



LABORATORIO DE JITTER CON RASPBERRY PI PICO 2W

Vallentina Diaz Valbuena

est.vallentina.diaz@unimilitar.edu.co

Docente: José de Jesús Rugeles

I. RESUMEN

Este informe presenta el desarrollo experimental de un laboratorio orientado al uso del conversor A/D integrado en la Raspberry Pi Pico 2W, con el objetivo de analizar el proceso de muestreo y los efectos del *jitter* en la digitalización de señales analógicas. Inicialmente, se configuró un generador de señales para producir una onda senoidal de 200 Hz con una componente DC, verificando la captura mediante el ADC y la posterior aplicación de la Transformada Rápida de Fourier (FFT). Se implementaron funciones en Python para adquisición de datos, conversión a voltaje, eliminación de offset, aplicación de ventana de Hanning y análisis espectral. Se evaluaron distintas frecuencias de entrada y tamaños de FFT, verificando la validez del teorema de Nyquist y la influencia del número de puntos en la resolución espectral. Posteriormente, se introdujo intencionalmente *jitter* en el tiempo de muestreo, demostrando sus efectos en la precisión de la señal y en la degradación del SNR. Finalmente, se analizaron alternativas de mejora con el Raspberry Pi Pico 2W, incluyendo el uso de temporizadores de hardware, PIO y DMA para reducir el *jitter* y optimizar el muestreo. Los resultados experimentales confirmaron que, a pesar de las limitaciones del ADC interno, es posible realizar una digitalización estable y confiable con estrategias adecuadas de temporización y procesamiento.

Palabras clave— Conversor A/D, Raspberry Pi Pico 2W, muestreo, FFT, jitter, teorema de Nyquist, procesamiento digital de señales.

II. ABSTRACT

Abstract— This report presents the experimental development of a laboratory focused on the use of the A/D converter integrated in the Raspberry Pi Pico 2W, with the objective of analyzing the sampling process and the effects of *jitter* on the digitization of analog signals. Initially, a signal generator was configured to produce a 200 Hz sine wave with a DC component, verifying data acquisition through the ADC and the subsequent application of the Fast Fourier Transform (FFT). Python functions were implemented for data acquisition, voltage conversion, offset removal, Hanning

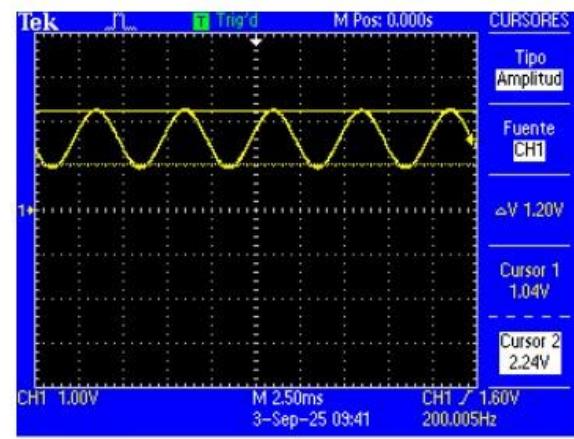
window application, and spectral analysis. Different input frequencies and FFT sizes were evaluated, verifying the Nyquist theorem and the influence of the number of points on spectral resolution. Later, *jitter* was intentionally introduced into the sampling period, demonstrating its effects on signal accuracy and SNR degradation. Finally, improvement alternatives for the Raspberry Pi Pico 2W were analyzed, including the use of hardware timers, PIO, and DMA to reduce *jitter* and optimize sampling. The experimental results confirmed that, despite the limitations of the internal ADC, it is possible to achieve stable and reliable digitization with appropriate timing and processing strategies.

Keywords— A/D converter, Raspberry Pi Pico 2W, sampling, FFT, jitter, Nyquist theorem, digital signal processing.

III. DESARROLLO EXPERIMENTAL

Parte 1

Se configura el generador de señales Rigol DG1022 del laboratorio para que entregue una señal senoidal de 200 Hz, 1.2 Vpp y una componente DC de 1.6 V, para verificar la señal se conecta al osciloscopio y se observa la siguiente imagen:



TDS 2012C - 8:35:38 p. m. 2/09/2025

Imagen 1. Señal seno en osciloscopio

Datasheet de la Raspberry Pi Pico 2W

El datasheet destaca sus 26 GPIO multifunción (con ADC de 12 bits, UART, SPI e I2C), capacidad de operar entre 1.8V y 5.5V, y modos de bajo consumo energético. Además, su diseño mantiene compatibilidad con las librerías estándar de MicroPython. Esta combinación de potencia, conectividad y eficiencia la posiciona como una solución versátil para prototipado rápido y sistemas integrados avanzados. A continuación se muestra el datasheet descrito:

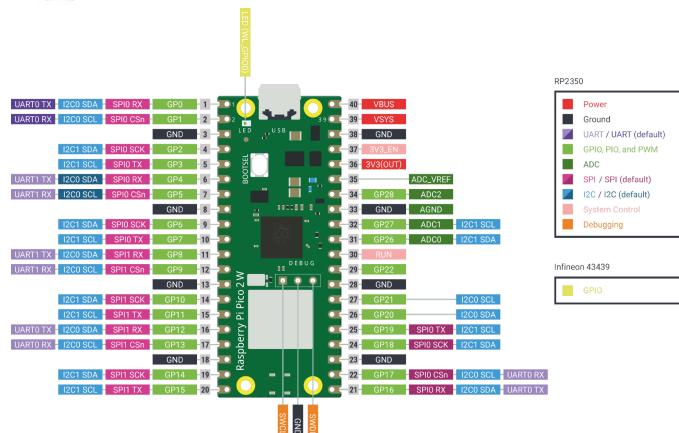


Imagen 2. Datasheet Raspberry Pi Pico 2W

Ahora bien, con toda la información técnica, se procede a identificar los terminales ADC en las GPIO del dispositivo, ya determinados se conecta al generador de señales Rigol DG1022 del laboratorio, entre la terminal ADC1 y tierra del dispositivo. En la siguiente imagen se observa que el terminal GP27 es el pin 32 y el terminal GND es el pin 33.

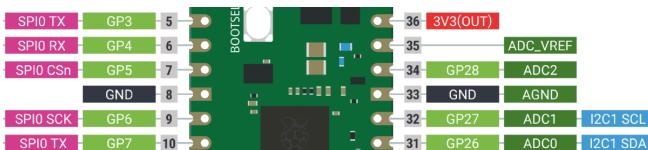


Imagen 3. ADC y GND

A continuación, se muestra cómo sería esa conexión física:



Imagen 4. ADC y GND al generador de señales

Se ejecuta el programa [ADC_testing.py](#) disponible https://github.com/jrugeles/source_coding en el entorno de desarrollo integrado Thonny con Python, para analizar señales con FFT (Transformada Rápida de Fourier). Las funciones incluidas en el código fueron:

Función	Descripción	Parámetros
acquire_data()	Adquiere N muestras del ADC con timing preciso	Ninguno
convert_to_voltage(data, VREF=3.3)	Convierte valores ADC a voltajes	data: array de muestras, VREF: voltaje de referencia
remove_offset(data)	Elimina el componente DC (offset) de la señal	data: array de voltajes
apply_hannning_window(data)	Aplica ventana de Hanning para reducir la fuga espectral	data: array de señales
fft_manual(x, N_FFT)	Implementa FFT	x: señal de entrada, N_FFT: tamaño de FFT
analyze_fft(result, fs_real, N_FFT)	Analiza resultados FFT y calcula métricas	fft_result: resultado FFT, fs_real: frecuencia de muestreo, N_FFT: tamaño FFT
main()	Función principal que estructura todo el proceso	Ninguno

Tabla 1. Funciones del código

Ahora bien, ¿Para qué sirve la ventana Hanning y como se utiliza en el programa?

Su propósito principal es reducir un artefacto matemático llamado "leakage" (fugas o derrame espectral).

- El problema (Leakage): La FFT asume que la señal que se le da se repite infinitamente. Si se capturan un número entero de ciclos de tu señal, el principio y el final de tu bloque de datos encajan perfectamente. Pero si no es un número entero de ciclos, hay una discontinuidad (un "salto") entre el final del bloque de datos y el inicio del siguiente bloque que la FFT imagina. Esta discontinuidad introduce frecuencias falsas y ensancha los picos reales en el espectro.
- La Solución (Ventana): Una función de ventana, como la de Hanning, suaviza los extremos de la señal hacia cero. Esto elimina la discontinuidad al principio y al final del bloque de datos, haciendo que la señal "parezca" continua cuando se repite. El *trade-off* es que se pierde algo de información en los extremos, pero se gana una enorme precisión espectral.



Es como suavizar los bordes de dos piezas de madera para que encajen mejor, aunque les tengas que quitar un poco de material. En resumen, la ventana de Hanning se aplica a la señal en el dominio del tiempo justo antes de calcular la FFT para suprimir las fugas espectrales y obtener un espectro mucho más limpio y preciso, lo que permite una mejor identificación de la frecuencia dominante y un cálculo más exacto del SNR y el ruido.

La información es tabulada y entregada por el programa, generando los siguientes archivos:

muestras.txt

1	Tiempo(s)	Voltaje(V)
2	0.000066	2.26204
3	0.000654	2.12749
4	0.001186	1.75194
5	0.001712	1.34175
6	0.002237	1.04844
7	0.002797	1.01052
8	0.003322	1.24185
9	0.003848	1.63104
10	0.004375	2.01948

Imagen 5. Tabulación del archivo de muestras

fft.txt

1	Frecuencia(Hz)	Magnitud(V)
2	0.00	0.00041
3	1.86	0.00033
4	3.72	0.00018
5	5.58	0.00027
6	7.44	0.00044
7	9.30	0.00053
8	11.15	0.00053
9	13.01	0.00050
10	14.87	0.00049

Imagen 6. Tabulación del archivo de fft

Estos archivos se guardan en el pc para posteriormente poder graficarlos en MatLab, primero se exportan los datos del archivo *muestras.txt* y se implementa el siguiente código:

```
t=A.Tiempo_s_;
v=A.Voltaje_V_;
plot(t, v, Color='b')
xlabel("Tiempo (s)");
ylabel("Voltaje (V)");
title("Señal senoidal f=200 Hz N=1024");
```

```
grid on;
xlim([0 0.025])
ylim([0.8 2.4])
```

A continuación, se logra visualizar la gráfica generada:

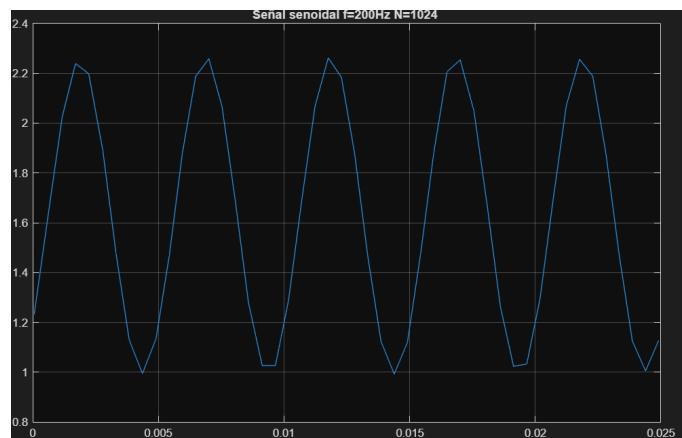


Imagen 7. Señal seno en MatLab

A partir de la imagen anterior se puede comprobar el buen funcionamiento del ADC de la Raspberry, ya que aproximadamente se observan 5 períodos con una frecuencia de 200Hz.

Luego, se exportan los datos del archivo *fft.txt* y se implementa el siguiente código:

```
f=B.Frecuencia_Hz_;
m=B.Magnitud_V_;
plot(f, m, Color='b')
xlabel("Frecuencia (Hz)");
ylabel("Magnitud (V)");
title("FFT f=200 Hz N=1024");
grid on;
```

A continuación, se logra visualizar la gráfica generada:

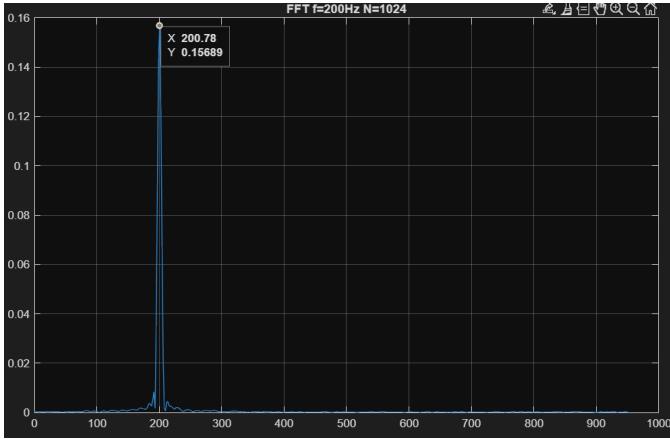


Imagen 8. FFT en MatLab

A partir de la imagen anterior se puede comprobar el buen funcionamiento del ADC de la Raspberry, ya que aproximadamente se observa que el pico de la FFT está ubicado aproximadamente a una frecuencia de 200Hz y se confirma que la frecuencia de muestreo es de 2 kHz.

Ahora bien, se repite el proceso realizando las siguientes modificaciones:

- Se modifica el número de puntos para la FFT en el programa por: 64,128,256,512,1024,2048
- Se modifica la frecuencia de la señal de entrada de la siguiente forma: F=100,300,600,900,1500,1800 Hz.

A continuación se observan las **señales en el tiempo**:

Al código de MatLab se le agrega la siguiente línea de código para poder visualizar las muestras puntuales de la gráfica:

```
hold on
stem(t, v, 'Color', [.3 .8 .3],
      'Marker', 'none')
hold off
```

- 100 Hz

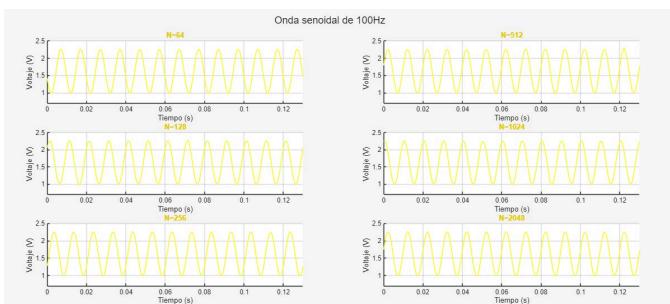


Imagen 9. Señal seno de 100Hz

Se ajusta el programa para observar los primeros 4 períodos de la señal en el dominio del tiempo:

$$\frac{1}{100\text{Hz}} = 0.01\text{s} \times 4 = 0.04\text{s}$$

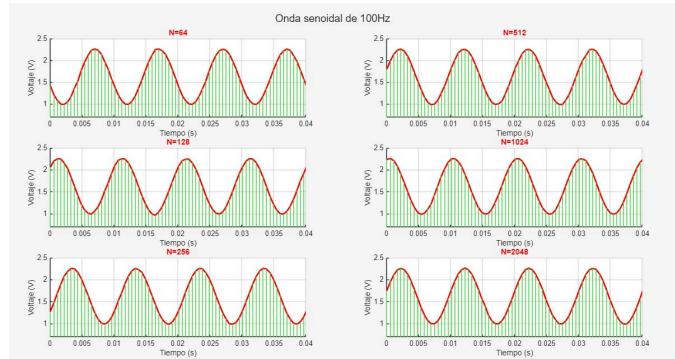


Imagen 10. Señal seno de 100Hz (4 períodos)

De las imágenes anteriores se puede concluir que la forma de onda es limpia y muestra la característica forma suave de una senoidal, confirmando una buena digitalización por parte del ADC de la Raspberry.

- 300 Hz

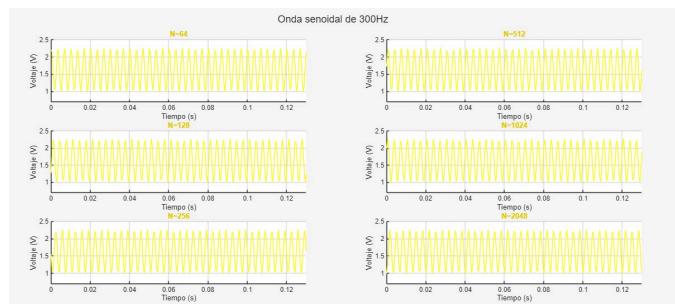


Imagen 11. Señal seno de 300Hz

Se ajusta el programa para observar los primeros 4 períodos de la señal en el dominio del tiempo:

$$\frac{1}{300\text{Hz}} = 3.3\text{ms} \times 4 = 0.014\text{s}$$

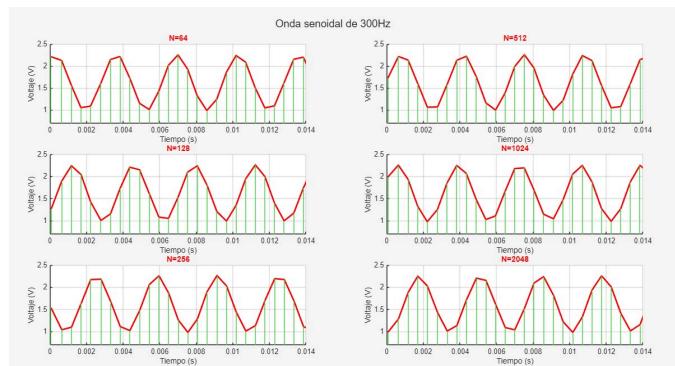


Imagen 12. Señal seno de 300Hz (4 períodos)

De las imágenes anteriores se puede concluir que la amplitud de la señal se mantiene constante a lo largo de los ciclos, lo



que indica una buena respuesta en frecuencia del sistema de adquisición para esta banda.

- 600 Hz

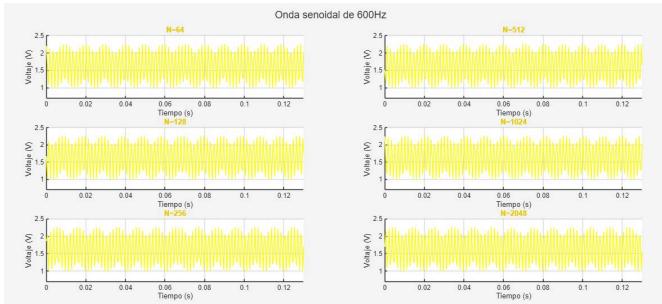


Imagen 13. Señal seno de 600Hz

Se ajusta el programa para observar los primeros 4 períodos de la señal en el dominio del tiempo:

$$\frac{1}{600\text{Hz}} = 1.6\text{ms} \times 4 = 7\text{ms}$$

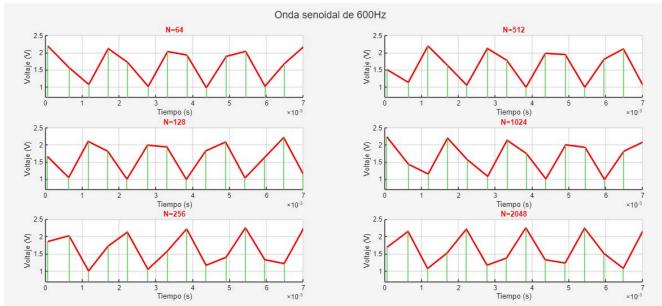


Imagen 14. Señal seno de 600Hz (4 períodos)

De las imágenes anteriores se puede concluir que la forma de la onda sigue siendo claramente senoidal, demostrando que el ADC puede seguir cambios de voltaje más rápidos sin distorsión aparente.

- 900 Hz

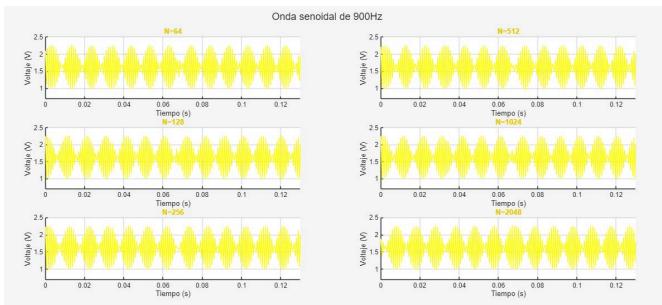


Imagen 15. Señal seno de 900Hz

Se ajusta el programa para observar los primeros 4 períodos de la señal en el dominio del tiempo:

$$\frac{1}{900\text{Hz}} = 1.1\text{ms} \times 4 = 5\text{ms}$$

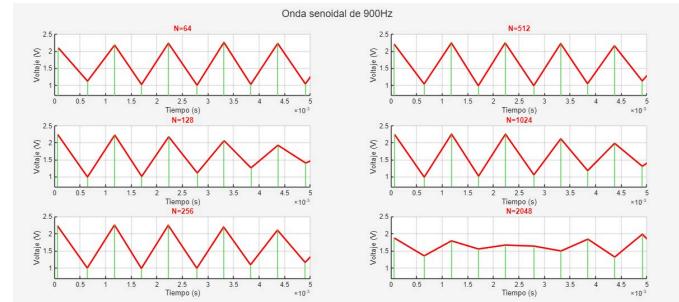


Imagen 16. Señal seno de 900Hz (4 períodos)

De las anteriores imágenes se puede concluir que la señal continua mostrando una forma de onda coherente, aunque a esta frecuencia más alta es posible observar ligeramente el efecto de la resolución finita del ADC en la suavidad de la curva.

- 1800 Hz

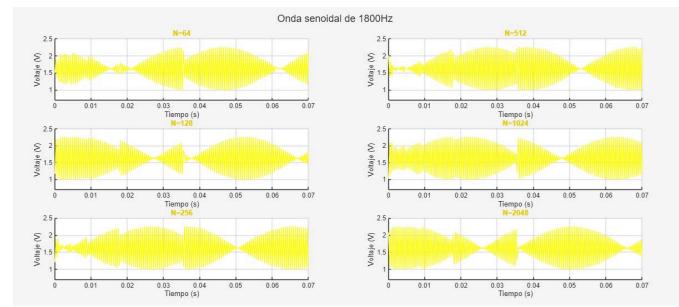


Imagen 17. Señal seno de 1800Hz

Se ajusta el programa para observar los primeros 4 períodos de la señal en el dominio del tiempo:

$$\frac{1}{1800\text{Hz}} = 0.5\text{ms} \times 4 = 3\text{ms}$$

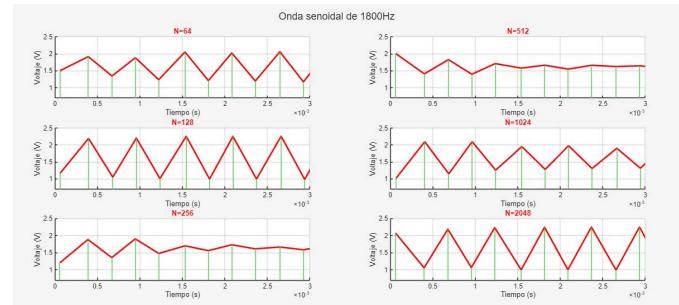


Imagen 18. Señal seno de 1800Hz (4 períodos)

De las anteriores imágenes se puede concluir que la forma de onda muestra una distorsión visible y ya no parece una senoidal perfecta. Esto es un síntoma claro de que la frecuencia de muestreo es insuficiente para reconstruir correctamente la señal de alta frecuencia. Este resultado es fundamental para demostrar experimentalmente la importancia del teorema de muestreo de Nyquist.

A continuación se observan las FFT:



Al código de MatLab se le agrega la siguiente línea de código para poder visualizar las muestras puntuales de la gráfica:

```
hold on

stem(f, m, 'Color',[.3 .8 .3],
'Marker', 'o')

hold off
```

- 100 Hz

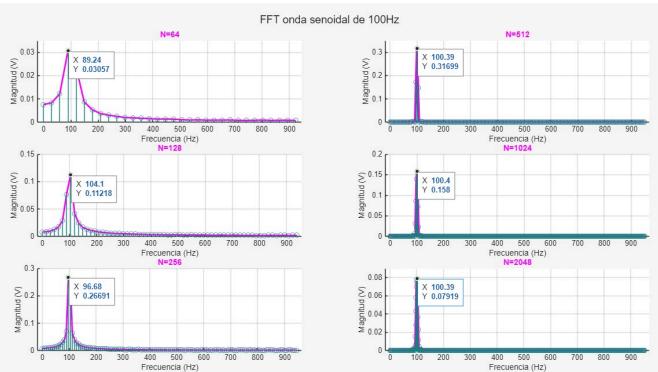


Imagen 19. FFT señal seno de 100Hz

De la anterior imagen se puede concluir que a mayor N, mejor es la resolución, es decir a medida que N aumenta, los picos espectrales se vuelven más delgados y definidos, en N=64 se obtiene un pico ancho y una resolución pobre, pero en N=2048 se obtiene un pico muy estrecho y una excelente resolución.

- 300 Hz

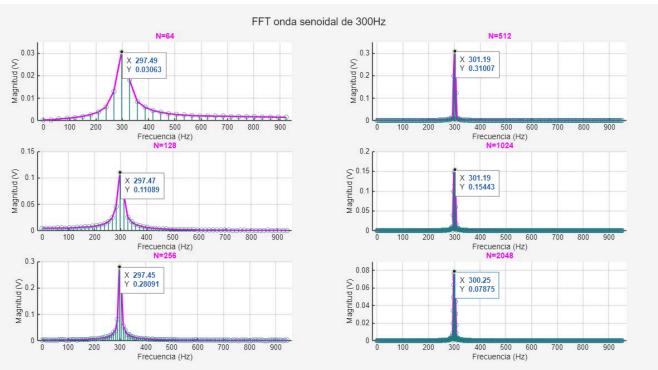


Imagen 20. FFT señal seno de 300Hz

De la imagen anterior se puede concluir que la fuga espectral es pronunciada en N bajos (N=64 y N=128) ya que hay energía dispersa alrededor de 300 Hz, por otro lado la fuga espectral es minimizada en N altos (N=1024 y N=2048) ya que la energía se concentra casi completamente en el pico principal.

- 600 Hz

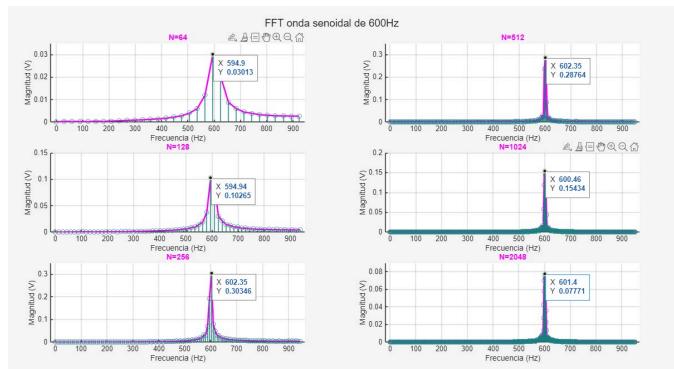


Imagen 21. FFT señal seno de 600Hz

De la imagen anterior se puede concluir que el ruido disminuye con N mayor, es decir el piso de ruido es más bajo en N=1024 y N=2048.

- 900 Hz

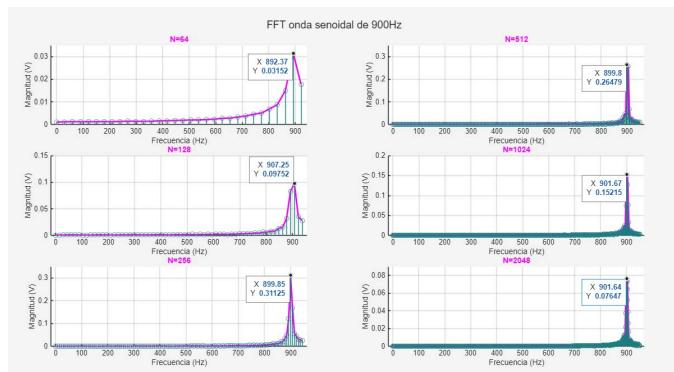


Imagen 22. FFT señal seno de 900Hz

De la imagen anterior se puede concluir que se identifica fácilmente la frecuencia fundamental, los 900 Hz son claramente visibles, en todas las gráficas el pico principal está aproximadamente en 900 Hz, pero con N=2048 la localización de la frecuencia es más precisa.

- 1500 Hz

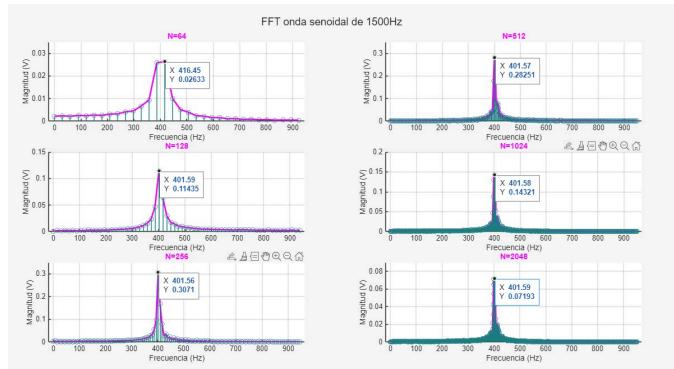


Imagen 23. FFT señal seno de 1500Hz



En la imagen anterior se puede observar que como se mencionó en un principio la frecuencia de muestreo es de 2 kHz, y la frecuencia de la señal de entrada es de 1500 Hz, por lo cual está fuera del límite y no cumple con el teorema de Nyquist, el cual dice que la frecuencia de muestreo debe ser al menos el doble de la frecuencia más alta presente en la señal. Por lo cual el programa se quedó con una frecuencia de aproximadamente 400Hz.

- 1800 Hz

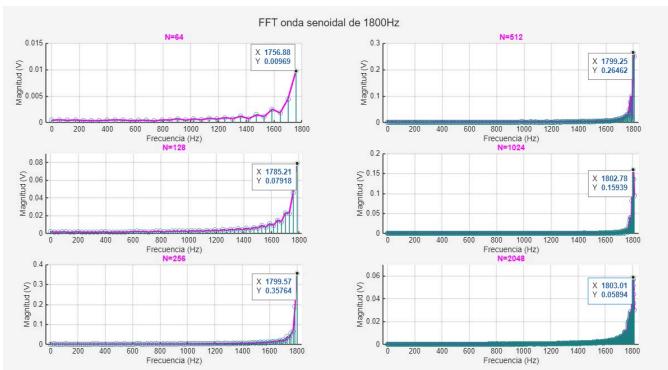


Imagen 24. FFT señal seno de 1800Hz

En la imagen anterior se observa que la frecuencia de muestreo del programa fue modificada a 4 kHz, al reconocer el problema que ocasionó con el ejemplo anterior. Por lo tanto, se identifica fácilmente la frecuencia fundamental, los 1800 Hz son claramente visibles, en todas las gráficas el pico principal está aproximadamente en 1800 Hz, pero con N=2048 la localización de la frecuencia es más precisa.

Ahora bien, ¿Cómo se establece la tasa de muestreo en el programa?

En el programa la tasa de muestreo se define a partir de la variable `f_muestreo`, donde se indica la frecuencia a la que se desea capturar las muestras. Con ese valor se calcula un intervalo de tiempo en microsegundos que se utiliza en la instrucción `utime.sleep_us`, la cual introduce una pausa entre cada lectura del ADC. De esta forma se consigue que las muestras se tomen de manera uniforme, acercándose a la frecuencia de muestreo deseada. Finalmente, el programa calcula la frecuencia real alcanzada y la muestra en pantalla para comprobar la precisión del proceso.

Parte 2

Se diseñó un programa diferente al sugerido que permitiera muestrear una señal senoidal de 200 Hz, 1.2 Vpp y una componente DC de 1.6 V con un tiempo de muestreo estable. Este código se encuentra en el repositorio. Al implementarlo se importa el archivo generado en MatLab y se observa la siguiente señal:

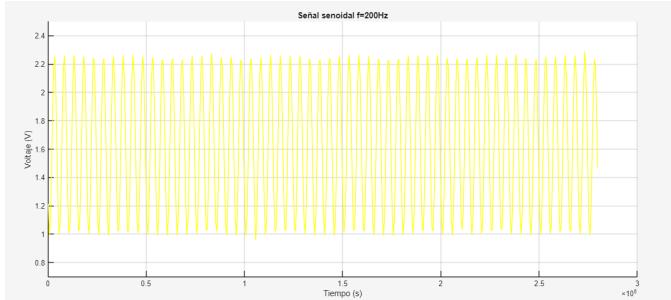


Imagen 25. Señal seno en MatLab

En el cual limitando y filtrando los períodos genera la siguiente señal:

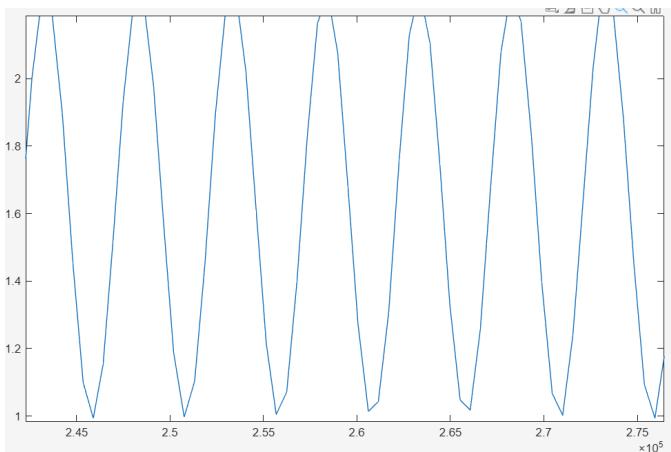


Imagen 26. Señal seno en MatLab

Se analizaron las posibilidades y restricciones del dispositivo, realizando pruebas y comparando el resultado respecto a los valores teóricos para una señal senoidal.

Ahora bien, ¿Qué es el Jitter y qué implicaciones tiene en el proceso de codificación de la fuente?

El *jitter* es la variación aleatoria o no deseada en los instantes de muestreo de una señal. En lugar de que las muestras se tomen en intervalos de tiempo perfectamente uniformes, aparecen pequeñas desviaciones (adelantos o retrasos) respecto al tiempo ideal de muestreo. Dicho de otra manera, el jitter es la inestabilidad temporal en el proceso de muestreo.

En la codificación de la fuente (digitalización de una señal analógica), el jitter afecta la precisión con la que se representan las muestras:

- Distorsión espectral: al no tomarse las muestras en instantes exactos, la reconstrucción en frecuencia de la señal se ve alterada, introduciendo ruido adicional.
- Error en amplitud: aunque la señal se cuantiza con buena resolución, si el muestreo ocurre en un instante adelantado o atrasado, el valor leído puede no coincidir con el real de la onda, se traduce en un error



de cuantización adicional.

- Degrado del SNR y ENOB: el jitter reduce la relación señal a ruido efectiva y disminuye el número efectivo de bits del convertidor A/D.
- Problemas en señales de alta frecuencia: mientras mayor sea la frecuencia de la señal a muestrear, más crítico se vuelve el efecto del jitter, porque pequeñas variaciones en el tiempo producen grandes errores en el valor de la muestra.

Para este apartado una vez tenemos claro que es el jitter se puede implementar en el código donde se toma y ayuda a tomar la estabilidad de tiempo, esto se logra apreciar en la siguiente línea de código:

```
# ===== Adquisición con jitter =====

print("Adquiriendo      datos      con
      jitter...")

t0 = utime.ticks_us()

for i in range(N):

    data[i] = adc.read_u16()

    tiempos[i] = utime.ticks_diff(utime.ticks_us(),
                                   t0)

# Calcular periodo con jitter

    jitter = urandom.randint(-jitter_max,
                           jitter_max)

    utime.sleep_us(periodo_us +
                  jitter)

print("Adquisición completa.")
```

Recordando lo que hace esta parte del código:

1. Lee N muestras del ADC → cada una se guarda en `data[i]`.
2. Registra el tiempo real en que se toma cada muestra en `tiempos[i]`.
3. Introduce jitter → en vez de esperar siempre el mismo periodo (`periodo_us`), se suma o resta un pequeño valor aleatorio ($\pm \text{jitter_max}$).

En esta parte experimental el periodo ideal es 500 ms (para 2000 Hz de muestreo), y con un jitter de $\pm 20 \mu\text{s}$, el intervalo entre muestras estará entre 480 y 520 ms.

Cada muestra siguiente debería aumentar aproximadamente en $500 \mu\text{s}$, pero no de manera exacta. Los primeros tiempos reales se muestran a continuación:

Indice	Tiempo_u
0	181 1.21360
1	785 0.99199
2	1318 1.10
3	1877 1.46
4	2406 1.87
5	2940 2.18
6	3505 2.25
7	4060 2.01
8	4619 1.58
9	5183 1.19
10	5717 1.00

Imagen 27. Tabulación del archivo de muestras

Por lo cual hallando los intervalos de cada muestra

- Entre 0 y 1 → $785 - 181 = 604 \mu\text{s}$
- Entre 1 y 2 → $1318 - 785 = 533 \mu\text{s}$
- Entre 2 y 3 → $1877 - 1318 = 559 \mu\text{s}$
- Entre 3 y 4 → $2406 - 1877 = 529 \mu\text{s}$
- Entre 4 y 5 → $2940 - 2406 = 534 \mu\text{s}$
- Entre 5 y 6 → $3505 - 2940 = 565 \mu\text{s}$

El intervalo entre muestras se encuentra en un rango de aproximadamente 530 a 560 ms, lo que corresponde a una frecuencia de muestreo cercana a 1.8 kHz, muy próxima al objetivo de 2 kHz. La ligera variación observada en esos intervalos, de unos ± 30 ms, representa el jitter, es decir, el pequeño “ruido temporal” introducido intencionalmente mediante la función `urandom.randint(jitter_max, jitter_max)`. Aun con estas variaciones, los tiempos mantienen una tendencia de crecimiento lineal, lo que indica que la adquisición avanza de forma estable en promedio, aunque con pequeñas irregularidades entre muestra y muestra

La columna de tiempos de la imagen 27 (181, 785, 1318, ...) refleja los instantes exactos de muestreo. Al calcular las diferencias, se observa que las muestras están separadas en promedio por $\sim 540 \mu\text{s}$, lo que corresponde a una frecuencia real cercana a 1.8 kHz. La ligera variación entre intervalos confirma la presencia del jitter, que hace que las muestras no caigan en un espacio perfectamente uniforme.

Para finalizar, ¿Qué alternativas existen con el dispositivo Raspberry Pi Pico 2W para realizar un muestreo exitoso considerando las características técnicas y posibilidades del dispositivo ?



En el caso del Raspberry Pi Pico 2W, existen varias alternativas para realizar un muestreo exitoso considerando sus características técnicas. Una de las más importantes es el uso de temporizadores de hardware que permiten generar interrupciones periódicas, asegurando que las lecturas del ADC se realicen en intervalos de tiempo uniformes, mucho más precisos que los retardos por software como `sleep_us`, los cuales tienden a introducir jitter. Otra opción avanzada consiste en emplear los bloques PIO (Programmable I/O), que ofrecen gran flexibilidad para controlar periféricos y señales externas con sincronización exacta, pudiendo configurarse para capturar datos del ADC o gestionar protocolos de comunicación sin depender de la CPU. Además, el Pico 2W puede aprovechar el DMA (Direct Memory Access) para transferir datos del ADC directamente a memoria, reduciendo la carga del procesador y evitando pérdidas de información durante adquisiciones largas o de alta velocidad. Cuando se requiere mayor resolución o estabilidad, también es posible integrar convertidores A/D externos conectados por interfaces como SPI o I²C, los cuales suelen ofrecer mejores prestaciones que el ADC interno. En conjunto, estas herramientas permiten diseñar sistemas de adquisición robustos, con menor jitter, mejor aprovechamiento de los recursos y resultados más confiables en la digitalización de señales.

IV. CONCLUSIONES

1. La práctica permitió comprobar experimentalmente el funcionamiento del conversor A/D de la Raspberry Pi Pico 2W, logrando digitalizar señales senoidales en un rango de frecuencias hasta donde lo permite el teorema de Nyquist.
2. El uso de la Transformada Rápida de Fourier (FFT) evidenció cómo el número de puntos influye en la resolución espectral, observándose mejoras notables en la definición de los picos a medida que se incrementa el tamaño de la FFT.
3. La introducción intencional de *jitter* en el muestreo mostró sus efectos negativos sobre la reconstrucción de la señal, afectando la precisión en amplitud y frecuencia, así como degradando la relación señal-ruido efectiva (SNR).
4. Se confirmó que el muestreo estable con funciones de retardo por software (`sleep_us`) presenta limitaciones, siendo más apropiado implementar temporizadores de hardware, módulos PIO o DMA para obtener adquisiciones más precisas y confiables.
5. El experimento permitió reforzar la importancia del teorema de Nyquist, ya que al superar la mitad de la

frecuencia de muestreo se produjeron aliasing y distorsiones notorias en el dominio de la frecuencia.

6. A pesar de las limitaciones propias del ADC interno de la Raspberry Pi Pico 2W, se demostró que, mediante técnicas adecuadas de procesamiento y control de temporización, es posible realizar adquisiciones robustas y con resultados comparables a los teóricos.

V. REFERENCIAS

- [1] Raspberry Pi Foundation, *Raspberry Pi Pico 2 W Datasheet*,
- [2] R. Long, “Measuring Small Voltages With Pi Pico ADC and Comparison With …”, *Instructables*
- [3] Analog Devices, “Understanding Noise, ENOB, and Effective Resolution in Analog-to-Digital Converter Circuits

VI. REPOSITORIO DE GITHUB

<https://github.com/Vallentincita/Comunicaci-n-digital-TEL-A->