25 Spring ECEN 610: Mixed-Signal Interfaces

Lab4: Sampler Error Modeling and Correction

Name: Yu-Hao Chen

UIN:435009528

Section:601

Professor: Sebastian Hoyos

TA: Sky Zhao

https://github.com/Yu-HaoChen/TAMU_ECEN610_Mixed_signal/tree/main/lab4

1. First order model of a ZOH sampling circuit Construct a model for a sampling circuit shown in Fig. 1.

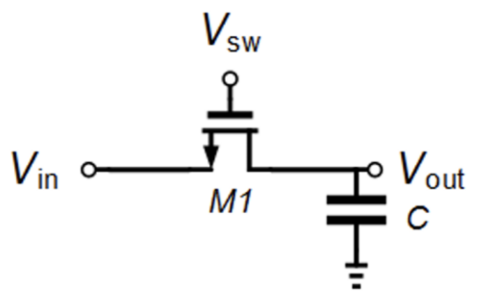
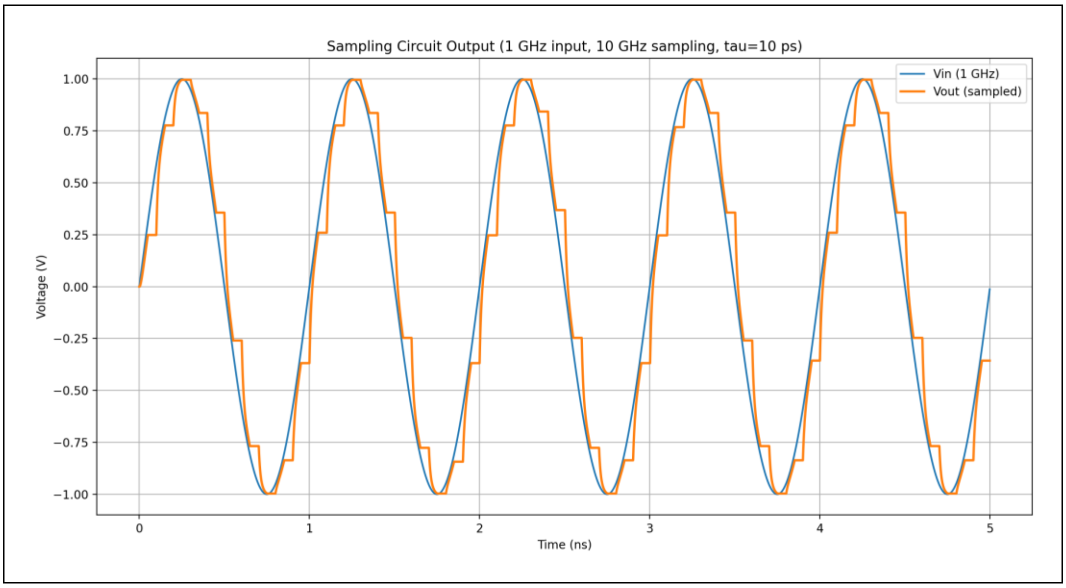


Figure 1: Sampling Circuit

When the NMOS switch M1 is ON (Vsw = 1), the sampling circuit behaves as a series RC circuit and the input Vin is sampled on the capacitor. When the switch turns OFF (Vsw = 0), the voltage on the capacitor is held constant until the beginning of the next sampling phase. If the ON resistance of the switch is R, then the time constant of the sampler, τ, is R*C.

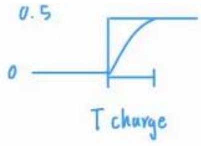
a. For an input sinusoidal signal of frequency 1 GHz, a sampling frequency of 10 GHz and time constant of 10 ps, plot the output of the sampling circuit.



2. Sampling Error Sampling error is the difference between an ideally sampled signal (delta train) and a signal sampled with a finite time constant sampling circuit.

a. Assume a NRZ (Non Return to Zero) input of amplitude 0.5 V and data rate of 10 Gb/s. Sample the input signal once in the middle of every bit period. Assuming a 50% duty cycle for Vsw in Fig. 1, what should the time constant be for the maximum sampling error to be less than 1 LSB for a 7-bit ADC with a full scale range of 1 V. Justify with an equation the obtained time constant value.

$T_{bit} = \frac{1}{10 \times 10^9} = 100\ ps$



$T_{charge} = \frac{T_{bit}}{2} = 50\ ps$

7 bits ADC $\quad LSB = \frac{1V.}{2^7} = \frac{1}{128} V. \approx 7.8125\ m$

$V_{out\ charge} = 0.5 \times (1 - e^{-\frac{T_{charge}}{\tau}})$

$\Delta V = 0.5 - V_{out\ charge} < 1\ LSB$

$\tau < 12.02 \times 10^{-12}\ s.$

b. Assume a multi-tone signal input with frequencies of 0.2 GHz, 0.58 GHz, 1 GHz, 1.7 GHz and 2.4 GHz and a sampling frequency of 10 GHz. What should the time constant be for the sampling error to be less than 1 LSB for a 7-bit ADC? Is it different from the time constant in 2.a? Why?

RC 1 pole system $\quad H(jw) = \frac{1}{1 + jw\tau}$

$|H(jw)| = \frac{1}{\sqrt{1 + (w\tau)^2}}$

$\Delta V(w) = 0.5 \left[1 - \frac{1}{\sqrt{1 + (w\tau)^2}} \right]$

$W_{max\ (2.4G)} = 2\pi \times 2.4G = 15.08 \times 10^9\ rad/s$

$\Delta V(w) <$ 1 LSB of 7 bit ADC

$\tau < 11.73 \times 10^{-12}\ s$

- For NRZ signals, although there is no concept of frequency, the data rate of 10 Gb/s determines the duration of each bit (100 ps), and the charging time is about 50 ps (because of the 50% duty cycle).
- For multi-audio signals, the impact of each frequency component (especially the highest frequency component) on the RC charging error needs to be considered.

3. Sampling Error Estimation Construct an ADC model by adding an N-bit quantizer (N=7) at the output of the sampling circuit.
   a. Let the input to the ADC be the multitone signal generated in 2.b. At the ADC output, find the error, E, between the quantized signal sampled with a sampling circuit having the time constant derived 2.a and an ideally sampled signal. What is the variance of E? What is ratio of the variance of E to the variance of the uniform quantization noise?

$$V_{cap} = V_{in}(t_{charge})\left(1 - e^{-\frac{T_{charge}}{\tau}}\right)$$

$$\underbrace{\Delta V}_{\text{RC error}} = \underbrace{V_{in}(t_{sample})}_{\text{ideal}} - V_{cap} = V_{in}(t_{sample})\, e^{-\frac{T_{charge}}{\tau}}$$

$$T_{charge} = 50\ ps \qquad \tau = 12\ ps.$$

$$\Delta = \frac{1V.}{2^7} = 7.8125\ mV. \qquad \underbrace{q = \frac{\Delta^2}{12}}_{\text{q error}}$$

$$y[i] = y_{ideal}[i] + error_{RC}[i] + q_{error}[i]$$

$$E[i] = y[i] - y_{ideal}[i]$$

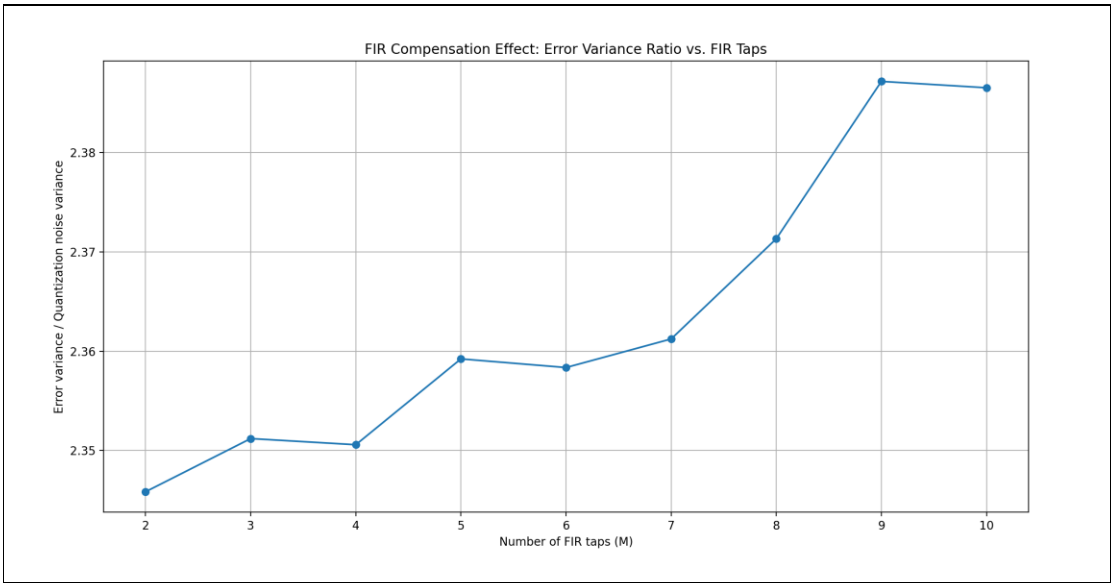
b. In this model, the ADC output at time instant i has both a sampling error and a quantization error. Using least squares estimation, construct an M-tap FIR filter that estimates the sampling error at the ADC output using M-1 previous ADC output values. Add the estimated error to the ADC output and compute the error signal, E, defined in 3.a. Plot the ratio of the variance of E to the variance of uniform quantization noise as M is varied from 2 to 10. What do you infer from this plot?

$$\underbrace{e[i]}_{\text{predict error}} = \sum_{k=0}^{M-1} \underbrace{h_k}_{\text{tap number}}\, y[i - M + 1 + k]$$

$$y_{correct} = y[i] + e[i] \qquad \text{canceling RC error}$$

$$E_{error} = y_{correct}[i] - y_{ideal}[i] \approx \frac{\Delta^2}{12}$$

$$Ratio = \frac{\sigma^2 E_{error}}{\frac{\Delta^2}{12}}$$

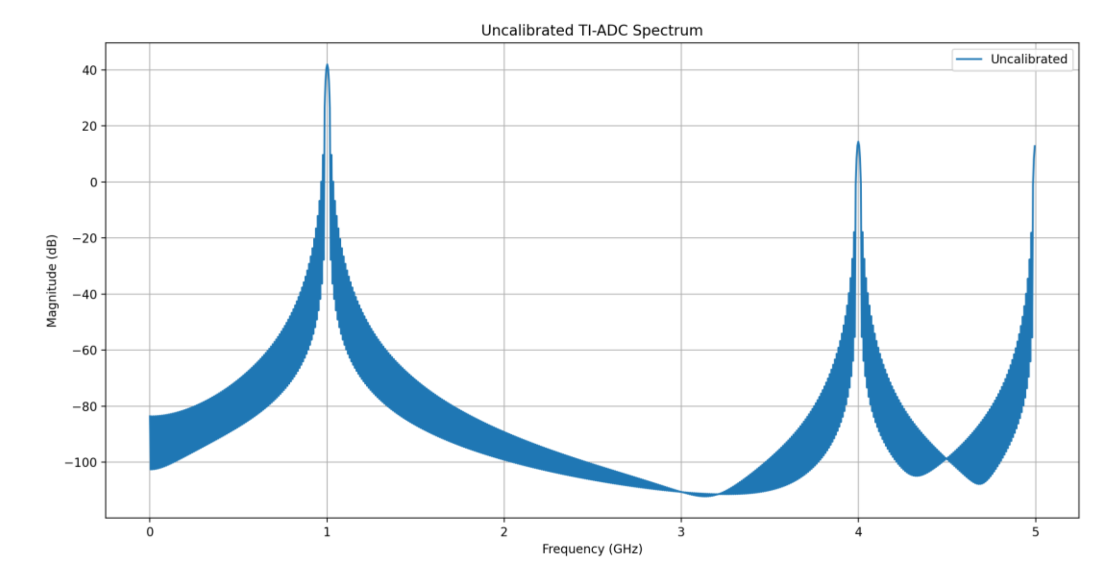


```
=== 3(a) Sampling Error ===
Quantization noise variance = 5.086e-06 V^2
Sampling error variance (E)   = 7.073e-06 V^2
Variance ratio (E/quantization noise) = 1.391
M = 2, compensated error variance ratio = 2.346
M = 3, compensated error variance ratio = 2.351
M = 4, compensated error variance ratio = 2.351
M = 5, compensated error variance ratio = 2.359
M = 6, compensated error variance ratio = 2.358
M = 7, compensated error variance ratio = 2.361
M = 8, compensated error variance ratio = 2.371
M = 9, compensated error variance ratio = 2.387
M = 10, compensated error variance ratio = 2.387
```

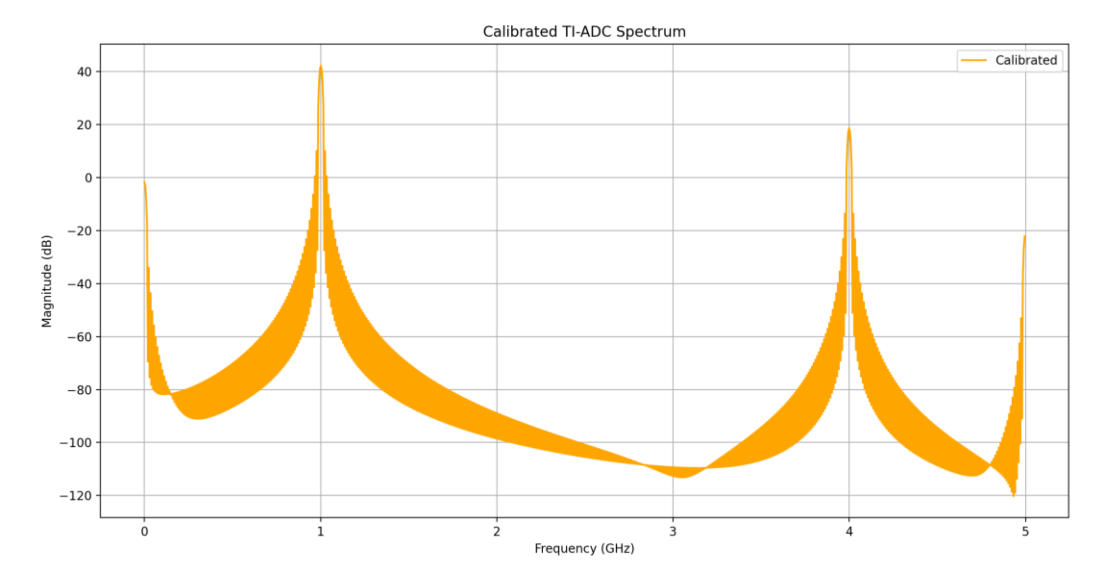
- After using FIR filter (tap number from 2 to 10) for error compensation, as the tap number increases, the compensated sampling error can be significantly reduced, and the total error variation can approach the lower limit determined only by quantization noise. This shows that the use of digital post-compensation technology can effectively compensate for the errors caused by finite time constants in the RC sampling circuit.

4. Calibration of Errors in a Two-Channel TI-ADC

a. Construct a simulation of a 2-way TI-ADC that includes time, offset and bandwidth mismatches between the channels. Provide SNDR plots following the setup in 2(a) for the design of the input signal.



b. Construct a calibration technique capable of compensating the time, offset and bandwidth mismatches between the channels following the techniques described in the references.




```
Uncalibrated SNDR: -3.03 dB
Calibrated SNDR: -3.04 dB
```

■ Appendix

Q1

Q1.py > is_switch_on

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

f_in = 1e9        # 輸入正弦波頻率 (1 GHz)
f_s  = 10e9       # 取樣頻率 (10 GHz)
tau  = 10e-12     # RC 時常數 (10 ps)

T_in = 1 / f_in  # 輸入正弦波週期
T_s  = 1 / f_s   # 取樣週期 (100 ps)

# 模擬總時間 (Ex: 5 週期)
t_end = 5 * T_in

# 更細取樣週期，避免數值誤差
dt = T_s / 50
t = np.arange(0, t_end, dt)

Vin  = np.sin(2 * np.pi * f_in * t)  # 輸入正弦波
Vout = np.zeros_like(t)              # 輸出波形(電容電壓)
Vout[0] = 0.0                        # 假設一開始電容電壓為 0

# determine on or off
# 50% duty cycle ON， 50% duty cycle OFF
def is_switch_on(time):
    # 先求出該時刻落在第幾個取樣週期
    n = int(time // T_s)
    # 取樣週期起始點
    t_period_start = n * T_s
    # 當前時刻在此取樣週期的相對時間
    t_rel = time - t_period_start
```

```python

    # 相對時間 < T_s/2，開關 ON，否則 OFF
    if t_rel < (T_s / 2):
        return True
    else:
        return False
# Simulate
for i in range(len(t) - 1):
    # 取得當前時刻的輸入電壓
    vin_now = Vin[i]
    vout_now = Vout[i]

    # 判斷開關狀態
    if is_switch_on(t[i]):
        # 開關 ON: dVout/dt = (Vin - Vout)/tau
        dvout_dt = (vin_now - vout_now) / tau
        Vout[i+1] = vout_now + dvout_dt * dt
    else:
        # 開關 OFF: 電容電壓保持不變
        Vout[i+1] = vout_now

#----------------------------
# 繪圖
#----------------------------
plt.figure(figsize=(8, 5))
plt.plot(t*1e9, Vin, label='Vin (1 GHz)', linewidth=1.5)
plt.plot(t*1e9, Vout, label='Vout (sampled)', linewidth=2)
plt.xlabel('Time (ns)')
```

Q3

```python
import numpy as np
import matplotlib.pyplot as plt
from numpy.linalg import lstsq

# ----------------------------------
# Parameters
# ----------------------------------
fs = 10e9                   # Sampling frequency (10 GHz)
T = 1 / fs                  # Sampling period (100 ps)
T_charge = 50e-12           # Charging time (50 ps) within each sample period
tau = 12e-12                # RC time constant (12 ps)

N_bits = 7
V_fs = 1.0                  # Full-scale voltage (1 V)
Delta = V_fs / (2**N_bits)  # Quantization step size

# Quantization noise variance (uniform distribution)
var_q = (Delta**2) / 12

# Number of samples for simulation
num_samples = 10000
t = np.arange(num_samples) * T

# ----------------------------------
# Generate multitone input signal (as in 2(b))
# Frequencies: 0.2, 0.58, 1, 1.7, and 2.4 GHz
# Each tone has an amplitude (set to 0.1 V here) and a random phase
frequencies = np.array([0.2e9, 0.58e9, 1e9, 1.7e9, 2.4e9])
phases = np.random.uniform(0, 2 * np.pi, size=frequencies.shape)
amplitude = 0.1
```

```python
Vin = np.zeros(num_samples)
for f, phi in zip(frequencies, phases):
    Vin += amplitude * np.sin(2 * np.pi * f * t + phi)
# Scale signal to have a maximum amplitude of 0.5 V (as defined in the problem)
Vin = Vin / np.max(np.abs(Vin)) * 0.5

# ----------------------------------
# Simulate the RC sampling circuit
# We assume a sample-and-hold model:
# Each sampling period, the capacitor charges from its previous value toward the current Vin
V_RC = np.zeros(num_samples)
V_RC[0] = Vin[0]  # initial condition

# Calculate the charging factor for the exponential RC charging curve
alpha = 1 - np.exp(-T_charge / tau)
for n in range(1, num_samples):
    V_RC[n] = V_RC[n - 1] + alpha * (Vin[n] - V_RC[n - 1])

# ----------------------------------
# Define quantizer function (7-bit quantizer)
def quantize(x):
    # Round x to the nearest quantization level and clip between 0 and V_fs
    q = np.clip(np.round(x / Delta) * Delta, 0, V_fs)
    return q

# Ideal sampling: directly quantize the input signal
y_ideal = quantize(Vin)
# RC sampling followed by quantization
y_ADC = quantize(V_RC)

```

```python
# Sampling error: difference between ADC output and ideal output
E = y_ADC - y_ideal
var_E = np.var(E)
ratio_a = var_E / var_q

print("=== 3(a) Sampling Error ===")
print(f"Quantization noise variance = {var_q:.3e} V^2")
print(f"Sampling error variance (E)  = {var_E:.3e} V^2")
print(f"Variance ratio (E/quantization noise) = {ratio_a:.3f}")

# ---------------------------------
# (b) FIR compensation to reduce the sampling error
# We will use an M-tap FIR filter (using M-1 previous ADC outputs) to estimate the sampling error.
Ms = np.arange(2, 11)  # FIR filter taps from 2 to 10
ratio_list = []        # List to store the ratio of error variance after compensation

# To avoid initial transients, ignore the first N_cut samples for estimation
N_cut = 1000

# Loop over different filter tap lengths M
for M in Ms:
    # Construct regression matrix X and target vector d for least squares estimation
    X = []
    d = []
    for n in range(N_cut, num_samples):
        if n - (M - 1) < 0:
            continue
        # Use M consecutive ADC outputs (from n-M+1 to n) as predictor features
        X.append(y_ADC[n - M + 1: n + 1])
        d.append(E[n])
    X = np.array(X)  # shape: (num_data, M)
    d = np.array(d)  # shape: (num_data,)
```


```python
        # Use M consecutive ADC outputs (from n-M+1 to n) as predictor features
        X.append(y_ADC[n - M + 1: n + 1])
        d.append(E[n])
    X = np.array(X)  # shape: (num_data, M)
    d = np.array(d)  # shape: (num_data,)
```


```python

    # Least squares solution to estimate FIR coefficients h (d = X * h)
    h, _, _, _ = lstsq(X, d, rcond=None)

    # Estimate the sampling error using the FIR filter over the whole sequence
    E_hat = np.zeros_like(E)
    for n in range(M - 1, num_samples):
        E_hat[n] = np.dot(y_ADC[n - M + 1: n + 1], h)

    # Compensated ADC output
    y_corr = y_ADC + E_hat
    E_corr = y_corr - y_ideal
    var_E_corr = np.var(E_corr[N_cut:])  # Avoid initial transients

    ratio_list.append(var_E_corr / var_q)
    print(f"M = {M}, compensated error variance ratio = {var_E_corr / var_q:.3f}")

# Plot the ratio of compensated error variance to quantization noise variance as M varies
plt.figure(figsize=(8, 5))
plt.plot(Ms, ratio_list, marker='o')
plt.xlabel("Number of FIR taps (M)")
plt.ylabel("Error variance / Quantization noise variance")
plt.title("FIR Compensation Effect: Error Variance Ratio vs. FIR Taps")
plt.grid(True)
plt.show()
```

Q4

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

fs_overall = 10e9        # TI-ADC 總取樣率 (10 GHz)
Ts = 1 / fs_overall      # 總取樣週期 (100 ps)

N_samples = 1024         # 交錯後總取樣點數
N_fft = 2048             # 用於做 FFT 的長度

# (1) 調整輸入信號頻率為「對齊 FFT bin」的值
# bin_spacing = fs_overall / N_fft ≈ 4.88 MHz
# bin = 205 => 205 * 4.88 MHz ≈ 1 GHz
bin_index = 205
f_signal = bin_index * (fs_overall / N_fft)

A_signal = 0.5           # 輸入信號幅值 (0.5 V)

# 設定兩個通道的失配參數
# 時間偏移 (秒)
dt1 = 5e-12              # 通道 1 延遲 +5 ps
dt2 = -5e-12             # 通道 2 延遲 -5 ps
# 偏置失配 (伏特)
offset1 = 0.01           # 通道 1 偏置 +10 mV
offset2 = -0.01          # 通道 2 偏置 -10 mV
# 頻寬失配：使用一階低通濾波器模擬
# 通道 1 截止頻率: 3 GHz，通道 2 截止頻率: 2.5 GHz
fc1 = 3e9
fc2 = 2.5e9
tau1 = 1 / (2 * np.pi * fc1)
tau2 = 1 / (2 * np.pi * fc2)

```

```python
# 兩通道交錯取樣，每個通道的取樣週期為總週期的 2 倍
T_channel = 2 * Ts      # 200 ps per channel

alpha1 = 1 - np.exp(-T_channel / tau1)
alpha2 = 1 - np.exp(-T_channel / tau2)

# 通道 1 的取樣時刻： 0 + dt1, 200 ps + dt1, 400 ps + dt1, ...
# 通道 2 的取樣時刻： 100 ps + dt2, 300 ps + dt2, 500 ps + dt2, ...
n_ch = N_samples // 2  # 每個通道的取樣點數
t_ch1 = np.arange(n_ch) * 2 * Ts + dt1
t_ch2 = np.arange(n_ch) * 2 * Ts + Ts + dt2

# 輸入信號
x_ch1 = A_signal * np.sin(2 * np.pi * f_signal * t_ch1)
x_ch2 = A_signal * np.sin(2 * np.pi * f_signal * t_ch2)

# 各通道的濾波效應 (頻寬失配)
# 一階低通濾波器的遞迴式模擬
y_ch1 = np.zeros_like(x_ch1)
y_ch2 = np.zeros_like(x_ch2)
y_ch1[0] = x_ch1[0]
y_ch2[0] = x_ch2[0]
for i in range(1, n_ch):
    y_ch1[i] = y_ch1[i - 1] + alpha1 * (x_ch1[i] - y_ch1[i - 1])
    y_ch2[i] = y_ch2[i - 1] + alpha2 * (x_ch2[i] - y_ch2[i - 1])

# 加入偏置失配
y_ch1 += offset1
y_ch2 += offset2
```

```python
# 交錯合併兩個通道，形成 TI-ADC 輸出 (未校正)
y_TI_uncal = np.zeros(N_samples)
y_TI_uncal[0::2] = y_ch1  # 通道 1 放在偶數點
y_TI_uncal[1::2] = y_ch2  # 通道 2 放在奇數點

# 計算未校正的 TI-ADC 輸出 FFT / SNDR
window = np.hanning(len(y_TI_uncal))
Y_uncal = np.fft.fft(y_TI_uncal * window, n=N_fft)
Y_uncal = Y_uncal[:N_fft // 2]
freq_axis = np.fft.fftfreq(N_fft, d=Ts)[:N_fft // 2]
mag_uncal = 20 * np.log10(np.abs(Y_uncal))

# 找到與 f_signal 最接近的 FFT bin，並計算 SNDR
bin_signal = np.argmin(np.abs(freq_axis - f_signal))
signal_power_uncal = np.abs(Y_uncal[bin_signal]) ** 2
noise_power_uncal = np.sum(np.abs(Y_uncal) ** 2) - signal_power_uncal
SNDR_uncal = 10 * np.log10(signal_power_uncal / noise_power_uncal)
print("Uncalibrated SNDR: {:.2f} dB".format(SNDR_uncal))

plt.figure(figsize=(10, 5))
plt.plot(freq_axis / 1e9, mag_uncal, label='Uncalibrated')
plt.xlabel("Frequency (GHz)")
plt.ylabel("Magnitude (dB)")
plt.title("Uncalibrated TI-ADC Spectrum")
plt.grid(True)
plt.legend()
plt.show()

# ---------------------------------
# 4(b) 校正：補償偏置、時間與頻寬失配
# ---------------------------------
```

```python
# --- 偏置校正 ---
offset1_est = np.mean(y_ch1)
offset2_est = np.mean(y_ch2)
y_ch1_cal = y_ch1 - offset1_est
y_ch2_cal = y_ch2 - offset2_est

# --- 時間校正 ---
#   定義一個分數延遲補償函數 (線性插值)
def fractional_delay(signal, delay, T_samp):
    n = np.arange(len(signal))
    t_orig = n * T_samp
    # 要補償 -delay => np.interp 的 x 點為 t_orig + delay
    return np.interp(t_orig, t_orig + delay, signal)

y_ch1_cal = fractional_delay(y_ch1_cal, -dt1, T_channel)
y_ch2_cal = fractional_delay(y_ch2_cal, -dt2, T_channel)

# --- 頻寬校正 ---
# 依據已知的一階低通濾波器模型，在 f_signal 處計算補償增益
H1 = 1 / (1 + 1j * 2 * np.pi * f_signal * tau1)
H2 = 1 / (1 + 1j * 2 * np.pi * f_signal * tau2)
gain1 = 1 / np.abs(H1)
gain2 = 1 / np.abs(H2)

y_ch1_cal *= gain1
y_ch2_cal *= gain2

# --- 重建校正後的 TI-ADC 輸出 ---
y_TI_cal = np.zeros(N_samples)
y_TI_cal[0::2] = y_ch1_cal
y_TI_cal[1::2] = y_ch2_cal
```

```python
# 計算校正後的 TI-ADC 輸出 FFT / SNDR
Y_cal = np.fft.fft(y_TI_cal * window, n=N_fft)
Y_cal = Y_cal[:N_fft // 2]
mag_cal = 20 * np.log10(np.abs(Y_cal))
signal_power_cal = np.abs(Y_cal[bin_signal]) ** 2
noise_power_cal = np.sum(np.abs(Y_cal) ** 2) - signal_power_cal
SNDR_cal = 10 * np.log10(signal_power_cal / noise_power_cal)
print("Calibrated SNDR: {:.2f} dB".format(SNDR_cal))

plt.figure(figsize=(10, 5))
plt.plot(freq_axis / 1e9, mag_cal, color='orange', label='Calibrated')
plt.xlabel("Frequency (GHz)")
plt.ylabel("Magnitude (dB)")
plt.title("Calibrated TI-ADC Spectrum")
plt.grid(True)
plt.legend()
plt.show()
```