Título de la Práctica: Diseño e Implementación de Filtros Digitales FIR con Arduino

Carrera: Ingeniería en Electrónica y Telecomunicaciones
Materia: Procesamiento Digital de Señales
Profesor: Alan David Blanco Miranda **Laboratorio:** Electrónica
Fecha: 31/03/2025 **Duración:** 3 hrs.

OBJETIVO DE LA ACTIVIDAD:

- Comprender los conceptos fundamentales de los filtros digitales FIR
- Diseñar e implementar diferentes tipos de filtros digitales (paso-bajas, paso-altas, paso-banda)
- Analizar el efecto de diferentes ventanas en el diseño de filtros FIR
- Aplicar filtros digitales a señales reales adquiridas mediante Arduino
- Comparar el desempeño de filtros FIR en aplicaciones prácticas

HERRAMIENTAS, MATERIAL Y/O REACTIVOS A UTILIZAR:

1. **Hardware:**
    - Generador de funciones (con capacidad hasta 5kHz)
    - Osciloscopio digital (mínimo 2 canales)
    - Arduino UNO o similar
    - Protoboard
    - Cables jumper
    - Resistencias: 2x 10kΩ, 2x 1kΩ, 2x 4.7 kΩ
    - Capacitor de 0.1µF
    - Cable USB
    - Computadora con Python y Arduino IDE instalados
    - Micrófono electret (opcional)
    - Potenciómetro 10kΩ (opcional para generación de señales variables)
    - LED (opcional para visualización de señales filtradas)

2. **Software:**
   - Arduino IDE (última versión)
   - Python 3.x con las siguientes bibliotecas:
     - numpy
     - matplotlib
     - scipy
     - pyserial
   - Editor de código (VS Code recomendado))

PARTE 1: SIMULACIÓN DE FILTROS DIGITALES EN PYTHON

1.1 Fundamentos de Filtros FIR e IIR

Crear un archivo Python llamado `filtros_conceptos.py` con el siguiente código para visualizar las diferencias fundamentales entre filtros FIR e IIR:

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

# Parámetros generales
fs = 1000  # Frecuencia de muestreo (Hz)
T = 1      # Duración de la señal (segundos)
N = T * fs  # Número de muestras
t = np.arange(0, T, 1/fs)  # Vector de tiempo

# Función para mostrar respuesta al impulso
def mostrar_respuesta_impulso(b, a, titulo, N_imp=100):
    # Generar impulso
    impulso = np.zeros(N_imp)
    impulso[0] = 1

    # Calcular respuesta al impulso
```

```
    respuesta = signal.lfilter(b, a, impulso)

    # Graficar
    plt.figure(figsize=(10, 6))
    plt.stem(np.arange(N_imp), respuesta)
    plt.title(f'Respuesta al Impulso - {titulo}')
    plt.xlabel('n [muestras]')
    plt.ylabel('Amplitud')
    plt.grid(True)

    return respuesta

# Función para mostrar respuesta en frecuencia
def mostrar_respuesta_frecuencia(b, a, titulo, fs=1000):
    # Calcular respuesta en frecuencia
    w, h = signal.freqz(b, a, worN=8000)
    f = w * fs / (2 * np.pi)  # Convertir a Hz

    # Graficar magnitud
    plt.figure(figsize=(12, 8))

    # Magnitud en escala lineal
    plt.subplot(2, 1, 1)
    plt.plot(f, np.abs(h))
    plt.title(f'Respuesta en Frecuencia - {titulo}')
    plt.xlabel('Frecuencia (Hz)')
    plt.ylabel('Magnitud')
    plt.grid(True)
    plt.xlim([0, fs/2])

    # Magnitud en dB
    plt.subplot(2, 1, 2)
```

```
    plt.plot(f, 20 * np.log10(np.abs(h) + 1e-10))  # Evitar log(0)
    plt.xlabel('Frecuencia (Hz)')
    plt.ylabel('Magnitud (dB)')
    plt.grid(True)
    plt.xlim([0, fs/2])
    plt.ylim([-80, 5])

    plt.tight_layout()

# 1. Filtro FIR Paso-Bajas (ventana rectangular)
fc = 100  # Frecuencia de corte (Hz)
nyq = fs / 2  # Frecuencia de Nyquist
M = 31  # Orden del filtro (M impar para tener simetría)

# Diseño mediante ventana rectangular
h_fir = signal.firwin(M, fc/nyq, window='boxcar')
print(f"Coeficientes del filtro FIR (Rectangular, Orden {M-1}):")
print(h_fir)

# 2. Filtro IIR Paso-Bajas (Butterworth)
orden_iir = 4
b_iir, a_iir = signal.butter(orden_iir, fc/nyq, 'low')
print(f"\nCoeficientes del filtro IIR (Butterworth, Orden {orden_iir}):")
print(f"Numerador: {b_iir}")
print(f"Denominador: {a_iir}")

# Mostrar respuestas al impulso
resp_fir = mostrar_respuesta_impulso(h_fir, 1, f'Filtro FIR Orden {M-1}')
resp_iir = mostrar_respuesta_impulso(b_iir, a_iir, f'Filtro IIR Orden
{orden_iir}')

# Mostrar respuestas en frecuencia
```

```
mostrar_respuesta_frecuencia(h_fir, 1, f'Filtro FIR Paso-Bajas (Rectangular, Orden {M-1})', fs)
mostrar_respuesta_frecuencia(b_iir, a_iir, f'Filtro IIR Paso-Bajas (Butterworth, Orden {orden_iir})', fs)

# Mostrar gráficos
plt.show()
```

Ejecute el código y observe:

- Las diferencias en la respuesta al impulso entre filtros FIR e IIR
- Las características de la respuesta en frecuencia de cada tipo de filtro
- La relación entre los coeficientes y la función de transferencia

1.2 Diseño de Filtros FIR con Diferentes Ventanas

Crear un archivo Python llamado `filtros_fir_ventanas.py` con el siguiente código:

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

# Parámetros de simulación
fs = 1000  # Frecuencia de muestreo (Hz)
fc = 100   # Frecuencia de corte (Hz)
nyq = fs / 2  # Frecuencia de Nyquist
M = 51     # Orden del filtro (impar para simetría)

# Definir diferentes ventanas
ventanas = {
    'Rectangular': 'boxcar',
    'Hanning': 'hann',
    'Hamming': 'hamming',
    'Blackman': 'blackman',
    'Kaiser (β=4.0)': ('kaiser', 4.0)
```

```
}

# Función para mostrar ventanas en el dominio del tiempo
def mostrar_ventanas(ventanas, M):
    plt.figure(figsize=(10, 6))
    n = np.arange(M)

    for nombre, ventana in ventanas.items():
        if isinstance(ventana, tuple):
            window = signal.get_window(ventana, M)
        else:
            window = signal.get_window(ventana, M)
        plt.plot(n, window, label=nombre)

    plt.title('Ventanas en el Dominio del Tiempo')
    plt.xlabel('n [muestras]')
    plt.ylabel('Amplitud')
    plt.legend()
    plt.grid(True)

# Función para diseñar y analizar filtros FIR con diferentes ventanas
def analizar_filtros_fir(ventanas, M, fc, fs):
    nyq = fs / 2
    filtros = {}

    # Diseñar filtros con diferentes ventanas
    for nombre, ventana in ventanas.items():
        if isinstance(ventana, tuple):
            h = signal.firwin(M, fc/nyq, window=ventana)
        else:
            h = signal.firwin(M, fc/nyq, window=ventana)
        filtros[nombre] = h
```

```
    # Calcular y graficar respuestas en frecuencia
    plt.figure(figsize=(12, 10))
    w = np.linspace(0, np.pi, 1000)

    for i, (nombre, h) in enumerate(filtros.items(), 1):
        w, H = signal.freqz(h, 1, worN=1000)
        f = w * fs / (2 * np.pi)

        # Magnitud en escala lineal
        plt.subplot(len(filtros), 2, 2*i-1)
        plt.plot(f, np.abs(H))
        plt.title(f'Filtro FIR con Ventana {nombre}')
        plt.xlabel('Frecuencia (Hz)')
        plt.ylabel('Magnitud')
        plt.grid(True)
        plt.xlim([0, fs/2])

        # Magnitud en dB
        plt.subplot(len(filtros), 2, 2*i)
        plt.plot(f, 20 * np.log10(np.abs(H) + 1e-10))
        plt.xlabel('Frecuencia (Hz)')
        plt.ylabel('Magnitud (dB)')
        plt.grid(True)
        plt.xlim([0, fs/2])
        plt.ylim([-100, 5])

    plt.tight_layout()

    return filtros

# Mostrar las ventanas en el dominio del tiempo
```

```
mostrar_ventanas(ventanas, M)

# Diseñar y analizar filtros FIR con diferentes ventanas
filtros = analizar_filtros_fir(ventanas, M, fc, fs)

# Aplicar filtros a una señal de prueba
# Crear una señal de prueba con múltiples componentes frecuenciales
t = np.arange(0, 1, 1/fs)
f1, f2, f3 = 50, 150, 300  # Frecuencias de componentes (Hz)
x = np.sin(2*np.pi*f1*t) + 0.5*np.sin(2*np.pi*f2*t) +
0.25*np.sin(2*np.pi*f3*t)

plt.figure(figsize=(12, 6))
plt.subplot(3, 1, 1)
plt.plot(t[:200], x[:200])
plt.title('Señal Original (Primeras 200 muestras)')
plt.xlabel('Tiempo (s)')
plt.ylabel('Amplitud')
plt.grid(True)

# Aplicar diferentes filtros y comparar resultados
for i, (nombre, h) in enumerate(filtros.items(), 2):
    if i > 3:  # Mostrar solo dos filtros por claridad
        break

    y = signal.lfilter(h, 1, x)

    plt.subplot(3, 1, i)
    plt.plot(t[:200], y[:200])
    plt.title(f'Señal Filtrada con Ventana {nombre}')
    plt.xlabel('Tiempo (s)')
    plt.ylabel('Amplitud')
```

```
    plt.grid(True)

plt.tight_layout()

# Análisis en el dominio de la frecuencia
plt.figure(figsize=(12, 6))

# FFT de la señal original
X = np.fft.fft(x)
freq = np.fft.fftfreq(len(x), 1/fs)
plt.subplot(3, 1, 1)
plt.plot(freq[:len(freq)//2], np.abs(X[:len(X)//2])/len(X))
plt.title('Espectro de la Señal Original')
plt.xlabel('Frecuencia (Hz)')
plt.ylabel('Magnitud')
plt.grid(True)
plt.xlim([0, fs/2])

# FFT de las señales filtradas
for i, (nombre, h) in enumerate(filtros.items(), 2):
    if i > 3:  # Mostrar solo dos filtros por claridad
        break

    y = signal.lfilter(h, 1, x)
    Y = np.fft.fft(y)

    plt.subplot(3, 1, i)
    plt.plot(freq[:len(freq)//2], np.abs(Y[:len(Y)//2])/len(Y))
    plt.title(f'Espectro de la Señal Filtrada con Ventana {nombre}')
    plt.xlabel('Frecuencia (Hz)')
    plt.ylabel('Magnitud')
    plt.grid(True)
```

```
    plt.xlim([0, fs/2])

plt.tight_layout()
plt.show()

# Guardar coeficientes del filtro Hamming para usarlo más adelante
np.save('coef_fir_hamming.npy', filtros['Hamming'])
print(f"Coeficientes del filtro FIR con ventana Hamming guardados en 'coef_fir_hamming.npy'")
```

Ejecute el código y observe:

- Las formas de las diferentes ventanas en el dominio del tiempo
- El efecto de cada ventana en la respuesta en frecuencia del filtro
- La relación entre el tipo de ventana y la atenuación en la banda de rechazo
- El comportamiento de cada filtro al aplicarse a una señal de prueba

PARTE 2: IMPLEMENTACIÓN DE FILTROS DIGITALES CON ARDUINO

2.1 Montaje del Circuito

1. Configuración del Generador de Funciones
    - Encender el generador
    - Configurar:
        - Forma de onda: Senoidal
        - Frecuencia: 100 Hz (inicialmente)
        - Amplitud: 2Vpp
        - Offset: +1V (para mantener la señal positiva)
2. Montaje del Circuito

```
[Generador] ---> [R1 10kΩ] ---> |punto A| ---> [R2 10kΩ] ---> GND
                                |punto A| ---> [C1 0.1µF] ---> GND
                                 |punto A| ---> Arduino A0
```

3. Verificación con Osciloscopio
    - Conectar Canal 1 al punto A
    - Configurar:
        - Base de tiempo: 2ms/div
        - Voltaje: 500mV/div

- Verificar que la señal:
  - Sea visible y estable
  - No exceda 5V pico a pico
  - Mantenga su forma senoidal

2.2 Código Arduino para Adquisición y Filtrado

1. Crear un archivo en Arduino IDE llamado `filtrado_digital.ino` con el siguiente código:

```
const int analogPin = A0;      // Pin analógico de entrada
const int ledPin = 9;          // Pin PWM para visualización (opcional)

// Parámetros de muestreo
const unsigned long SAMPLE_PERIOD = 1000;  // Microsegundos entre muestras (1kHz)
const int BUFFER_SIZE = 64;                // Tamaño del buffer (ajustar según memoria disponible)

// Buffers para almacenar muestras
int sample_buffer[BUFFER_SIZE];            // Buffer para muestras de entrada
int filtered_buffer[BUFFER_SIZE];          // Buffer para muestras filtradas
int buffer_index = 0;                      // Índice del buffer
bool buffer_full = false;                  // Indicador de buffer lleno

// Parámetros para filtro FIR paso-bajas (coeficientes precalculados en Python)
// Filtro FIR Hamming de orden 10, fc=100Hz @ fs=1kHz
const int FIR_ORDER = 10;
float fir_coeffs[FIR_ORDER + 1] = {
  0.0087, 0.0279, 0.0741, 0.1348, 0.1932, 0.2123,
  0.1932, 0.1348, 0.0741, 0.0279, 0.0087
};
```

```
float fir_buffer[FIR_ORDER + 1] = {0}; // Buffer circular para el filtro FIR


// Variable para seleccionar tipo de filtro
int filter_type = 0;  // 0: Sin filtro, 1: FIR

void setup() {
  Serial.begin(115200);       // Iniciar comunicación serial
  pinMode(ledPin, OUTPUT);    // Configurar pin de LED como salida

  Serial.println("Sistema de Adquisición y Filtrado Digital");
  Serial.println("Comandos disponibles:");
  Serial.println("0: Sin filtro");
  Serial.println("1: Filtro FIR");
  Serial.println("s: Enviar buffer completo");

  analogReference(DEFAULT);
  delay(1000);                // Estabilización
}

void loop() {
  static unsigned long last_sample = 0;
  unsigned long current_time = micros();

  // Verificar comandos seriales
  if (Serial.available() > 0) {
    char cmd = Serial.read();
    if (cmd >= '0' && cmd <= '2') {
      filter_type = cmd - '0';
      Serial.print("Filtro cambiado a tipo: ");
      Serial.println(filter_type);
```

```
  } else if (cmd == 's' || cmd == 'S') {
    send_buffer();
  }
}

// Muestrear a la frecuencia especificada
if (current_time - last_sample >= SAMPLE_PERIOD) {
  // Leer valor analógico
  int sensor_value = analogRead(analogPin);

  // Aplicar filtro según tipo seleccionado
  int filtered_value = 0;

  switch(filter_type) {
    case 0:  // Sin filtro
      filtered_value = sensor_value;
      break;
    case 1:  // Filtro FIR
      filtered_value = apply_fir_filter(sensor_value);
      break;

  }

  // Visualización opcional mediante PWM
  analogWrite(ledPin, map(filtered_value, 0, 1023, 0, 255));

  // Guardar en buffer
  if (buffer_index < BUFFER_SIZE) {
    sample_buffer[buffer_index] = sensor_value;
    filtered_buffer[buffer_index] = filtered_value;
    buffer_index++;
```

```
      if (buffer_index >= BUFFER_SIZE) {
        buffer_full = true;
        Serial.println("Buffer lleno. Enviar 's' para ver datos.");
      }
    }

    last_sample = current_time;
  }
}

// Implementación del filtro FIR
int apply_fir_filter(int new_sample) {
  // Desplazar valores en el buffer
  for (int i = FIR_ORDER; i > 0; i--) {
    fir_buffer[i] = fir_buffer[i-1];
  }

  // Añadir nueva muestra al inicio del buffer
  fir_buffer[0] = new_sample;

  // Aplicar coeficientes
  float result = 0;
  for (int i = 0; i <= FIR_ORDER; i++) {
    result += fir_coeffs[i] * fir_buffer[i];
  }

  return (int)result;
}


// Enviar buffer completo por puerto serial
void send_buffer() {
```

```
  if (!buffer_full) {
    Serial.println("Buffer no lleno. Esperando más muestras...");
    return;
  }

  // Enviar formato CSV: índice, original, filtrada
  Serial.println("index,original,filtered");
  for (int i = 0; i < BUFFER_SIZE; i++) {
    Serial.print(i);
    Serial.print(",");
    Serial.print(sample_buffer[i]);
    Serial.print(",");
    Serial.println(filtered_buffer[i]);
  }

  // Reiniciar buffer
  buffer_index = 0;
  buffer_full = false;
  Serial.println("Buffer enviado y reiniciado");
}
```

2. Cargar el código al Arduino y abrir el Monitor Serial
    - Configurar velocidad a 115200 baudios
    - Enviar los comandos para cambiar entre tipos de filtro
    - Observar los datos adquiridos y filtrados

2.3 Script Python para Visualizar Datos de Arduino

Crear un archivo Python llamado `visualizar_arduino.py` para recibir y graficar los datos del Arduino:

```
import serial
import time
import numpy as np
import matplotlib.pyplot as plt
```

```
from scipy import signal

def adquirir_datos(puerto='COM3', baudrate=115200, timeout=10):
    """
    Adquiere datos del Arduino
    """
    try:
        # Configurar puerto serial
        ser = serial.Serial(puerto, baudrate)
        print(f"Conectado al puerto: {puerto}")
        time.sleep(2)  # Esperar inicialización

        # Solicitar datos enviando 's'
        ser.write(b's')
        print("Solicitando datos del buffer...")

        # Esperar y leer datos
        inicio = time.time()
        datos_originales = []
        datos_filtrados = []
        encabezado_leido = False

        while (time.time() - inicio) < timeout:
            if ser.in_waiting:
                linea = ser.readline().decode().strip()

                if linea == "Buffer enviado y reiniciado":
                    break

                if encabezado_leido:
                    try:
                        partes = linea.split(',')
```

```
                    if len(partes) == 3:
                        datos_originales.append(int(partes[1]))
                        datos_filtrados.append(int(partes[2]))
                except:
                    pass  # Ignorar líneas mal formadas

            if linea == "index,original,filtered":
                encabezado_leido = True

        ser.close()
        return np.array(datos_originales), np.array(datos_filtrados)

    except Exception as e:
        print(f"Error al conectar con Arduino: {e}")
        return None, None

def cambiar_filtro(puerto='COM3', baudrate=115200, tipo_filtro=0):
    """
    Cambia el tipo de filtro en Arduino
    tipo_filtro: 0 (sin filtro), 1 (FIR), 2 (IIR)
    """
    try:
        ser = serial.Serial(puerto, baudrate)
        print(f"Conectado al puerto: {puerto}")
        time.sleep(2)  # Esperar inicialización

        # Enviar comando para cambiar filtro
        ser.write(str(tipo_filtro).encode())
        time.sleep(0.5)

        # Leer respuesta
        while ser.in_waiting:
```

```
            print(ser.readline().decode().strip())

            ser.close()
            return True
        except Exception as e:
            print(f"Error al conectar con Arduino: {e}")
            return False

def analizar_datos(originales, filtradas, fs=1000):
    """
    Analiza y grafica los datos adquiridos
    """
    if originales is None or filtradas is None:
        return

    t = np.arange(len(originales)) / fs  # Vector de tiempo

    # Gráficas en el dominio del tiempo
    plt.figure(figsize=(12, 8))
    plt.subplot(2, 1, 1)
    plt.plot(t, originales)
    plt.title('Señal Original')
    plt.xlabel('Tiempo (s)')
    plt.ylabel('Amplitud (ADC)')
    plt.grid(True)

    plt.subplot(2, 1, 2)
    plt.plot(t, filtradas)
    plt.title('Señal Filtrada')
    plt.xlabel('Tiempo (s)')
    plt.ylabel('Amplitud (ADC)')
    plt.grid(True)
```

```
plt.tight_layout()

# Gráficas en el dominio de la frecuencia
plt.figure(figsize=(12, 8))

# FFT de la señal original
X = np.fft.fft(originales)
freq = np.fft.fftfreq(len(originales), 1/fs)

plt.subplot(2, 1, 1)
plt.plot(freq[:len(freq)//2], np.abs(X[:len(X)//2])/len(X))
plt.title('Espectro de la Señal Original')
plt.xlabel('Frecuencia (Hz)')
plt.ylabel('Magnitud')
plt.grid(True)
plt.xlim([0, fs/2])

# FFT de la señal filtrada
Y = np.fft.fft(filtradas)

plt.subplot(2, 1, 2)
plt.plot(freq[:len(freq)//2], np.abs(Y[:len(Y)//2])/len(Y))
plt.title('Espectro de la Señal Filtrada')
plt.xlabel('Frecuencia (Hz)')
plt.ylabel('Magnitud')
plt.grid(True)
plt.xlim([0, fs/2])

plt.tight_layout()
plt.show()
```

```
def main():
    puerto = input("Ingrese el puerto COM (default: COM3): ") or "COM3"

    print("\n--- SISTEMA DE ANÁLISIS DE FILTROS DIGITALES ---")
    print("1. Adquirir datos sin filtro")
    print("2. Adquirir datos con filtro FIR")
    print("3. Salir")

    while True:
        opcion = input("\nSeleccione una opción: ")

        if opcion == '1':
            cambiar_filtro(puerto, tipo_filtro=0)
            time.sleep(1)
            orig, filt = adquirir_datos(puerto)
            analizar_datos(orig, filt)

        elif opcion == '2':
            cambiar_filtro(puerto, tipo_filtro=1)
            time.sleep(1)
            orig, filt = adquirir_datos(puerto)
            analizar_datos(orig, filt)

        elif opcion == '3':
            print("Saliendo...")
            break

        else:
            print("Opción no válida. Intente de nuevo.")

if __name__ == "__main__":
    main()
```

Ejecute el script Python y:

1. Conecte con el Arduino
2. Seleccione el tipo de filtro a aplicar
3. Adquiera y visualice los datos
4. Analice el comportamiento de los filtros en señales reales

2.4 Experimentos con Señales Reales

1. **Experimento 1: Barrido de Frecuencia**
   - Configure el generador de funciones para realizar un barrido de frecuencia (de 10Hz a 500Hz)
   - Adquiera datos con diferentes tipos de filtro
   - Observe el comportamiento de los filtros en diferentes rangos de frecuencia
2. **Experimento 2: Filtrado de Ruido**
   - Añada una fuente de ruido a la señal (puede acercar un teléfono móvil o manipular cables cercanos)
   - Adquiera datos con y sin filtro
   - Compare la capacidad de cada filtro para eliminar el ruido
3. **Experimento 3: Generación de Señales con Arduino** Modificar el código Arduino para generar señales además de filtrarlas:

```
// Añadir al inicio del archivo filtrado_digital.ino:

// Parámetros para generación de señales
int signal_type = 0;  // 0: Externa, 1: Seno, 2: Cuadrada, 3: Triangular
float signal_freq = 50.0;  // Frecuencia en Hz
int signal_amp = 500;      // Amplitud (0-512)
int signal_offset = 512;   // Offset (0-512)
unsigned long signal_phase = 0;  // Fase actual en microsegundos

// Añadir al setup():
Serial.println("Generación de señales:");
Serial.println("e: Señal externa (A0)");
Serial.println("s: Señal senoidal");
Serial.println("q: Señal cuadrada");
Serial.println("t: Señal triangular");
```

```
Serial.println("f###: Cambiar frecuencia a ### Hz");
Serial.println("a###: Cambiar amplitud a ###");

// Modificar la sección de verificación de comandos seriales:
if (Serial.available() > 0) {
  char cmd = Serial.read();
  if (cmd >= '0' && cmd <= '2') {
    filter_type = cmd - '0';
    Serial.print("Filtro cambiado a tipo: ");
    Serial.println(filter_type);
  } else if (cmd == 's' || cmd == 'S') {
    send_buffer();
  } else if (cmd == 'e' || cmd == 'E') {
    signal_type = 0;
    Serial.println("Señal externa seleccionada");
  } else if (cmd == 's' || cmd == 'S') {
    signal_type = 1;
    Serial.println("Señal senoidal seleccionada");
  } else if (cmd == 'q' || cmd == 'Q') {
    signal_type = 2;
    Serial.println("Señal cuadrada seleccionada");
  } else if (cmd == 't' || cmd == 'T') {
    signal_type = 3;
    Serial.println("Señal triangular seleccionada");
  } else if (cmd == 'f') {
    // Leer frecuencia
    String freq_str = "";
    while (Serial.available() > 0) {
      char c = Serial.read();
      if (isDigit(c)) freq_str += c;
      delay(2);
    }
```

```
    if (freq_str.length() > 0) {
      signal_freq = freq_str.toFloat();
      Serial.print("Frecuencia cambiada a: ");
      Serial.print(signal_freq);
      Serial.println(" Hz");
    }
  } else if (cmd == 'a') {
    // Leer amplitud
    String amp_str = "";
    while (Serial.available() > 0) {
      char c = Serial.read();
      if (isDigit(c)) amp_str += c;
      delay(2);
    }
    if (amp_str.length() > 0) {
      signal_amp = amp_str.toInt();
      Serial.print("Amplitud cambiada a: ");
      Serial.println(signal_amp);
    }
  }
}

// Modificar la sección de muestreo:
if (current_time - last_sample >= SAMPLE_PERIOD) {
  // Leer o generar valor según el tipo de señal
  int sensor_value;

  switch(signal_type) {
    case 0:  // Señal externa
      sensor_value = analogRead(analogPin);
      break;
    case 1:  // Señal senoidal
```

```
      signal_phase += SAMPLE_PERIOD;
      if (signal_phase >= (1000000 / signal_freq))
        signal_phase = 0;

      float phase_rad = (2.0 * PI * signal_phase * signal_freq) /
1000000.0;
      sensor_value = signal_offset + signal_amp * sin(phase_rad);
      break;
    case 2:  // Señal cuadrada
      signal_phase += SAMPLE_PERIOD;
      if (signal_phase >= (1000000 / signal_freq))
        signal_phase = 0;

      if (signal_phase < (500000 / signal_freq))
        sensor_value = signal_offset + signal_amp;
      else
        sensor_value = signal_offset - signal_amp;
      break;
    case 3:  // Señal triangular
      signal_phase += SAMPLE_PERIOD;
      if (signal_phase >= (1000000 / signal_freq))
        signal_phase = 0;

      float phase_norm = (float)signal_phase / (1000000.0 / signal_freq);
      if (phase_norm < 0.5)
        sensor_value = signal_offset - signal_amp + 4 * signal_amp *
phase_norm;
      else
        sensor_value = signal_offset + 3 * signal_amp - 4 * signal_amp *
phase_norm;
      break;
  }
```

```
  // Continuar con el filtrado como antes...
}
```

- Experimente con diferentes formas de onda generadas por el Arduino
- Aplique filtros y observe el comportamiento

4. **Experimento 4: Procesamiento de Audio (Opcional)** Si se dispone de un micrófono electret:

```
[Micrófono] ---> [R1 10kΩ] ---> +5V
                 |
                 +---> [C1 0.1µF] ---> Arduino A0
                 |
                 +---> [R2 10kΩ] ---> GND
```

   - Adquiera señales de audio y aplique diferentes filtros
   - Observe el efecto del filtrado en el espectro de frecuencia del audio
   - Experimente con filtros paso-banda para extraer componentes específicas

PARTE 3: ANÁLISIS DE RESULTADOS Y EJERCICIOS

3.1 Análisis de Filtros FIR

Para cada tipo de filtro implementado, analice:

1. **Respuesta en Frecuencia**:
   - ¿Qué tipo de filtro (FIR) proporciona una mayor pendiente en la banda de transición?
   - ¿Cómo afecta el orden del filtro a la selectividad en frecuencia?
   - Compare la atenuación en la banda de rechazo entre diferentes tipos de ventanas
2. **Respuesta de Fase**:
   - ¿Qué tipo de filtro proporciona una respuesta de fase más lineal?
   - ¿Qué implicaciones tiene esto en aplicaciones prácticas?
3. **Eficiencia Computacional**:
   - Compare el tiempo de procesamiento entre filtros FIR
   - ¿Qué tipo de filtro es más eficiente para un mismo nivel de rendimiento?

3.2 Ejercicios Propuestos

1. **Diseño de Filtros para Aplicaciones Específicas**:

- Diseñe un filtro paso-banda para extraer una señal de 100 Hz en presencia de ruido
- Diseñe un filtro paso-altas para detectar transiciones rápidas en una señal

2. **Optimización de Parámetros**:
   - Experimente con diferentes órdenes de filtro para encontrar el equilibrio óptimo entre rendimiento y carga computacional
   - Explore diferentes tipos de ventanas y su efecto en el rendimiento del filtro
3. **Aplicaciones Prácticas**:
   - Detección de tonos específicos en una señal de audio
   - Eliminación de ruido en señales biomédicas
   - Diseño de un ecualizador simple para procesar audio

ANÁLISIS DE RESULTADOS

Para cada experimento, analice:

1. **Comparación entre Ventanas**:
   - ¿Qué ventana ofrece la mejor resolución espectral?
   - ¿Qué ventana tiene mejor supresión de lóbulos secundarios?
   - ¿Cuál es el compromiso entre resolución y fuga espectral?
2. **Análisis del Efecto del Orden del Filtro**:
   - ¿Cómo afecta el orden del filtro a la precisión en la banda de paso?
   - ¿Existe un punto óptimo entre complejidad computacional y calidad?
3. **Evaluación de Capacidad de Filtrado**:
   - ¿Qué tipo de filtro funciona mejor para eliminar ruido de alta frecuencia?
   - ¿Qué filtro es más efectivo para separar componentes frecuenciales cercanas?

PREGUNTAS DE COMPRENSIÓN

1. ¿Cuáles son las principales diferencias entre filtros FIR en términos de respuesta al impulso, estabilidad y eficiencia?
2. Explique cómo afecta el tipo de ventana al diseño de filtros FIR y qué ventajas ofrece cada tipo de ventana.
3. ¿Qué consideraciones deben tenerse en cuenta al implementar filtros digitales en sistemas con recursos limitados como Arduino?
4. ¿Cómo se relaciona la frecuencia de muestreo con la frecuencia máxima que puede ser procesada por un filtro digital?
5. Explique el concepto de "fuga espectral" y cómo puede mitigarse mediante el uso de ventanas.
6. ¿Cómo afecta el retardo de grupo a las aplicaciones en tiempo real y qué tipo de filtro minimiza este efecto?

7. ¿Qué problemas pueden surgir al implementar filtros digitales de alto orden en sistemas embebidos?

CRITERIOS DE EVALUACIÓN

1. Implementación correcta de los filtros digitales (25%)
2. Análisis de resultados y comprensión de conceptos (25%)
3. Desarrollo experimental y mediciones (25%)
4. Respuestas a las preguntas de comprensión y ejercicios propuestos (25%)

OBSERVACIONES:

- Tomar fotografías del montaje experimental
- Guardar todas las gráficas generadas en Python para el reporte
- Documentar cualquier observación relevante durante los experimentos
- Mantener respaldo digital del código utilizado

REFERENCIAS Y RECURSOS ADICIONALES

1. Documentación de Arduino:
    - Referencia AnalogRead: https://www.arduino.cc/reference/en/language/functions/analog-io/analogread/
    - Comunicación Serial: https://www.arduino.cc/reference/en/language/functions/communication/serial/
2. Documentación de Python:
    - NumPy Documentation: https://numpy.org/doc/
    - SciPy Signal Processing: https://docs.scipy.org/doc/scipy/reference/signal.html
    - Matplotlib Documentation: https://matplotlib.org/stable/contents.html
3. Teoría de Filtros Digitales:
    - Oppenheim, A. V., & Schafer, R. W. (2010). Discrete-time signal processing. Pearson.
    - Smith, S. W. (1997). The scientist and engineer's guide to digital signal processing. California Technical Pub.