JITTER

Brayan Steven Mendivelso Pérez
est.brayan.mendive@unimilitar.edu.co
Docente: José de Jesús Rúgeles

***Resumen*— En este laboratorio de Comunicaciones Digitales se trabajó con la tarjeta Raspberry Pi Pico, configurando un generador de señales para producir una onda senoidal de 200 Hz, 1.2 Vpp y una componente DC de 1.6 V, la cual fue muestreada mediante el programa ADC_testing.py. Se verificó la señal con el osciloscopio y se obtuvieron archivos de datos en el dominio del tiempo y la frecuencia, los cuales fueron graficados en MATLAB para analizar el comportamiento.**

Posteriormente, se realizaron variaciones en el número de puntos de la FFT (64, 128, 256, 512, 1024 y 2048) y en la frecuencia de la señal de entrada (100, 200, 900y 1800 Hz), observando los efectos sobre la resolución espectral y la representación en el dominio de la frecuencia.

En la segunda parte, se diseñó un programa alternativo para el muestreo estable de la señal, evaluando limitaciones del hardware, la importancia del jitter en el proceso de codificación y alternativas para optimizar el muestreo con la Raspberry Pi Pico 2W.

Finalmente, se analizó la implementación de la tasa de muestreo dentro del código, así como el uso de la ventana Hanning para mejorar el análisis espectral. Los resultados evidencian la relación entre el número de puntos de la FFT, la frecuencia de muestreo y la calidad de la representación de la señal, destacando la relevancia de un muestreo adecuado en sistemas digitales de comunicación.

***Abstract*— In this Digital Communications laboratory, the Raspberry Pi Pico board was used along with a signal generator configured to produce a 200 Hz sinusoidal waveform with 1.2 Vpp and a 1.6 V DC component. The signal was sampled using the ADC_testing.py program, verified with an oscilloscope, and the resulting time and frequency domain data files were plotted in MATLAB for analysis.**

Different FFT sizes (64, 128, 256, 512, 1024, and 2048) and input frequencies (100, 200, 900 and 1800 Hz) were tested, allowing the observation of their effects on spectral resolution and signal representation in the frequency domain.

In the second part, an alternative program was designed to achieve stable sampling, considering hardware limitations, the impact of jitter on source coding, and possible strategies to optimize sampling with the Raspberry Pi Pico 2W.

Finally, the sampling rate implementation within the code was analyzed, as well as the role of the Hanning window in improving spectral analysis. The results highlight the relationship between FFT size, sampling frequency, and signal representation quality, emphasizing the importance of proper sampling in digital communication systems.

I. Introducción

El presente laboratorio de Comunicaciones Digitales tiene como propósito analizar el proceso de muestreo y digitalización de señales utilizando la tarjeta Raspberry Pi Pico como conversor analógico-digital (ADC). Para ello, se generó una señal senoidal con parámetros definidos y se verificó su comportamiento tanto en el dominio del tiempo como en el de la frecuencia, aplicando técnicas de análisis mediante la Transformada Rápida de Fourier (FFT).

Durante la práctica, se exploró la influencia de factores como la frecuencia de muestreo, el número de puntos de la FFT y las variaciones en la frecuencia de entrada sobre la calidad de la representación de la señal. Asimismo, se evaluaron conceptos fundamentales como el jitter y su impacto en la codificación de la fuente, además de implementar un programa alternativo que permitiera mejorar la estabilidad del muestreo.

Este trabajo busca fortalecer la comprensión de los principios de muestreo digital y su relevancia en los sistemas de comunicación, destacando la importancia de un diseño adecuado para garantizar una representación precisa de la información en el ámbito digital.

II. Señal 200 Hz

Para esta parte del laboratorio lo primero que vamos a hacer es configurar un generador de señales de 200 Hz, 1,2 Vpp y un voltaje offset de 1,6 voltios. Ahora vamos a verificar esta señal en el osciloscopio para evitar cualquier tipo de daño en el Raspberry Pi Pico 2w ya que sabemos que este solo maneja voltaje positiva.

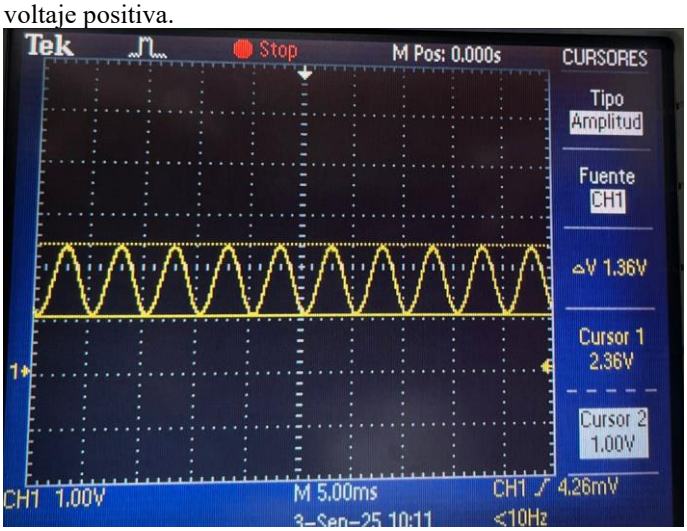

Ilustración 1 verificación de la señal en osciloscopio.

Una vez verificada la correcta generación de la señal, se procedió a conectarla al pin 26 de la Raspberry Pi Pico, asegurando también la conexión a tierra. Posteriormente, según las indicaciones del documento de laboratorio, se accedió al repositorio de GitHub proporcionado por el docente, en el cual se encuentra disponible el programa ADC_testing.py, utilizado para realizar el proceso de muestreo y análisis de la señal.

Se inició la práctica configurando la señal de entrada con una frecuencia de 200 Hz. A partir de esta señal, se realizaron pruebas utilizando seis diferentes valores del número de puntos de la FFT (64, 128, 256, 512, 1024 y 2048), con el propósito de analizar cómo varía la resolución espectral y la representación en el dominio de la frecuencia en función de este parámetro. Las gráficas obtenidas se organizaron mediante la función subplot, lo que permitió visualizar todos los resultados en una misma figura y facilitar la comparación de las diferencias entre los distintos casos, evidenciando el efecto del número de puntos en la resolución y el nivel de detalle del análisis espectral.

Vamos a comenzar con los gráficos de la FFT para esto usaremos el siguiente programa

```
% FFT 64 puntos
t1 = fft64_200.Frecuencia_Hz_;
m1 = fft64_200.Magnitud_V_;
% FFT 128 puntos
t2 = fft128_200_csv.Frecuencia_Hz_;
m2 = fft128_200_csv.Magnitud_V_;
% FFT 256 puntos
t3 = fft256_200.Frecuencia_Hz_;
m3 = fft256_200.Magnitud_V_;
% FFT 512 puntos
t4 = fft512_200.Frecuencia_Hz_;
m4 = fft512_200.Magnitud_V_;
% FFT 1024 puntos
t5 = fft.Frecuencia_Hz_;
m5 = fft.Magnitud_V_;
% FFT 2048 puntos
t6 = fft2048_200.Frecuencia_Hz_;
m6 = fft2048_200.Magnitud_V_;
% Crear figura con 6 subplots en una sola columna
figure;
subplot(6,1,1);
plot(t1, m1, 'b-'); hold on;
plot(t1, m1, 'ro');
title('FFT 64 puntos');
xlabel('Frequency (Hz)'); ylabel('Magnitude'); grid on;
subplot(6,1,2);
plot(t2, m2, 'b-'); hold on;
plot(t2, m2, 'ro');
```

Ilustración 2 programa FFT 200 Hz.

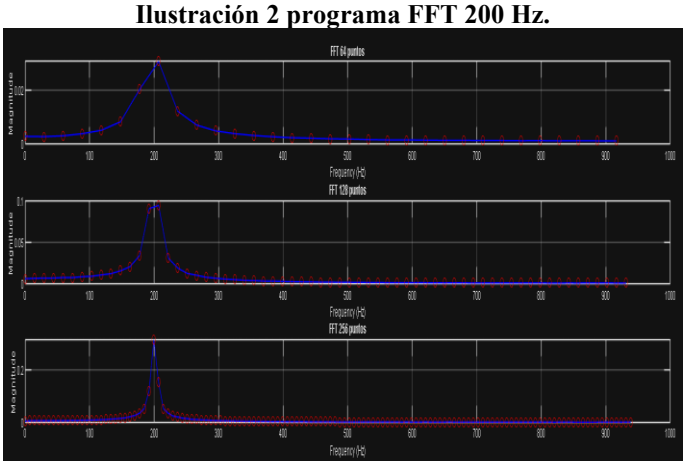

Ilustración 3 FFT parte 1.

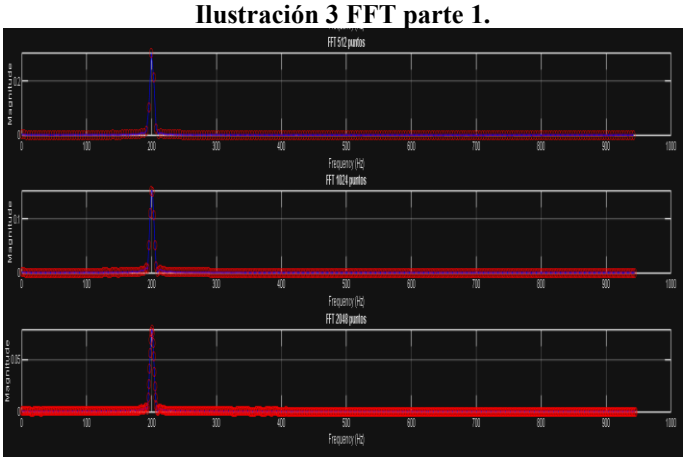

Ilustración 4 FFT parte 2.

En el gráfico se observa cómo el número de puntos de la FFT influye directamente en la resolución espectral de la señal de 200 Hz. Con un bajo número de puntos (64 y 128), el pico principal aparece ancho y poco definido, lo que dificulta identificar con precisión la frecuencia fundamental. A medida que aumenta el tamaño de la FFT (256 y 512 puntos), el espectro comienza a mostrar un mayor detalle y el pico en 200 Hz se hace más estrecho. Finalmente, con un número elevado de puntos (1024 y 2048), la representación es mucho más precisa, con un pico claramente definido y una mejor resolución en el dominio de la frecuencia. Esto evidencia que, entre más puntos se utilicen, la FFT tendrá una mayor cantidad de muestras en el eje de frecuencia, lo que mejora la exactitud del análisis, aunque a costa de un mayor esfuerzo de la Raspberry.

Ahora vamos a hacer lo mismo con las muestras que es la señal en el tiempo para esto vamos a usar la siguiente señal

```
t = muestras2048_200.Tiempo_s_
v = muestras2048_200.Voltaje_V_;
T = 0.005;
indices = t <= 5*T;
t_filtrado = t(indices);
v_filtrado = v(indices);
plot(t_filtrado, v_filtrado);
xlabel('Tiempo (s)');
ylabel('Voltaje (V)');
```

Ilustración 5 Código muestra.

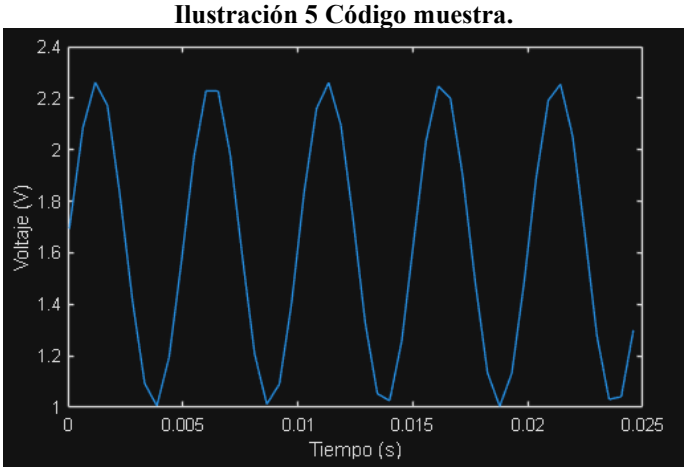

Ilustración 6 muestras.

En el dominio del tiempo solo se representó una señal muestreada, ya que la forma de la onda no cambia independientemente del número de puntos utilizados en la FFT. La señal corresponde a la misma onda senoidal de 200 Hz muestreada, y se limitó la visualización a cinco periodos para apreciar mejor su comportamiento. Posteriormente, las variaciones en el número de puntos de la FFT se analizaron únicamente en el dominio de la frecuencia, donde sí se observan diferencias en la resolución espectral y el detalle de la representación.

III. 100 Hz.

Ahora vamos a cambiar la frecuencia en el generador de señales a 100 Hz y mirar sus cambios en la FFT, vamos a usar un código parecido lo único que cambiamos del código anterior de la FFT es lo siguiente.

```
% FFT 64 puntos
t1 = fft64_100.Frecuencia_Hz_;
m1 = fft64_100.Magnitud_V_;
% FFT 128 puntos
t2 = fft128_100.Frecuencia_Hz_;
m2 = fft128_100.Magnitud_V_;
% FFT 256 puntos
t3 = fft256_100.Frecuencia_Hz_;
m3 = fft256_100.Magnitud_V_;
% FFT 512 puntos
t4 = fft512_100.Frecuencia_Hz_;
m4 = fft512_100.Magnitud_V_;
% FFT 1024 puntos
t5 = fft1024_100.Frecuencia_Hz_;
m5 = fft1024_100.Magnitud_V_;
% FFT 2048 puntos
t6 = fft2048_100.Frecuencia_Hz_;
m6 = fft2048_100.Magnitud_V_;
% Crear figura con 6 subplots en una sola columna
```

Ilustración 7 Código 100 Hz.

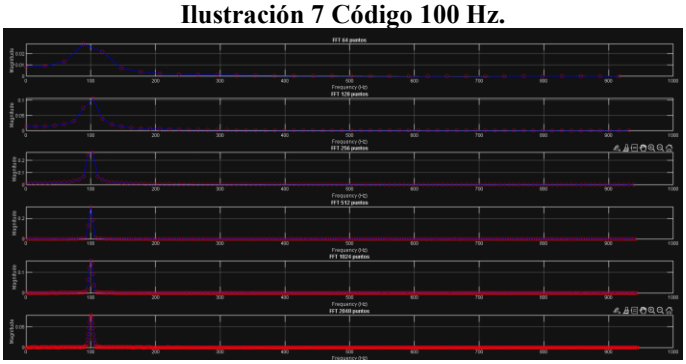

Ilustración 8 FFT 100 Hz.

Ahora vamos a hacer zoom para poder visualizar mejor los punto de la FFT y que diferencias tenemos con la de 100 Hz.

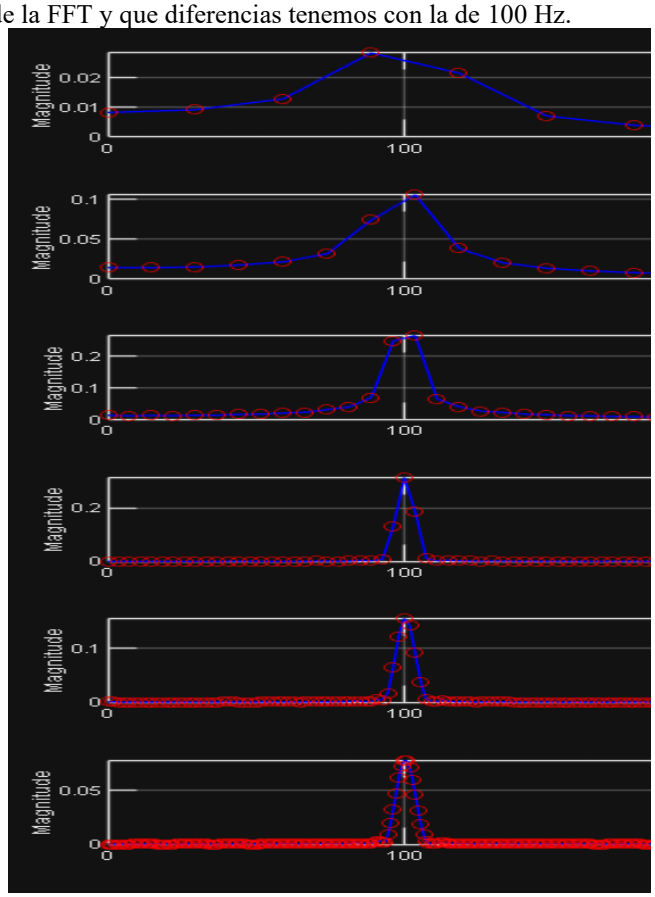

Ilustración 9 Zoom FFT.

Al analizar los resultados de las FFT para 100 Hz y 200 Hz, se observa que el número de puntos afecta de manera distinta la resolución espectral según la frecuencia de la señal. En el caso de 100 Hz, con 64 y 128 puntos la separación entre bins de frecuencia es demasiado amplia (Δf = 31.25 Hz y 15.6 Hz, respectivamente), lo que genera un pico ancho y deformado por el efecto de la ventana Hann, cuyo lóbulo principal se extiende sobre un rango mucho mayor que la frecuencia fundamental. Incluso con 256 puntos ($\Delta f \approx$ 7.8 Hz) todavía se aprecia dispersión y ensanchamiento, de modo que recién a partir de 512 puntos ($\Delta f \approx$ 3.9 Hz) el pico en 100 Hz se define con claridad, y con 1024 o 2048 puntos se obtiene un espectro mucho más preciso y estable gracias a la observación de un mayor número de periodos de la señal. En contraste, para 200 Hz la misma rejilla espectral resulta más adecuada, ya que con 256 puntos ($\Delta f \approx$ 7.8 Hz) el pico ya se representa con mejor definición, y desde 512 puntos en adelante la representación es bastante fiel. Esto confirma que las señales de frecuencia más baja exigen un mayor número de puntos en la FFT para evitar deformaciones y capturar la energía de la frecuencia fundamental, mientras que a frecuencias más altas la misma cantidad de puntos ofrece una resolución suficiente en el dominio de la frecuencia.

Ahora vamos a hacer lo mismo para la muestra

```
t = muestras2048_100.Tiempo_s_
v = muestras2048_100.Voltaje_V_;
T = 0.005;
indices = t <= 5*T;
t_filtrado = t(indices);
v_filtrado = v(indices);
plot(t_filtrado, v_filtrado);
xlabel('Tiempo (s)');
ylabel('Voltaje (V)');
```

Ilustración 10 muestras 100 Hz.

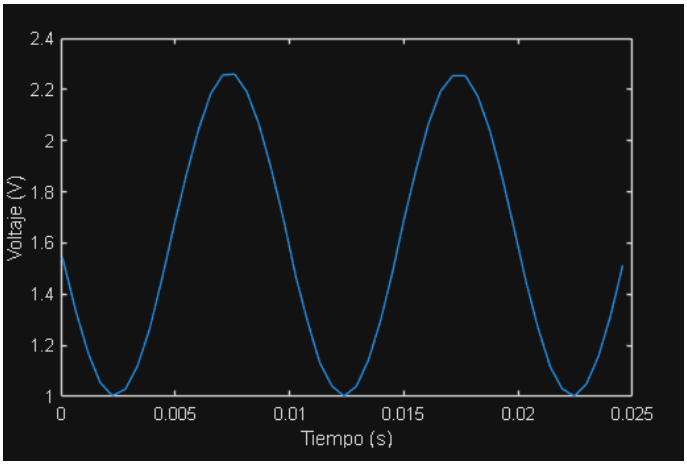

Ilustración 11 100 HZ en el tiempo.

En el dominio del tiempo, la señal muestreada se aprecia con mayor claridad al representarse únicamente durante cinco periodos, lo que permite observar con detalle su forma de onda. La gráfica muestra una senoidal de 200 Hz con componente DC alrededor de 1.6 V, alcanzando valores mínimos cercanos a 1.0 V y máximos de aproximadamente 2.2 V. Al recortar la visualización, se evita la compresión excesiva de la señal y se facilita la verificación de la amplitud y la frecuencia, asegurando

que el proceso de muestreo y digitalización se realizó de manera adecuada antes del análisis en el dominio de la frecuencia.

IV. 900 Hz.

Ahora vamos a cambiar la frecuencia en el generador de señales a 900 Hz y mirar sus cambios en la FFT, vamos a usar un código parecido lo único que cambiamos del código anterior de la FFT es lo siguiente.

```
% FFT 64 puntos
t1 = fft900_64.Frecuencia_Hz_;
m1 = fft900_64.Magnitud_V_;
% FFT 128 puntos
t2 = fft900_128.Frecuencia_Hz_;
m2 = fft900_128.Magnitud_V_;
% FFT 256 puntos
t3 = fft900_256.Frecuencia_Hz_;
m3 = fft900_256.Magnitud_V_;
% FFT 512 puntos
t4 = fft900_512.Frecuencia_Hz_;
m4 = fft900_512.Magnitud_V_;
% FFT 1024 puntos
t5 = fft900_1024.Frecuencia_Hz_;
m5 = fft900_1024.Magnitud_V_;
% FFT 2048 puntos
t6 = fft900_2048.Frecuencia_Hz_;
m6 = fft900_2048.Magnitud_V_;
```

Ilustración 12 900 Hz.

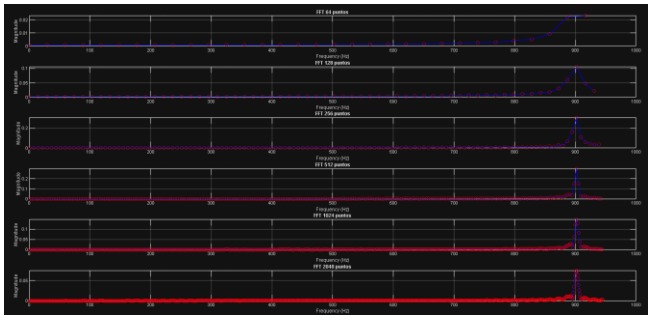

Ilustración 13 FFT 900 Hz.

Ahora vamos a hacer zoom para poder visualizar mejor los punto de la FFT y que diferencias tenemos con la de 900 Hz.

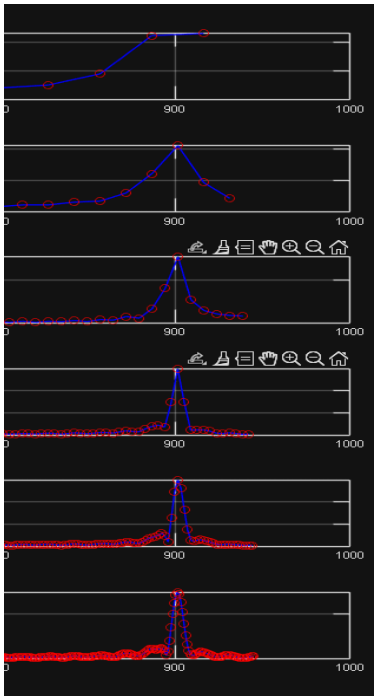

Ilustración 14 Zoom FFT 900 Hz.

En el análisis de la señal de 900 Hz, se evidencia con mayor claridad la limitación que impone el número de puntos de la FFT en relación con la frecuencia de muestreo. Como el sistema está configurado para muestrear a 2000 Hz, el rango útil de análisis llega hasta 1000 Hz (frecuencia de Nyquist), lo que significa que la señal de 900 Hz se encuentra muy próxima a este límite. Con un tamaño de 64 puntos, la resolución espectral es de aproximadamente 31.25 Hz, un valor demasiado grande frente a la frecuencia analizada, por lo que el bin de la FFT no coincide con la frecuencia real y el pico característico de la señal no alcanza a formarse, apareciendo deformado e impreciso. Al aumentar el número de puntos a 128 y 256, la resolución mejora (15.6 Hz y 7.8 Hz, respectivamente), lo que permite que el pico comience a delinearse, aunque todavía con cierto ensanchamiento debido a la fuga espectral y a la proximidad del límite de Nyquist.

Es a partir de 512 puntos (3.9 Hz de resolución) cuando el espectro de la señal de 900 Hz se representa de manera mucho más nítida, logrando un pico centrado y definido, mientras que con 1024 y 2048 puntos el resultado es aún más preciso, mostrando una coincidencia casi exacta entre la frecuencia dominante y el bin correspondiente de la FFT. Esto contrasta con los casos de 100 Hz y 200 Hz, donde la frecuencia se encuentra más alejada de los límites y el efecto de la baja resolución es menos severo, de modo que con 256 puntos ya es posible identificar los picos de forma aceptable. La situación en 900 Hz demuestra que, cuando la señal se ubica en frecuencias cercanas a Nyquist, las exigencias sobre el número de puntos de la FFT aumentan, ya que la resolución debe ser lo suficientemente fina para evitar deformaciones y para representar con exactitud la energía concentrada en la frecuencia fundamental. En síntesis, el caso de 900 Hz pone en evidencia que tanto en frecuencias muy bajas como en las próximas al límite del rango de muestreo se requiere un mayor número de puntos en la FFT, siendo indispensable este ajuste para garantizar una representación espectral confiable.

```
t = muestras900_2048.Tiempo_s_;
v = muestras900_2048.Voltaje_V_;
T = 0.005;
indices = t <= 5*T;
t_filtrado = t(indices);
v_filtrado = v(indices);
plot(t_filtrado, v_filtrado);
xlabel('Tiempo (s)');
ylabel('Voltaje (V)');
```

Ilustración 15 Código muestra 900 Hz.

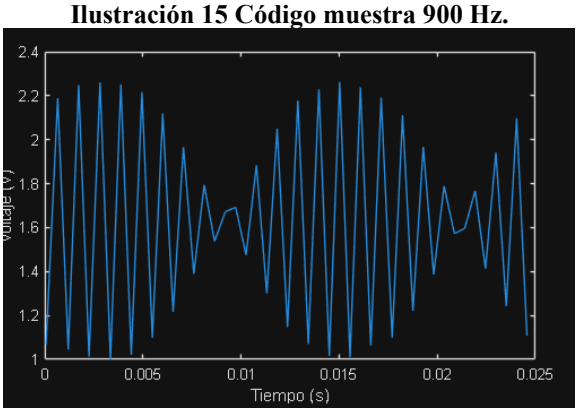

Ilustración 16 señal en el tiempo.

En la gráfica del dominio del tiempo correspondiente a la señal de 900 Hz, se evidencia una distorsión notoria en comparación con las señales de menor frecuencia, como la de 200 Hz. Este comportamiento se explica por la relación entre la frecuencia de la señal y la frecuencia de muestreo utilizada. En este caso, el sistema opera con una frecuencia de muestreo de 2000 Hz, lo que implica que, para una señal de 900 Hz, únicamente se capturan alrededor de 2.2 muestras por ciclo. Esta cantidad de muestras es insuficiente para reconstruir de forma precisa la forma de onda senoidal, provocando que la representación en el tiempo se vea irregular y con aparentes oscilaciones erráticas que no corresponden al comportamiento real de la señal analógica.

A diferencia de lo que ocurre con frecuencias más bajas, como 200 Hz, donde se obtienen aproximadamente 10 muestras por ciclo, permitiendo una visualización limpia y definida de la onda, en el caso de 900 Hz el muestreo cercano al límite de Nyquist genera una pérdida considerable de información. Aunque el teorema de Nyquist establece que con al menos dos muestras por ciclo es posible evitar el aliasing, en la práctica esta condición mínima no garantiza una representación fiel ni estable de la señal. Para que la reconstrucción en el tiempo sea adecuada, se necesitan bastantes más muestras por ciclo, lo cual solo puede lograrse incrementando la frecuencia de muestreo.

La deformación observada en esta señal de 900 Hz ilustra de manera evidente las limitaciones de un sistema de adquisición digital cuando se trabaja con frecuencias cercanas a la mitad de la frecuencia de muestreo. Este fenómeno también explica por qué, en el dominio de la frecuencia, se requieren FFT con un mayor número de puntos para mejorar la resolución espectral y permitir la correcta identificación de la frecuencia fundamental. En conclusión, la distorsión en el tiempo refleja no solo el cumplimiento mínimo de Nyquist, sino también la importancia práctica de contar con una tasa de muestreo significativamente mayor para garantizar representaciones confiables tanto en el dominio temporal como en el espectral.

V. 1800 Hz.

Ahora vamos a cambiar la frecuencia en el generador de señales a 1800 Hz y mirar sus cambios en la FFT, vamos a usar un código parecido lo único que cambiamos del código anterior de la FFT es lo siguiente.

```
% FFT 64 puntos
t1 = fft1800_64.Frecuencia_Hz_;
m1 = fft1800_64.Magnitud_V_;
% FFT 128 puntos
t2 = fft1800_128.Frecuencia_Hz_;
m2 = fft1800_128.Magnitud_V_;
% FFT 256 puntos
t3 = fft1800_256.Frecuencia_Hz_;
m3 = fft1800_256.Magnitud_V_;
% FFT 512 puntos
t4 = fft1800_512.Frecuencia_Hz_;
m4 = fft1800_512.Magnitud_V_;
% FFT 1024 puntos
t5 = fft1800_1024.Frecuencia_Hz_;
m5 = fft1800_1024.Magnitud_V_;
% FFT 2048 puntos
t6 = fft1800_2048.Frecuencia_Hz_;
m6 = fft1800_2048.Magnitud_V_;
% Crear figura con 6 subplots en una sola columna
```

Ilustración 17 código 1800 Hz.

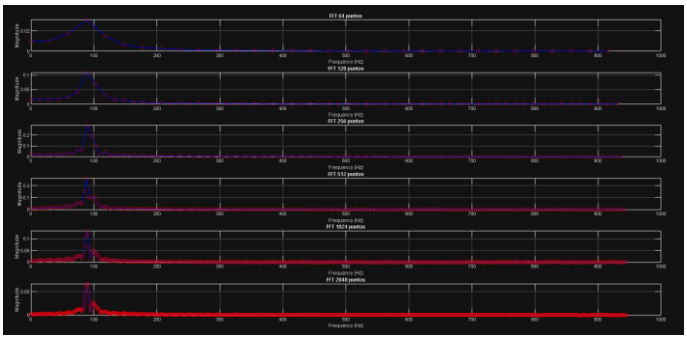

Ilustración 18 FFT 1800 Hz.

Ahora vamos a hacer zoom para poder visualizar mejor los punto de la FFT y que diferencias tenemos con la de 900 Hz.

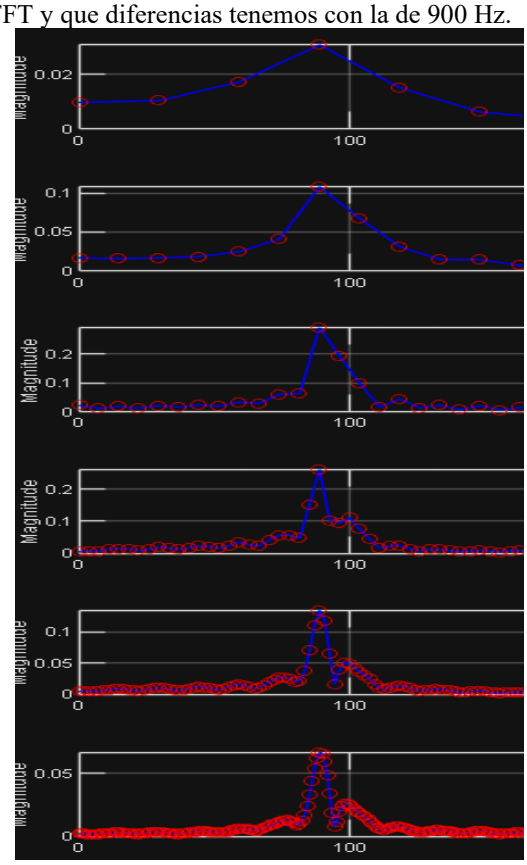

Ilustración 19 zoom FFT 1800 Hz.

La implementación de un analizador espectral en la Raspberry Pi Pico con MicroPython presenta limitaciones importantes que afectan la precisión de las mediciones. La diferencia fundamental radica en la resolución de la FFT ($\Delta f = fs/N$) versus las muestras por ciclo (fs/f_señal). Aumentar N mejora la resolución espectral pero no soluciona la deformación de la señal en el dominio del tiempo cuando la frecuencia de muestreo es insuficiente. Por ejemplo, a 900 Hz con fs≈2000 Hz se obtienen solo 2.22 muestras/ciclo, resultando en una señal escalonada, mientras que a 1800 Hz ocurre aliasing con solo 1.11 muestras/ciclo.

El aliasing teórico para 1800 Hz con fs=2000 Hz debería reflejarse en 200 Hz, pero en la práctica aparece alrededor de 80-90 Hz debido al jitter en la temporización causado por la inexactitud de utime.sleep_us() y la sobrecarga de MicroPython. Esto hace que la frecuencia de muestreo real (fs_real) difiera de la nominal. La conclusión clave es que siempre debe usarse la

fs_real calculada a partir del tiempo observado para generar el eje de frecuencias y corregir desplazamientos.
Las limitaciones prácticas incluyen el jitter temporal significativo debido al funcionamiento interpretado de MicroPython, la resolución efectiva de ~12 bits del ADC a pesar de lecturas de 16 bits, ruido en la referencia de 3.3V, y la reducción de throughput por las operaciones de E/S. La ventana Hann reduce el leakage pero ensancha los lóbulos principales, especialmente problemático con N pequeños donde el lóbulo puede abarcar decenas de Hz.
Para mejorar las mediciones: aumentar fs para mejor representación temporal (ej: ≥9000 Hz para 900 Hz), usar PIO+DMA mediante el SDK de C para muestreo determinista de alta velocidad, emplear siempre fs_real para el eje de frecuencias, normalizar la FFT correctamente, implementar promedio de múltiples FFT (Welch), y añadir filtro anti-aliasing analógico. También es crucial estabilizar Vref con desacoplamiento adecuado, evitar escrituras a flash durante la adquisición, minimizar operaciones en el bucle de lectura, y deshabilitar el garbage collector durante muestreo. Para máxima precisión, debe considerarse un ADC externo con interfaz SPI+DMA.

```
t = muestras1800_2048.Tiempo_s_;
v = muestras1800_2048.Voltaje_V_;
T = 0.005;
indices = t <= 5*T;
t_filtrado = t(indices);
v_filtrado = v(indices);
plot(t_filtrado, v_filtrado);
xlabel('Tiempo (s)');
ylabel('Voltaje (V)');
```

Ilustración 20 Código muestra 1800 Hz.

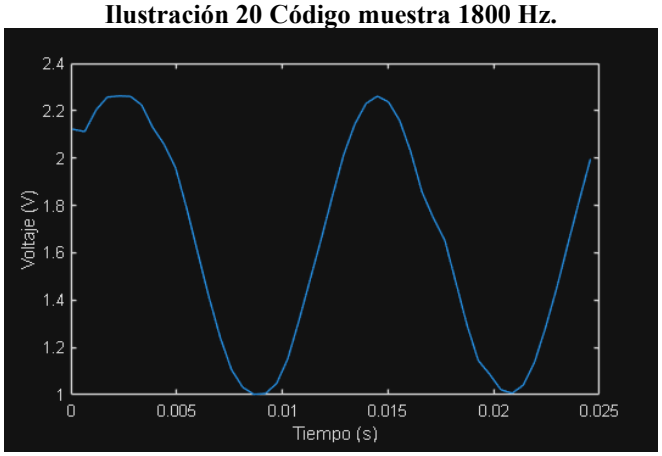

Ilustración 21 señal en el tiempo.

En el dominio del tiempo, la señal de 80 a 90 Hz se observa claramente bien formada, con una onda senoidal continua y simétrica que conserva tanto su amplitud como sus máximos y mínimos sin deformaciones notorias. Esto ocurre porque, con una frecuencia de muestreo de 2000 Hz, se obtienen aproximadamente 22 a 25 muestras por ciclo, lo que significa que cada periodo de la señal está descrito con suficiente cantidad de puntos para reconstruir su forma de manera precisa. Este nivel de muestreo no solo cumple el teorema de Nyquist, sino que lo supera con holgura, permitiendo una representación temporal fiel y libre de artefactos visuales. Además, al contar con tantas muestras por ciclo, la señal se ve suave, con transiciones claras y sin los escalonamientos bruscos que se observan cuando el muestreo es insuficiente.

En contraste, cuando se analizó la señal de 900 Hz bajo las mismas condiciones de muestreo, la situación fue muy diferente. Con solo 2.2 muestras por ciclo, la onda no dispone de suficientes puntos para reconstruir adecuadamente cada periodo, lo que genera una representación distorsionada en el tiempo. En este caso, aunque el sistema todavía cumple en teoría con el teorema de Nyquist, en la práctica la cantidad mínima de muestras por ciclo no es suficiente para que la forma de onda se vea limpia y estable, resultando en una señal deformada y con irregularidades aparentes. Este fenómeno evidencia que, mientras más baja sea la frecuencia de la señal en comparación con la frecuencia de muestreo, mayor será la fidelidad en la representación temporal, ya que se dispondrá de más muestras por ciclo para describir la onda.

Por tanto, el caso de la señal de 80–90 Hz muestra cómo un muestreo denso en relación con la frecuencia de la señal permite una visualización confiable y precisa en el tiempo, mientras que la señal de 900 Hz, al estar próxima al límite de Nyquist, pone en evidencia las limitaciones del muestreo digital cuando se trabaja con frecuencias altas en comparación con la frecuencia de adquisición. Esta diferencia resalta la importancia de garantizar no solo el cumplimiento teórico del criterio de Nyquist, sino también un margen de seguridad suficiente para obtener representaciones claras y útiles en el dominio temporal.
En el programa la tasa de muestreo se establece a partir de la variable f_muestreo, que en el código está definida con un valor de 2000 Hz. A partir de esta frecuencia deseada se calcula el intervalo de muestreo en microsegundos con la instrucción:
dt_us = int(1_000_000 / f_muestreo)
Este valor de dt_us es el tiempo que el programa utiliza en la función utime.sleep_us(dt_us) para esperar entre cada lectura del ADC, garantizando que las muestras se tomen de manera uniforme con la frecuencia definida. En otras palabras, el usuario fija primero la frecuencia de muestreo (f_muestreo), y el programa ajusta el retardo entre muestras para aproximarse a esa tasa. Además, al finalizar la adquisición, se calcula la frecuencia de muestreo real (fs_real) dividiendo el número de muestras tomadas entre el tiempo total transcurrido, con el fin de verificar la precisión respecto a la frecuencia deseada.

VI. CODIGO CON JITTER.

En esta sección del laboratorio se propone el desarrollo de un programa alternativo al suministrado por el docente (ADC_testing.py) con el objetivo de evaluar la adquisición de señales en la Raspberry Pi Pico de manera más controlada. El programa diseñado permite muestrear una señal senoidal de 200 Hz, con una amplitud de 1.2 Vpp y una componente de continua de 1.6 V, procurando garantizar un tiempo de muestreo lo más estable posible. Para este propósito se implementaron y compararon dos códigos diferentes: el primero, basado en el programa del profesor, presenta cierto nivel de jitter en los intervalos de muestreo debido a las limitaciones propias de MicroPython; y el segundo, diseñado por nosotros, busca reducir ese efecto y ofrecer una adquisición más uniforme. Este

ejercicio no solo busca validar la correcta captura de la señal, sino también analizar las posibilidades y limitaciones del dispositivo, como la resolución del ADC, la estabilidad de la temporización y los efectos de jitter introducidos por el software. Finalmente, se realizan pruebas experimentales con ambos programas y se comparan los resultados obtenidos respecto a los valores teóricos de una señal senoidal ideal, con el fin de valorar el desempeño real del sistema y las diferencias prácticas entre ambas implementaciones.

```python
# Cálculo de jitter (adaptado del código original)
def calculate_jitter(tiempos, fs_real):
    Ts_real = 1 / fs_real
    intervals = [tiempos[i+1] - tiempos[i] for i in range(len(tiempos)-1)]
    Ts_avg = sum(intervals)/len(intervals)
    jitter_mean = (Ts_avg - Ts_real)*1e6  # en microsegundos
    jitter_rms = (sum([(x - Ts_real)**2 for x in intervals])/len(intervals))**0.5*1e6
    print(f"Jitter medio: {jitter_mean:.2f} us | Jitter RMS: {jitter_rms:.2f} us")

    # Guardar jitter en archivo
    with open("jitter.txt", "w") as f:
        f.write("jitter_mean_us\tjitter_rms_us\n")
        f.write(f"{jitter_mean:.3f}\t{jitter_rms:.3f}\n")

    return jitter_mean, jitter_rms
```

Ilustración 22 JITTER código del docente.

```
>>> %Run -c $EDITOR_CONTENT

MPY: soft reboot
Iniciando adquisición y análisis...
Frecuencia deseada: 2000 Hz, frecuencia real: 1891.18 Hz
Offset DC removido: 0.727 V
Frecuencia dominante: 199.46 Hz
Amplitud señal: 0.015 V
Piso de ruido: 0.00020 V (-84.24 dB FS)
SNR: 37.62 dB, ENOB: 5.96 bits
Jitter medio: 0.75 us | Jitter RMS: 10.54 us
```

Ilustración 23 Datos obtenidos.

Este resultado muestra la ejecución del programa de adquisición y análisis de la señal senoidal en la Raspberry Pi Pico. Aunque la frecuencia de muestreo deseada se configuró en 2000 Hz, la frecuencia real medida fue de aproximadamente 1891.18 Hz, lo que evidencia una desviación causada por las limitaciones de temporización en MicroPython. Tras remover el componente DC, se observó un desplazamiento de 0.727 V, y la frecuencia dominante identificada fue de 199.46 Hz, muy cercana al valor teórico de 200 Hz, lo que confirma la correcta detección de la señal principal. La amplitud medida de la señal fue de 0.015 V, mientras que el piso de ruido se mantuvo bajo en 0.00020 V (-84.24 dBFS). El análisis de desempeño arrojó una relación señal-ruido (SNR) de 37.62 dB y una resolución efectiva (ENOB) de 5.96 bits, lo cual es consistente con las limitaciones prácticas del ADC del microcontrolador. Finalmente, se calcularon métricas de estabilidad temporal, encontrando un jitter medio de 0.75 µs y un jitter RMS de 10.54 µs, confirmando la presencia de variaciones en los intervalos de muestreo que afectan la fidelidad de la adquisición. Estos resultados reflejan tanto la capacidad del sistema para capturar adecuadamente la frecuencia de interés como las restricciones impuestas por la implementación en software y el hardware disponible.

Antes de crear nuestro código debemos responder las siguientes preguntas .

¿Qué es el Jitter? ¿Qué implicaciones tiene en el proceso de codificación de la fuente ?.

El jitter es la variación no deseada en los intervalos de tiempo entre muestras que deberían ser tomadas de manera uniforme durante el proceso de adquisición digital. Su presencia implica que la señal no se capture en los instantes exactos, lo que genera distorsiones en el dominio del tiempo, ensanchamiento o desplazamiento de picos en la FFT, y una disminución en la fidelidad de la señal digitalizada. En el contexto de la codificación de la fuente, el jitter afecta directamente la calidad del muestreo, ya que reduce la relación señal-ruido (SNR) y el número efectivo de bits (ENOB) del conversor, limitando la precisión con la que se representa la señal analógica. En consecuencia, un muestreo con alto jitter compromete la exactitud del análisis espectral y la confiabilidad de los datos adquiridos, siendo un factor crítico en aplicaciones de telecomunicaciones, instrumentación y procesamiento digital de señales.

```python
# ==========================
# Calcular jitter
# ==========================
jitter_mean = sum(intervals)/len(intervals) - 1e6/fs
jitter_rms = (sum([(x - 1e6/fs)**2 for x in intervals])/len(intervals))**0.5
with open("jitter.csv", "w") as f_j:
    f_j.write("jitter_mean_us,jitter_rms_us\n")
    f_j.write(f"{jitter_mean:.3f},{jitter_rms:.3f}\n")

print(f"Jitter medio = {jitter_mean:.2f} us, RMS = {jitter_rms:.2f} us")
```

Ilustración 24 Código propio.

```
MPY: soft reboot
Frecuencia 50 Hz -> Magnitud 0.01269 V
Frecuencia 200 Hz -> Magnitud 0.01684 V
Frecuencia 400 Hz -> Magnitud 0.00511 V
Jitter medio = 419.62 us, RMS = 3505.38 us
```

Ilustración 25 Jitter propio.

Este resultado corresponde a las pruebas realizadas con el código desarrollado de manera propia, distinto al entregado por el docente. En este caso, se observa la detección de varios armónicos de la señal: un componente en 50 Hz con magnitud de 0.01269 V, la frecuencia principal en 200 Hz con magnitud de 0.01684 V, y un armónico adicional en 400 Hz con magnitud de 0.00511 V. Esto evidencia que el programa logra identificar correctamente la frecuencia fundamental de la señal junto con componentes adicionales, aunque con amplitudes reducidas. Sin embargo, el aspecto más relevante de este resultado está en la medición del jitter, donde se obtuvo un valor medio de 419.62 µs y un RMS de 3505.38 µs. Estas cifras son considerablemente más altas que las observadas en el código del docente, lo que refleja una menor estabilidad en el tiempo de muestreo y una variación significativa entre intervalos de adquisición. En consecuencia, aunque el código propio permite identificar la frecuencia fundamental y los armónicos de la señal, el nivel de jitter introduce una inestabilidad importante que afecta la precisión de la adquisición y limita la calidad del análisis espectral.

VII. Análisis.

En primer lugar, al analizar el programa sugerido por el profesor (ADC_testing.py), se observa que la tasa de muestreo se establece a partir de la variable f_muestreo, definida con un valor nominal de 2000 Hz. Con este dato, el programa calcula el intervalo de muestreo en microsegundos mediante la instrucción dt_us = int(1_000_000 / f_muestreo), que luego se utiliza en la función utime.sleep_us(dt_us) para espaciar las lecturas del ADC. De esta manera se logra un muestreo aproximadamente uniforme, aunque en la práctica la frecuencia real (fs_real) difiere de la deseada por las limitaciones de temporización en MicroPython.

Dentro del código se emplean funciones que cumplen un papel fundamental en el proceso de adquisición y análisis. La función acquire_data() se encarga de tomar las muestras de la señal

analógica, almacenarlas junto con sus marcas de tiempo y calcular la frecuencia de muestreo real. Posteriormente, convert_to_voltage() transforma las lecturas crudas del ADC (valores digitales de 16 bits) en voltajes reales en función de la referencia de 3.3 V. La función remove_offset() elimina la componente de continua (DC) de la señal, lo cual es esencial para centrarla en torno a cero y facilitar el análisis espectral. Seguidamente, se aplica apply_hanning_window(), que multiplica la señal por una ventana de tipo Hanning con el fin de reducir la fuga espectral (spectral leakage) en la FFT. Finalmente, fft_manual() implementa de manera explícita el algoritmo de la Transformada Rápida de Fourier por el método Radix-2, y analyze_fft() procesa sus resultados, calculando magnitudes, identificando la frecuencia dominante y estimando parámetros de desempeño como SNR, piso de ruido y número efectivo de bits (ENOB).

La ventana Hanning, en particular, es un elemento clave en el análisis. Su función es suavizar las discontinuidades que se producen en los extremos del intervalo de muestreo, lo que reduce los lóbulos laterales en el espectro y evita que la energía de una frecuencia dominante se disperse sobre otras frecuencias cercanas. En el programa, esta ventana se aplica antes de calcular la FFT, garantizando que la representación espectral sea más precisa y que los picos de frecuencia aparezcan definidos.

Con el uso del programa del docente se observó que, aunque se logra capturar la señal y detectar correctamente la frecuencia dominante, existe un jitter relativamente bajo (≈0.75 µs de media) y una diferencia entre la frecuencia de muestreo deseada y la real (2000 Hz vs. 1891 Hz). Posteriormente, al diseñar un código propio, se buscó mejorar la estabilidad, pero en los resultados se evidenció un jitter mucho mayor (≈419 µs de media, 3505 µs RMS), lo que afecta significativamente la calidad del muestreo, pese a que las frecuencias fundamentales fueron identificadas. Al comparar ambos enfoques, se confirma que el programa del profesor ofrece un muestreo más estable, mientras que el programa propio, aunque permite mayor control y flexibilidad, requiere optimización para minimizar la variación en los tiempos de adquisición.

Finalmente, al analizar las señales en diferentes frecuencias (100 Hz, 200 Hz, 900 Hz y 1800 Hz) se constató el efecto del número de muestras por ciclo en la fidelidad de la representación temporal: las señales de baja frecuencia (80–200 Hz) se reconstruyen adecuadamente en el tiempo al contar con suficientes muestras por periodo, mientras que a frecuencias cercanas a Nyquist (900 Hz) la forma de onda se deforma y a frecuencias superiores (1800 Hz) se produce aliasing, reflejándose la señal en una frecuencia más baja (≈80–90 Hz). Estos resultados permiten concluir que la práctica no solo evidenció el funcionamiento del ADC y del análisis espectral con FFT, sino también las limitaciones prácticas del muestreo digital en la Raspberry Pi Pico, como la dependencia del temporizador en MicroPython, la influencia del jitter y la necesidad de un mayor margen de seguridad en la frecuencia de muestreo para garantizar adquisiciones confiables.

VIII. Conclusiones.

El desarrollo de esta práctica permitió analizar de manera detallada el proceso de adquisición y procesamiento digital de señales utilizando la Raspberry Pi Pico y comparar diferentes métodos de implementación en MicroPython. En primer lugar, el estudio del programa ADC_testing.py mostró cómo se establece la tasa de muestreo a partir de una frecuencia deseada y cómo se traduce en un retardo en microsegundos para el ADC. Sin embargo, se evidenció que la frecuencia real alcanzada difiere de la nominal, debido a las limitaciones de precisión en la temporización de MicroPython, lo cual introduce un error en el muestreo. A pesar de ello, el programa fue capaz de capturar correctamente la señal y de identificar su frecuencia dominante, mostrando un desempeño estable y un nivel de jitter relativamente bajo.

Por otra parte, el diseño e implementación de un programa propio permitió experimentar con un enfoque alternativo para la adquisición de datos. Este código logró identificar no solo la frecuencia fundamental, sino también algunos armónicos de la señal, mostrando mayor flexibilidad en el análisis. No obstante, las pruebas revelaron un jitter considerablemente más alto en comparación con el programa del docente, lo que afectó la estabilidad de los intervalos de muestreo y redujo la precisión del análisis espectral. Esto evidenció la necesidad de optimizar la gestión del tiempo en la captura de datos para mejorar la calidad de los resultados.

Adicionalmente, el uso de la ventana Hanning se confirmó como un elemento clave dentro del procesamiento, ya que permitió reducir la fuga espectral y obtener picos más definidos en el dominio de la frecuencia. Este recurso resultó esencial para interpretar con claridad los resultados de la FFT y corroborar la frecuencia fundamental de la señal, lo que valida su importancia en el análisis digital.

En cuanto al estudio de las señales en diferentes frecuencias, se comprobó experimentalmente el impacto que tiene el número de muestras por ciclo sobre la representación temporal de la onda. Las señales de baja frecuencia, como las de 80 a 200 Hz, pudieron representarse de forma clara en el tiempo, ya que disponen de un número suficiente de muestras por periodo. Sin embargo, en señales cercanas a Nyquist, como los 900 Hz, la onda se deformó notablemente al contar con apenas 2 muestras por ciclo, y en el caso de 1800 Hz, que supera Nyquist, se produjo aliasing, observándose la señal reflejada en frecuencias más bajas. Estos resultados confirman la teoría del muestreo digital y evidencian la importancia de trabajar con márgenes de seguridad amplios entre la frecuencia de muestreo y la frecuencia máxima de la señal.

En conclusión, la práctica no solo permitió validar los fundamentos teóricos del muestreo y la transformada rápida de Fourier, sino que también permitió reconocer de manera práctica las posibilidades y limitaciones del hardware y software de la Raspberry Pi Pico. Se evidenció que la plataforma es adecuada para adquirir y analizar señales de baja frecuencia, pero presenta restricciones importantes cuando se requiere alta precisión en la temporización. Finalmente, se destaca la importancia de explorar recursos más avanzados como el uso de PIO o DMA en el RP2040 para lograr tiempos de muestreo más estables y confiables en aplicaciones de mayor exigencia.

Referencias

[1] Raspberry Pi Ltd., “Raspberry Pi Pico W Datasheet”, 2022. [Online]. Available: https://datasheets.raspberrypi.com/picow/pico-w-datasheet.pdf

[2] Raspberry Pi Ltd., “Getting Started with Raspberry Pi Pico”, 2021. [Online]. Available: https://datasheets.raspberrypi.com/pico/getting-started-with-pico.pdf

[3] B. Razavi, Design of Analog CMOS Integrated Circuits, McGraw-Hill, 2001. (Referencia clásica sobre jitter, temporización y errores de muestreo).

[4] A. V. Oppenheim and R. W. Schafer, Discrete-Time Signal Processing, 3rd ed., Pearson, 2010. (Fundamento de muestreo, criterio de Nyquist y distorsiones).
[5] Texas Instruments, “Understanding Data Converters”, 1995. (Explica errores de cuantización, jitter y limitaciones del ADC).
[6] A. Maxim, “ADC Sampling and Aliasing”, Maxim Integrated, Application Note 634. [Online]. Available: https://www.analog.com/media/en/technical-documentation/application-notes/636.pdf
[7] ARM Ltd., “ARM Cortex-M0+ Technical Reference Manual”, 2011. [Online]. Available: https://developer.arm.com/documentation/ddi0484/c