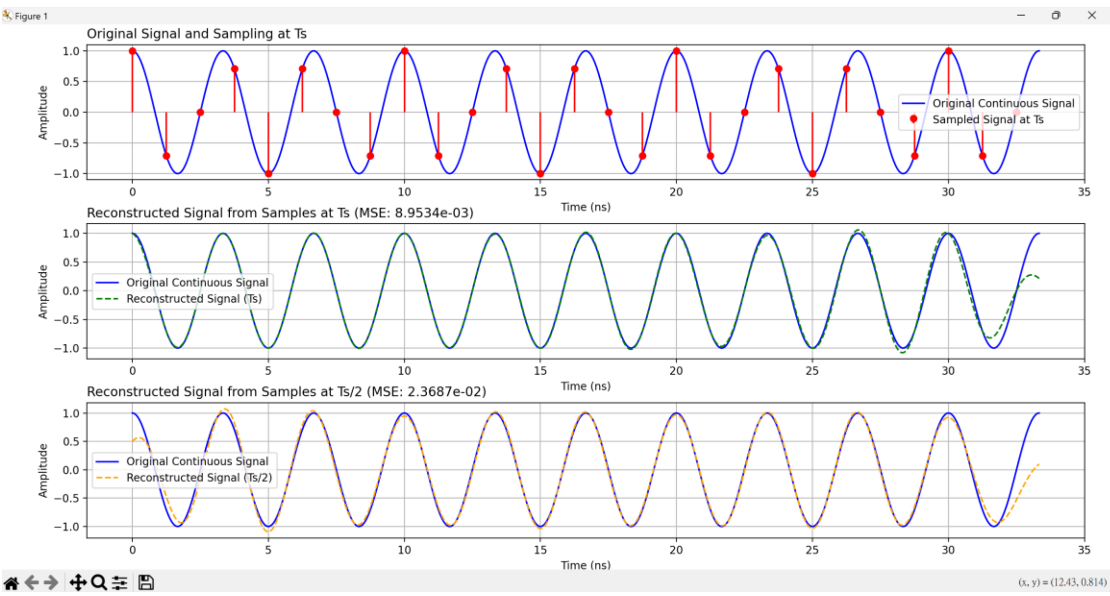

b. The best way to avoid this situation is to increase the Fs

c. xr(t)= ∑( n=−∞~∞) x(n)·hZOH(t−nT), hZOH=1 when 0 ≤t<T, =0 when others

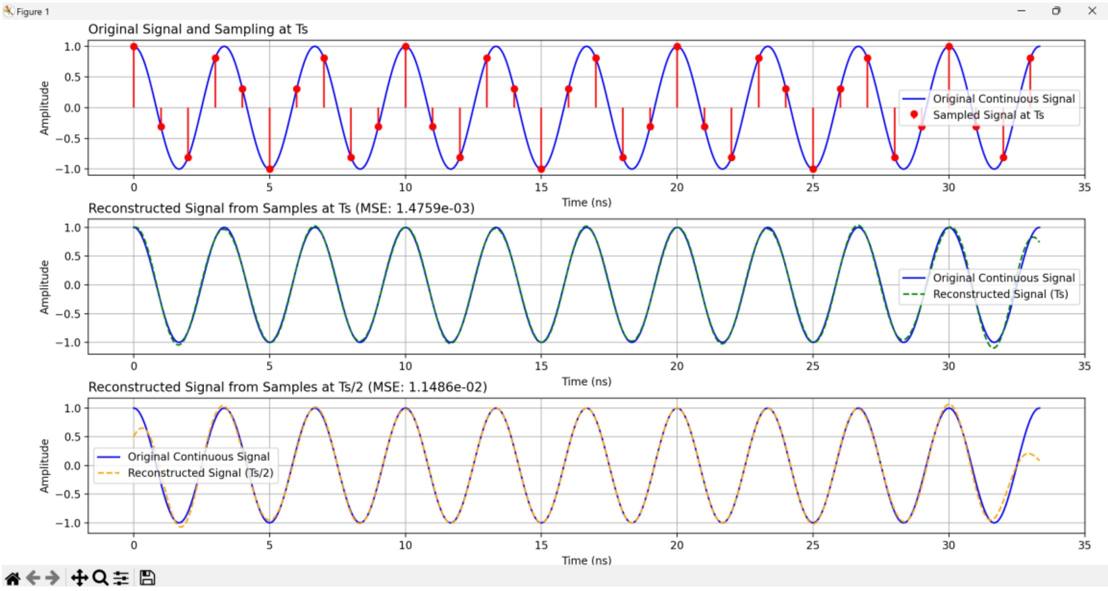
d.

- we can use x(t)= ∑( n=−∞~∞) x(nTs)·sinc(Tst−nTs) to reconstruct the original signal
- sinc(x)=sin(πx)/ πx, idea LPF can preserve the 0-Fs/2 signal
- sinc(n−m)=1 when n=m (at x[n] point), =0 when n!=m (at others), ideally keep the x[n] signal and avoid alias.
- xr(t)=n=∑(−∞~∞) x(nTs)·sinc(Tst−nTs) in time domain means x[n] will be affected by all the other none x[n] point, but other none x[n] point=0 in sinc.

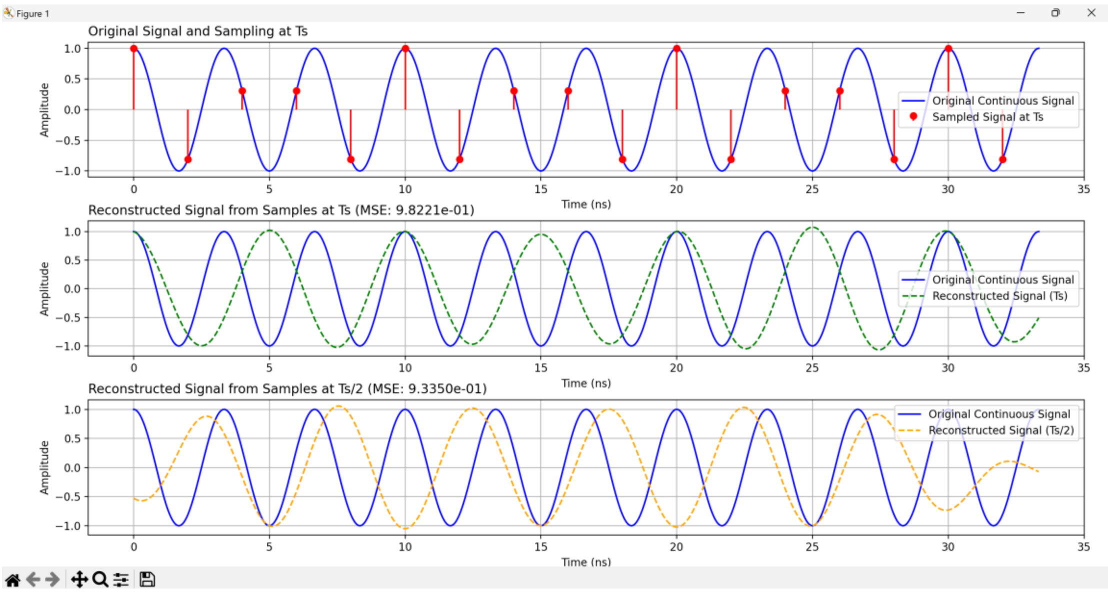
Fs= 800M



Fs= 1000M




Fs= 500M




- If the Fs isn't large enough (compared to the Fin (Nyquist)), the reconstruct signal maybe contain alias signal which might cause wrong reconstruction.
- Shifting the fs won't affect the reconstruct signal (if no alias), but only affect the phase of it.

3. DISCRETE FOURIER TRANSFORM (30%)

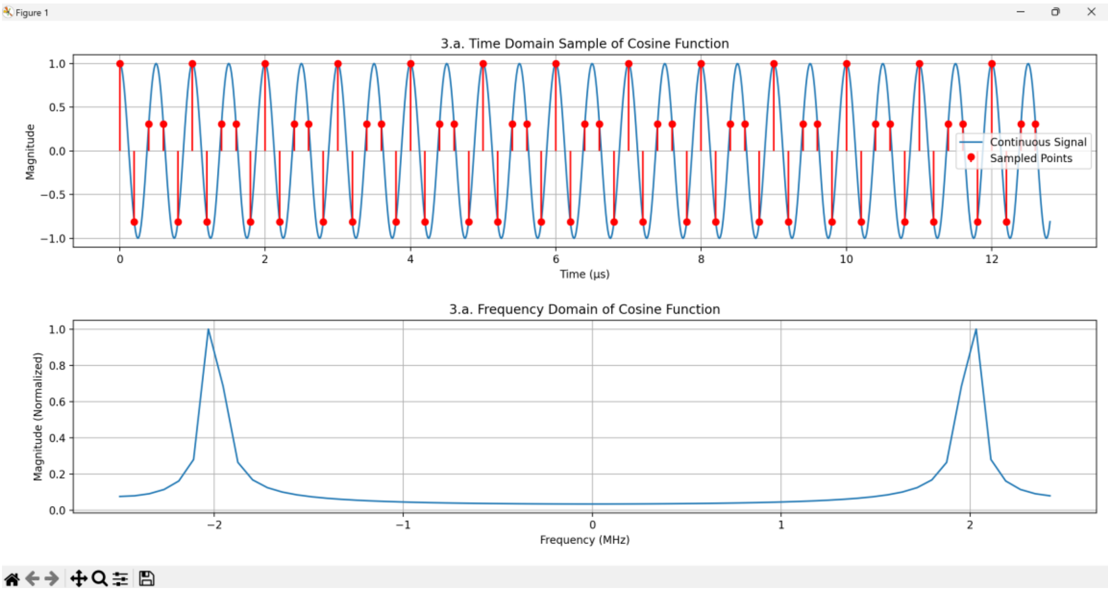
a. Consider the signal $x(t) = cos(2\pi \cdot F \cdot t)$ where F = 2MHz. Sample the signal at Fs = 5MHz. Compute a 64 point DFT in Python and plot the output. (see **fft** command in SciPy documentation).

b. Consider another signal $y(t) = cos(2\pi \cdot F1 \cdot t) + cos(2\pi \cdot F2 \cdot t)$ where F1 = 200MHz and F2 = 400MHz. Sample this signal at Fs = 1GHz. Compute and plot a 64 point DFT. Can you identify the two components of the signal in the plot?

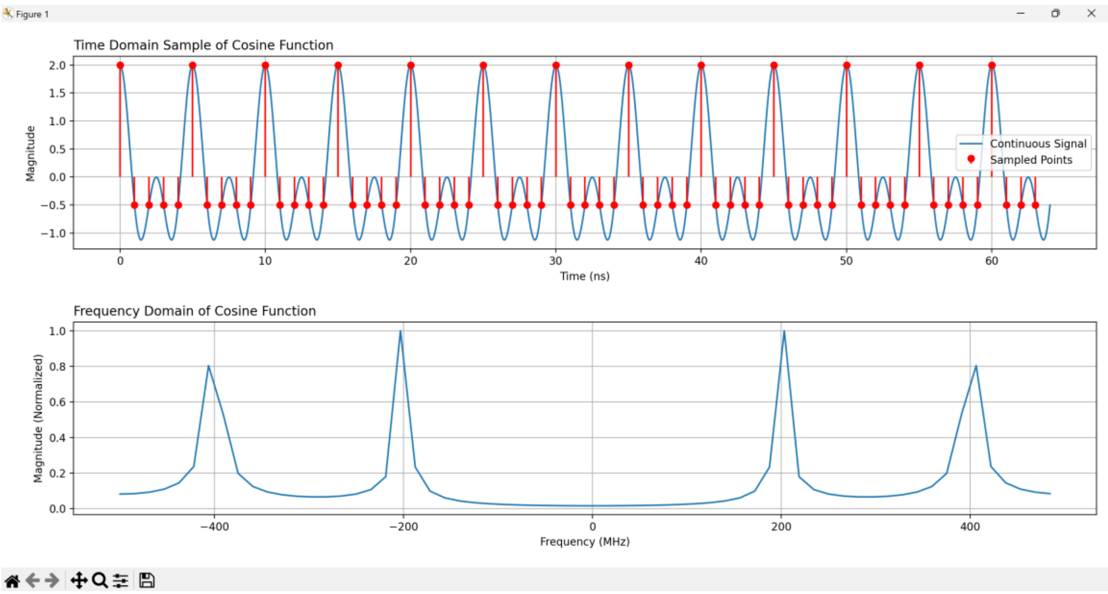
c. Repeat b. using Fs = 500MHz. Explain what you observe in your DFT plot.

d. Now apply a Blackman window as an envelope to the signal x(t) and y(t) and repeat the analysis. Please explain the differences.

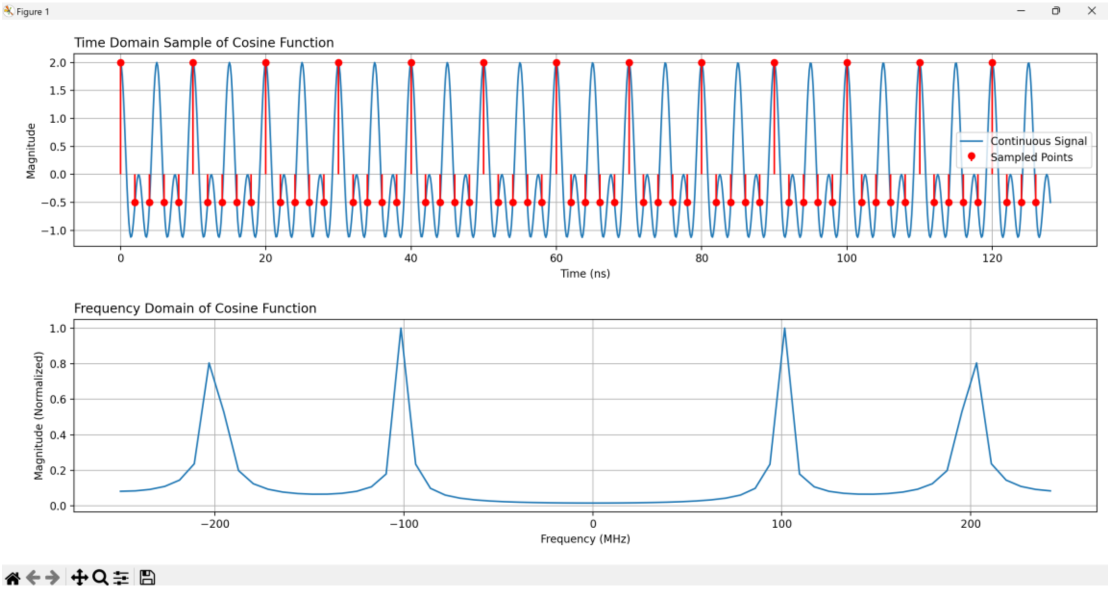
a.




b.



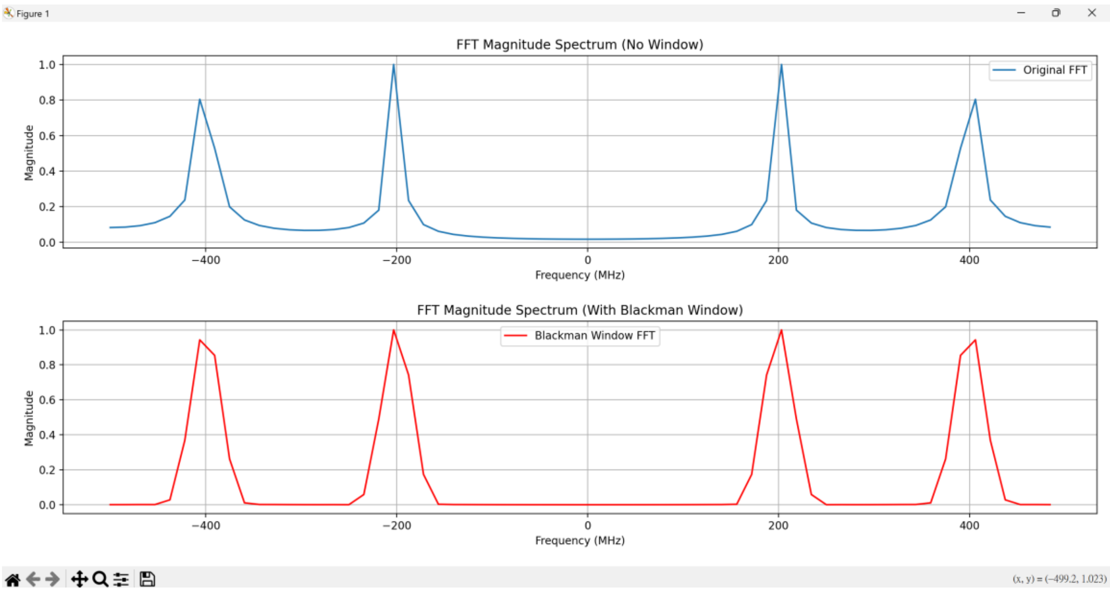
c.




Alias happens because of the fs decrease

d.

- In FFT-based spectral analysis, directly truncating a signal can introduce **edge effects**, causing energy to spread across different frequencies (**spectral leakage**).
- Applying a **Blackman window** helps **minimize leakage**, making the frequency spectrum more accurate, at the cost of slightly reduced frequency resolution.




- Appendix A (hand calculation)

IIR ( poles + zero )

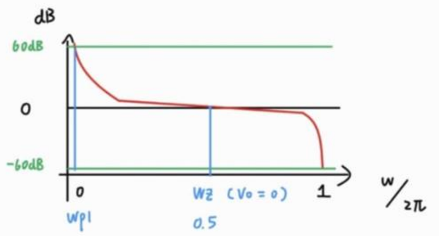
$$H(s) = \frac{1 + z^{-1}}{1 - z^{-1}} \longrightarrow z = e^{jw} \longrightarrow H(e^{jw}) = \frac{1 + e^{-jw}}{1 - e^{-jw}}$$

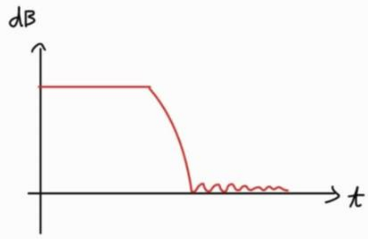
when $w = 0$ (DC) $H(e^{j0}) = \infty$ $w_{p1}$ : $z = 1$

when $w = \pi$ (nyquist) $H(e^{j\pi}) = 0$ $w_z$ : $z = -1$

$z = e^{jw}$ $0 \le w \le 2\pi$ $\longrightarrow$ normalized $0 \le w \le 1$

| | | | |
|---|---|---|---|
| $z$ = | 1 | -1 | |
| $w$ = | 0 | $\pi$ | $2\pi$ |
| normalized = | 0 | 0.5 | 1 |



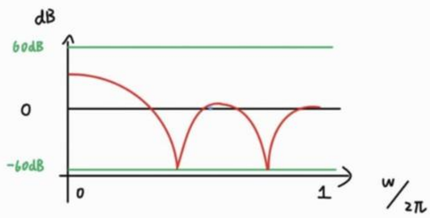



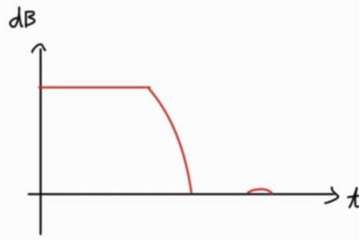

FIR ( only zero )

$$H(s) = 1 + z^{-1} + z^{-2} + z^{-3} + z^{-4}$$

$$1 + z^{-1} + z^{-2} + z^{-3} + z^{-4} = 0 \longrightarrow z^4 + z^3 + z^2 + z + 1 = 0$$

$$f_k = e^{j\frac{2\pi k}{5}} \quad k = 1,2,3,4 \quad f = \frac{2}{5}, \frac{4}{5}$$




## ■ Appendix B (Math-code)

a.FIR&IIR

C: > Users > harri > OneDrive > 桌面 > TAMU > ECEN 610 Mixed-signal > lab1 > code > a.py > ...

```python
import numpy as np
import scipy.signal as signal
import matplotlib.pyplot as plt

# === Define FIR and IIR Filters ===

# FIR filter: H(z) = 1 + z^(-1) + z^(-2) + z^(-3) + z^(-4)
fir_b = [1, 1, 1, 1, 1]  # FIR numerator coefficients (Zero)
fir_a = [1]  # FIR filter denominator (only 1 for FIR) (no pole)

# IIR filter: H(z) = (1 + z^(-1)) / (1 - z^(-1))
iir_b = [1, 1]  # IIR numerator coefficients (zero)
iir_a = [1, -1]  # IIR denominator coefficients (poles)

# === Compute Frequency Response ===
w_fir, h_fir = signal.freqz(fir_b, fir_a, worN=1024) # w, h = signal.freqz(b(分子係數), a(分母係數), worN=1024)
w_iir, h_iir = signal.freqz(iir_b, iir_a, worN=1024) # w(0 ~ π rad/sample) h(Complex values, including amplitude & phase)

# === Compute Poles and Zeros ===
zeros_fir, poles_fir, _ = signal.tf2zpk(fir_b, fir_a) # zeros, poles, gain = signal.tf2zpk(b, a)
zeros_iir, poles_iir, _ = signal.tf2zpk(iir_b, iir_a)
```

a.Alias to the same frequency

Q2_a.py > ...

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.fftpack import fft

# === Fin ===
Fs = 500e6
F1 = 300e6
F2 = 800e6
N = 128

t = np.linspace(0, N/Fs, N, endpoint=False)  # contionus time
n = np.arange(N)  # discrete time

# === contimuous time signal ===
x1_t = np.cos(2 * np.pi * F1 * t)  # contionus time x1(t)
x2_t = np.cos(2 * np.pi * F2 * t)  # contionus time x2(t)

# === Fa freq ===
k1 = round(F1 / Fs)
Fa1 = abs(F1 - k1 * Fs)
k2 = round(F2 / Fs)
Fa2 = abs(F2 - k2 * Fs)

x1_n = np.cos(2 * np.pi * Fa1 * n / Fs)  # x1(n)
x2_n = np.cos(2 * np.pi * Fa2 * n / Fs)  # x2(n)

# ===  FFT  ===
X1_f = np.abs(fft(x1_n))[:N//2]
X2_f = np.abs(fft(x2_n))[:N//2]
freqs = np.fft.fftfreq(N, d=1/Fs)[:N//2]  # freq ( Nyquist range)
```

b.code for testing alias f1&f2

Q2_a.py > ...

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.fftpack import fft

# === Fin ===
Fs = 500e6
F1 = 300e6
F2 = 800e6
N = 128

t = np.linspace(0, N/Fs, N, endpoint=False)  # contionus time
n = np.arange(N)  # discrete time

# === contimuous time signal ===
x1_t = np.cos(2 * np.pi * F1 * t)  # contionus time x1(t)
x2_t = np.cos(2 * np.pi * F2 * t)  # contionus time x2(t)

# === Fa freq ===
k1 = round(F1 / Fs)
Fa1 = abs(F1 - k1 * Fs)
k2 = round(F2 / Fs)
Fa2 = abs(F2 - k2 * Fs)

x1_n = np.cos(2 * np.pi * Fa1 * n / Fs)  # x1(n)
x2_n = np.cos(2 * np.pi * Fa2 * n / Fs)  # x2(n)

# ===  FFT  ===
X1_f = np.abs(fft(x1_n))[:N//2]
X2_f = np.abs(fft(x2_n))[:N//2]
freqs = np.fft.fftfreq(N, d=1/Fs)[:N//2]  # freq ( Nyquist range)
```

b.c~e

```python
Q2_d.py > ...
import numpy as np
import matplotlib.pyplot as plt

Fs = 800e6
F1 = 300e6
T = 10 / F1  # Running time for 10 cycle (10Ts)
Ts = 1 / Fs  # sample rate  1.25e-9 sec

# === Continuous time signal ===
t_cont = np.linspace(0, T, 1000)  # Continuous time 0 ~ T make up with 1000points
x_cont = np.cos(2 * np.pi * F1 * t_cont)  # Original signal Amp cos()

# === Sampled signal ===
t_sampled = np.arange(0, T-Ts, Ts)  # discrete time 0 ~ T sample every Ts
x_sampled = np.cos(2 * np.pi * F1 * t_sampled)

# === Sinc interpolation function ===
def sinc_interp(xn, tn, t_interp): #(sample point, sample point's time, reconstruct time)
    """ sinc to get original signal """
    Ts = tn[1] - tn[0]  # sample time= xn[i] - xn[i-1]= Ts= 1/Fs
    return np.sum(xn * np.sinc((t_interp[:, None] - tn) / Ts), axis=1)
# multiplies the sinc function weight by each sampled value xn, sinc, axis=1 (horizontal summation)

# Reconstructed signal (Ts sample)
x_reconstructed_Ts = sinc_interp(x_sampled, t_sampled, t_cont)

#**********************************************************************************************************************

# Shifted sample (Ts/2)
t_sampled_shifted = np.arange(Ts/2, T-Ts/2, Ts)  # Sample points shifted by Ts/2
x_sampled_shifted = np.cos(2 * np.pi * F1 * t_sampled_shifted)  # Shifted sampled values
x_reconstructed_Ts_half = sinc_interp(x_sampled_shifted, t_sampled_shifted, t_cont)

# === MSE ===
mse_Ts = np.mean((x_reconstructed_Ts - x_cont) ** 2)
mse_Ts_half = np.mean((x_reconstructed_Ts_half - x_cont) ** 2)

#----------------------------------------------------------------------------------------------------------------------
```

C1. (time domain and FFT DFT freq domain)

```python
Q3_a.py > ...
import numpy as np
import matplotlib.pyplot as plt

Fs = 5e6    # Sampling Frequency
F = 2e6     # Signal Frequency
N = 64      # FFT Points
Ts = 1 / Fs  # Sampling Period

# === Continuous Time Signal ===
t_cont = np.linspace(0, N * Ts, 1000)  # Continuous 1000 points
x_cont = np.cos(2 * np.pi * F * t_cont)  # Original signal

# === Sampled Signal (Discrete Time) ===
t_n = np.arange(N) * Ts  # 64 sample points
x_n = np.cos(2 * np.pi * F * t_n)  # Sampled cosine wave

# === FFT Calculation ===
X_k = np.fft.fft(x_n, N)  # 64-point FFT
frequencies = np.fft.fftfreq(N, Ts)  # Corresponding frequency axis fftfreq(N, time)
X_k_shifted = np.fft.fftshift(X_k)   # Center zero frequency
frequencies_shifted = np.fft.fftshift(frequencies) / 1e6  # Convert to MHz
```

C2.

```python
Q3_b.py > ...
import numpy as np
import matplotlib.pyplot as plt

# Sampling parameters
Fs = 0.5e9    # Sampling Frequency (1 GHz)
F1 = 200e6  # Frequency component 1 (200 MHz)
F2 = 400e6  # Frequency component 2 (400 MHz)
N = 64         # FFT Points
Ts = 1 / Fs # Sampling Period

# === Continuous Time Signal ===
t_cont = np.linspace(0, N * Ts, 1000)  # 1000-point continuous signal
y_cont = np.cos(2 * np.pi * F1 * t_cont) + np.cos(2 * np.pi * F2 * t_cont)

# === Sampled Signal (Discrete Time) ===
t_n = np.arange(N) * Ts  # 64 sample points
y_n = np.cos(2 * np.pi * F1 * t_n) + np.cos(2 * np.pi * F2 * t_n)  # Sampled signal

# === FFT Calculation ===
Y_k = np.fft.fft(y_n, N)  # 64-point FFT
frequencies = np.fft.fftfreq(N, Ts)  # Corresponding frequency axis
Y_k_shifted = np.fft.fftshift(Y_k)   # Center zero frequency
frequencies_shifted = np.fft.fftshift(frequencies) / 1e6  # Convert to MHz
```

C3. Blackman window

Q3_d.py > ...

```python
import numpy as np
import matplotlib.pyplot as plt

Fs = 1e9
F1 = 200e6
F2 = 400e6
N = 64
Ts = 1 / Fs

# === Discrete-Time Signal ===
t_n = np.arange(N) * Ts  # 64 points
y_n = np.cos(2 * np.pi * F1 * t_n) + np.cos(2 * np.pi * F2 * t_n)  # original signal

# === Blackman ===
blackman_window = np.blackman(N)  # generate blackman
y_n_windowed = y_n * blackman_window  # adding blackman

# === FFT ===
Y_k = np.fft.fft(y_n, N)  # FFT
Y_k_windowed = np.fft.fft(y_n_windowed, N)  # Black FFT

# === Freq axis ===
frequencies = np.fft.fftfreq(N, Ts)
frequencies_shifted = np.fft.fftshift(frequencies)

# ===  0 Hz center ===
Y_k_shifted = np.fft.fftshift(Y_k)
Y_k_windowed_shifted = np.fft.fftshift(Y_k_windowed)
```