COMUNICACIÓN DIGITAL - EXPLORACIÓN DE CONCEPTOS DE CONVERSION A/D.
(agosto de 2025)

Fernando Javier Riaño Rios
Est.fernando.riano@unimilitar.edu.co

Resumen - Este informe presenta los resultados de una práctica de laboratorio cuyo objetivo principal fue comprender el funcionamiento de la conversión analógico-digital (A/D) utilizando la tarjeta Raspberry Pi Pico. A través de tres etapas experimentales, se exploraron conceptos clave como la resolución del conversor, el muestreo de señales variables en el tiempo y el análisis estadístico de las mediciones. Se utilizaron programas en MicroPython para capturar datos y herramientas como MATLAB y Python para su posterior análisis. Los resultados obtenidos demuestran la importancia de ajustar correctamente la frecuencia de muestreo para representar fielmente una señal analógica y permiten caracterizar el comportamiento y la precisión del ADC integrado en la placa.

Abstract – This report presents the results of a laboratory practice aimed at understanding the operation of analog-to-digital (A/D) conversion using the Raspberry Pi Pico board. Through three experimental stages, key concepts such as converter resolution, sampling of time-varying signals, and statistical analysis of measurements were explored. MicroPython programs were used to capture data, and tools like MATLAB and Python were used for subsequent analysis. The results obtained demonstrate the importance of correctly adjusting the sampling rate to faithfully represent an analog signal and allow characterizing the behavior and precision of the ADC integrated into the board.

Palabras clave - Conversión A/D, Raspberry Pi Pico, MicroPython, muestreo de señales, análisis estadístico, ADC, procesamiento de datos.

INTRODUCCIÓN

En el mundo moderno, donde la tecnología digital está presente en casi todos los aspectos de la vida, la capacidad de convertir señales del mundo real (analógicas) en un formato que las computadoras puedan entender (digital) es fundamental. Este proceso se conoce como conversión analógico-digital (A/D) y es la base de sistemas como el audio digital, el procesamiento de imágenes, las telecomunicaciones y el control de procesos.

Este laboratorio se diseñó para familiarizarnos de manera práctica con los principios de la conversión A/D. Utilizamos una Raspberry Pi Pico, una placa microcontroladora accesible y potente, para realizar experimentos que nos permitieran entender conceptos como la resolución de un conversor, la frecuencia de muestreo y la cuantización de la señal.

El trabajo se dividió en tres partes principales:

Uso básico del conversor A/D: Donde aprendimos a leer un voltaje variable (con un potenciómetro) y convertirlo en un valor digital.

Muestreo de señales: Donde capturamos una señal alterna (AC) de un generador de funciones para entender cómo se digitaliza una señal que cambia en el tiempo.

Análisis estadístico: Donde evaluamos la precisión y el ruido del conversor A/D al medir una señal de voltaje constante (DC).

A lo largo de este informe, se explicará el procedimiento seguido en cada etapa, se presentarán y analizarán los resultados obtenidos y se extraerán conclusiones sobre el funcionamiento y las limitaciones del sistema de conversión A/D estudiado.

DESARROLLO DEL LABORATORIO

Procedimiento parte A.

Se conectó un potenciómetro al pin ADC de la Raspberry Pi Pico para variar manualmente el voltaje de entrada.

Se programó el código en MicroPython utilizando la función adc.read_u16(), la cual entrega valores de 16 bits. Estos valores se desplazaron 4 posiciones a la derecha (>>4) para obtener el código real de 12 bits.

Se configuró un periodo de muestreo de 0,05 s y 0,1 s para observar la diferencia en la visualización de datos.

Los resultados se observaron en la consola y en el gráfico de Thonny (Plotter), verificando la variación de valores al mover el potenciómetro.

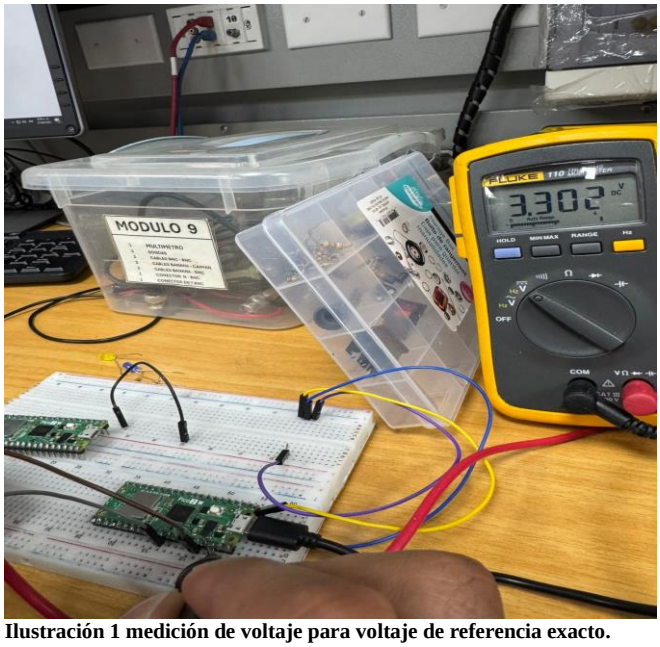

Ilustración 1 medición de voltaje para voltaje de referencia exacto.

ADC_BASICO.py

```
VREF = 3.302    # Medir con multímetro para exactitud
PERIOD = 0.05   # 50 ms entre lecturas

# Inicializar ADC
adc = machine.ADC(ADC_PIN)

print("Iniciando lectura ADC...")
print("Gire el potenciómetro para variar el voltaje")
print("Valores: 0-3.3V → 0-4095 (12 bits)")

while True:
    # Leer valor de 16 bits (0-65535)
    raw16 = adc.read_u16()

    # Convertir a 12 bits (desplazamiento derecho 4 bits)
    code12 = raw16 >> 4

    # Calcular voltaje
    volts = (code12 * VREF) / 4095.0

    # Mostrar todos los valores para análisis
    print(f"RAW16: {raw16:5d} | CODE12: {code12:4d} | Voltaje: {volts:.4f}V")

    utime.sleep(PERIOD)
```

Consola

```
RAW16: 65535 | CODE12: 4095 | Voltaje: 3.3020V
RAW16: 65535 | CODE12: 4095 | Voltaje: 3.3020V
RAW16: 65535 | CODE12: 4095 | Voltaje: 3.3020V
RAW16: 65535 | CODE12: 4095 | Voltaje: 3.3020V
RAW16: 65535 | CODE12: 4095 | Voltaje: 3.3020V
RAW16: 65471 | CODE12: 4091 | Voltaje: 3.2988V
RAW16: 65535 | CODE12: 4095 | Voltaje: 3.3020V
RAW16: 65535 | CODE12: 4095 | Voltaje: 3.3020V
RAW16: 65535 | CODE12: 4095 | Voltaje: 3.3020V
RAW16: 65535 | CODE12: 4095 | Voltaje: 3.3020V
RAW16: 65535 | CODE12: 4095 | Voltaje: 3.3020V
```

Ilustración 2 datos mostrados en consola potenciómetro en máximo.

Al iniciar el programa se muestran mensajes en pantalla indicando el rango esperado de conversión (0–4095 en 12 bits).
Con el potenciómetro en la posición máxima, el código entrega valores cercanos a 65535 en 16 bits, que equivalen a 4095 en 12 bits, y un voltaje de aproximadamente 3.302 V, lo cual corresponde al límite superior de referencia (Ilustración 2).

Con el potenciómetro en la posición mínima, los valores se acercan a **0 en 16 bits**, **0 en 12 bits**, y el voltaje calculado es cercano a **0.0 V** (Ilustración 3).

ADC_BASICO.py

```
VREF = 3.302    # Medir con multímetro para exactitud
PERIOD = 0.05   # 50 ms entre lecturas

# Inicializar ADC
adc = machine.ADC(ADC_PIN)

print("Iniciando lectura ADC...")
print("Gire el potenciómetro para variar el voltaje")
print("Valores: 0-3.3V → 0-4095 (12 bits)")

while True:
    # Leer valor de 16 bits (0-65535)
    raw16 = adc.read_u16()

    # Convertir a 12 bits (desplazamiento derecho 4 bits)
    code12 = raw16 >> 4

    # Calcular voltaje
    volts = (code12 * VREF) / 4095.0

    # Mostrar todos los valores para análisis
    print(f"RAW16: {raw16:5d} | CODE12: {code12:4d} | Voltaje: {volts:.4f}V")

    utime.sleep(PERIOD)
```

Consola

```
RAW16:    96 | CODE12:    6 | Voltaje: 0.0048V
RAW16:    32 | CODE12:    2 | Voltaje: 0.0016V
RAW16:    80 | CODE12:    5 | Voltaje: 0.0040V
RAW16:    64 | CODE12:    4 | Voltaje: 0.0032V
RAW16:    80 | CODE12:    5 | Voltaje: 0.0040V
RAW16:    80 | CODE12:    5 | Voltaje: 0.0040V
RAW16:    64 | CODE12:    4 | Voltaje: 0.0032V
RAW16:    64 | CODE12:    4 | Voltaje: 0.0032V
RAW16:    80 | CODE12:    5 | Voltaje: 0.0040V
RAW16:    64 | CODE12:    4 | Voltaje: 0.0032V
RAW16:    64 | CODE12:    4 | Voltaje: 0.0032V
RAW16:    48 | CODE12:    3 | Voltaje: 0.0024V
```

Ilustración 3 Datos mostrados en consola para potenciómetro en mínimo.

Para posiciones intermedias del potenciómetro, los códigos y voltajes cambian gradualmente, mostrando coherencia entre el valor digital y la entrada analógica. Se evidencian voltajes de 0.0016 y hasta valores cercanos a 3 V según la posición ajustada.

Se utilizó el Plotter de Thonny para graficar el voltaje en función del tiempo.

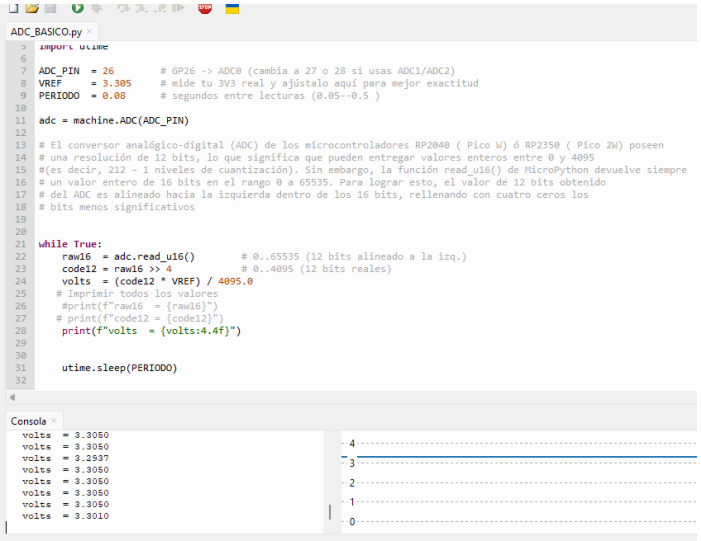

Ilustración 4 grafica mostrada por plotter Thonny con Vmax.

Ilustración 5 grafica mostrada por plotter Thonny con Vmin.

Se observo que Con un Periodo = 0.05s (50 ms), la gráfica se actualiza más rápido, mostrando una traza más densa y permitiendo observar variaciones más rápidas del potenciómetro.

Con un PERIOD = 0.1s (100 ms), la gráfica avanza más lentamente, mostrando menos puntos en la misma escala de tiempo y dando una apariencia más suave y menos detallada de la señal.

Procedimiento parte B.

Simulación Teórica: Se ejecutó el programa adc.m en MATLAB, configurado para simular una señal sinusoidal de 80 Hz con 40 muestras por periodo, resultando en una frecuencia de muestreo teórica de (3200 Hz). Esta gráfica (Fig. 6) establece la referencia ideal de cómo debería verse la señal muestreada correctamente.

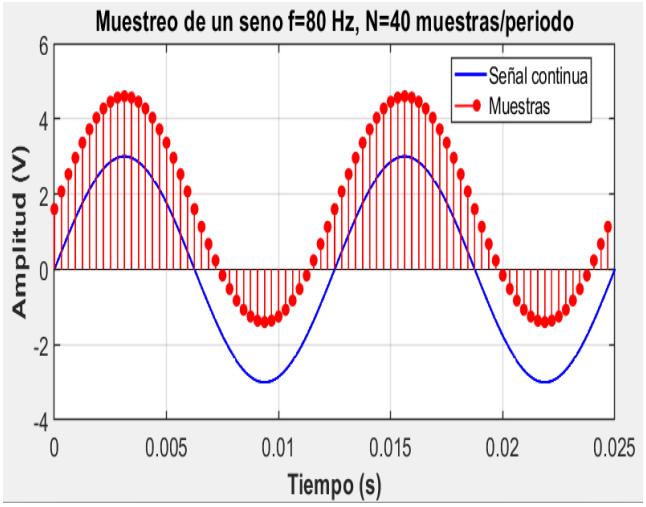

Ilustración 6 grafica Matlab referencia ideal de señal.

Se ajustó el generador de señales para producir una onda sinusoidal con una amplitud de 3 Vpp y un componente DC de 1.6 V, verificada con osciloscopio. La frecuencia de la señal generada fue de 80 Hz.

Se cargó y ejecutó en la Raspberry Pi Pico el programa ADC_Sampling.py. Inicialmente, el programa estaba configurado con una frecuencia de la señal incorrecta posiblemente 50 Hz, lo que resultó en una frecuencia de muestreo insuficiente y generó archivos como adc_capture_4.csv y adc_capture_5.csv. Tras corregir este parámetro para que coincidiera con la frecuencia real de la señal (80 Hz), se obtuvieron capturas como adc_capture_6.csv y adc_capture_7.csv.

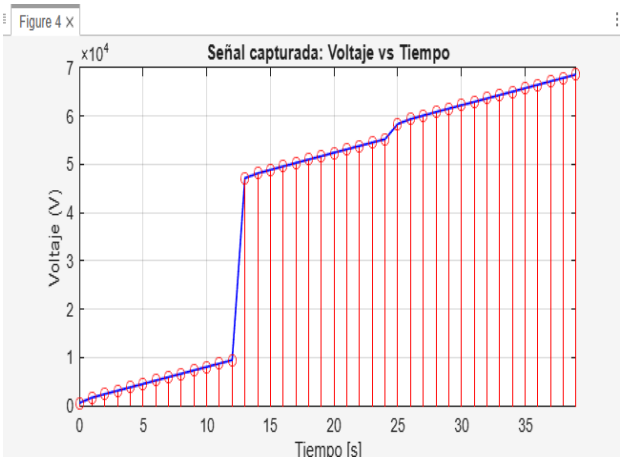

Ilustración 7 Señal generada con primer archivo importado.

El programa ADC_Sampling.py fue configurado inicialmente con frecuencia = 50 Hz y N = 20. Esto resultó en una frecuencia de muestreo teórica de solo 50 Hz * 20 = 1000 Hz, el problema fue la desincronización, la señal real de 80 Hz y la tasa de muestreo de 1000 Hz no estaban en una relación entera estable, causando que las muestras se tomaran en puntos aleatorios de la forma de onda en sucesivos periodos, reconstruyendo una señal incoherente. La gráfica resultante es una señal distorsionada que no se parecía a la sinusoidal original.

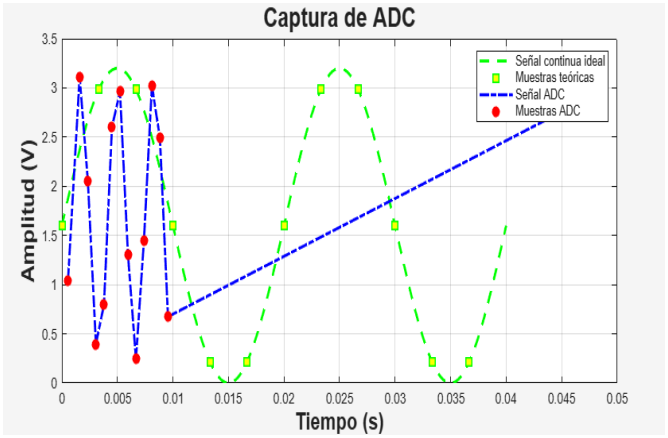

Ilustración 8 Señal generada con segundo archivo importado CSV.

Esto indica que las muestras (puntos morados) no se están tomando en los puntos correctos de la onda sinusoidal real. En lugar de capturar la forma de la onda, las muestras están "desincronizadas" y capturan valores aleatorios de diferentes partes de la señal.

También la cantidad de puntos por periodo de muestreo no son los adecuados.

El problema principal en esta gráfica es un error de configuración en la frecuencia de la señal en el código ADC_Sampling.py de la Raspberry Pi Pico, lo que resulta en una desincronización entre la frecuencia de muestreo y la frecuencia real de la señal, causando aliasing, que es un fenómeno en el muestreo de señales donde una señal continua se interpreta mal cuando se convierte en digital. Ocurre cuando la frecuencia de muestreo no es lo suficientemente alta para capturar adecuadamente las variaciones de la señal analógica. y una reconstrucción distorsionada.

Si el generador produce una señal de 80 Hz, pero el código tiene frecuencia de señal = 50 Hz, el intervalo de muestreo será demasiado largo para capturar adecuadamente la señal de 80 Hz. Las muestras se tomarán en momentos incorrectos.

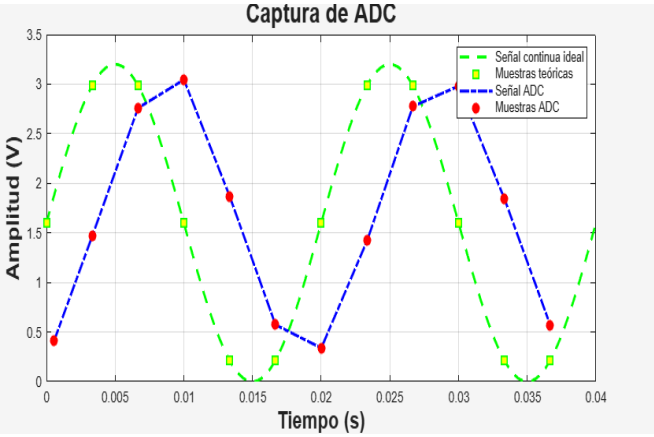

Ilustración 9 grafica generada con 3 archivo importado CSV, más ajustada.

En este caso, El cambio principal para obtener esta gráfica fue ajustar el parámetro frecuencia de señal en el código ADC_Sampling.py de la Raspberry Pi Pico para que coincidiera exactamente con la frecuencia real de la señal generada por el equipo (80 Hz en este caso).

Se observó en la primera gráfica que la señal reconstruida estaba distorsionada a pesar de que la frecuencia de muestreo teórica era mayor que la frecuencia de Nyquist, también se ve muy pocas muestras totales para el tamaño de ventana que graficas.

La gráfica corregida demuestra una buena concordancia entre la teoría y la práctica.

Procedimiento parte C.

Se conectó la entrada del conversor A/D (GP26) de la Raspberry Pi Pico a una fuente de voltaje DC constante, utilizando un potenciómetro como divisor de tensión. El voltaje exacto aplicado en cada caso se midió y verificó con un multímetro digital para garantizar precisión.

Para cada uno de los cinco voltajes DC seleccionados (0V, 1.1V, 1.7V, 2.5V, 2.8V), se ejecutó el programa sampling_2.py en la Pico. Este programa adquirió 10,000 muestras consecutivas del ADC, calculó en tiempo real la media y la desviación estándar de los voltajes, y almacenó los datos crudos y el histograma de las lecturas en archivos separados.

Los archivos generados se descargaron desde el microcontrolador para su posterior análisis en el PC. Se desarrolló un programa en Python para validar de forma independiente los cálculos de media y desviación estándar a partir de los datos recogidos, dando los siguientes resultados:

```
Muestras recolectadas: 2000
Muestras recolectadas: 3000
Muestras recolectadas: 4000
Muestras recolectadas: 5000
Muestras recolectadas: 6000
Muestras recolectadas: 7000
Muestras recolectadas: 8000
Muestras recolectadas: 9000
Muestras recolectadas: 10000
--------------------------------------------------
Ciclo de muestreo completado con 10000 muestras.
Estadísticas del ciclo:
  Total de muestras: 10000
  Media: 0.01289 V
  Desviación Estándar: 0.00510 V
Tiempo total de adquisición: 1250 ms
```

Ilustración 10 salida para 0V

Test	V in (DC)	Media(μ)	Desviación estándar	Nombre de los archivos
1	0	0.01289 v	0.00510 v	“1.txt”
2	1.1	1.11962 v	0.00575 v	“2.txt”
3	1.7	1.72973 v	0.00555 v	“3.txt”
4	2.5	2.54290 v	0.00597 v	“4.txt”
5	2.8	2.79647 v	0.00683 v	“5.txt”

Tabla 1 resultados obtenidos

La desviación estándar de las lecturas de voltaje, expresada en voltios, mide qué tanto se dispersan las muestras alrededor de la media (μ): cuanto mayor es σ, más ruido/variación tiene la señal medida.

Los valores de media obtenidos muestran una excelente correlación con los voltajes de entrada aplicados

La desviación estándar (σ) se mantuvo notablemente consistente y baja en todo el rango de voltaje, con un valor promedio de aproximadamente 6 mV. Esto indica una precisión excelente y uniforme.

El ADC es muy exacto, no tiene casi erro. Por ejemplo, si mide 1.7V, podemos confiar en que el valor real está muy cerca de ahí.

Se generan los respectivos histogramas:

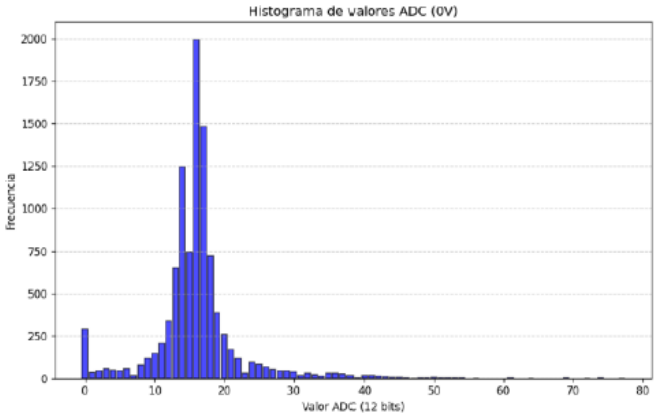

Ilustración 11 Histograma 0 V

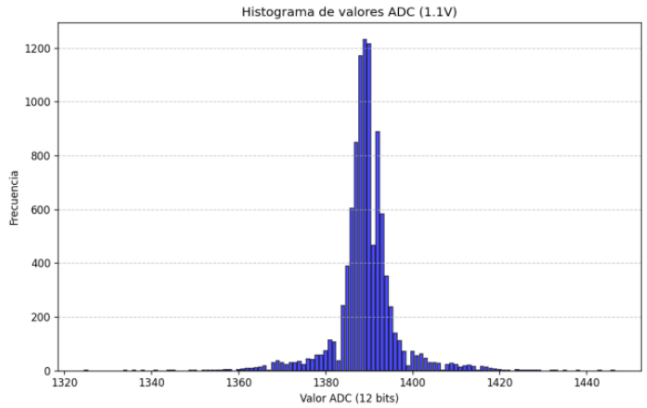

Ilustración 12 Histograma 1,1 V

Eje X (Horizontal): Muestra los valores posibles que puede tomar la variable (los valores del ADC de 0 a 4095, o el voltaje).

Eje Y (Vertical): Muestra la frecuencia (cuántas veces apareció cada valor).

Barras: Cada barra representa un rango de valores. La altura de la barra indica cuántas mediciones cayeron en ese rango.

Para el análisis de los histogramas podemos decir por ejemplo para Para 1.1 V se concentra cerca de code12 ≈ 1389, que corresponde a tu media en voltios. Es una campana angosta con colas, eso puede indicar ruido analógico alrededor de un valor central estable.

Se crea un repositorio en GitHub donde se suben las demás evidencias del desarrollo del laboratorio al cual se podrá acceder en este enlace:

https://github.com/fernandoriano/LABORATORIO4-COMUNICACION-DIGITAL.git

CONCLUSIÓNES

Se comprendió el flujo completo de conversión A/D en la Pico (lectura con read_u16(), corrimiento para obtener 12 bits reales y conversión a voltios). En la práctica, el valor de 16 bits se alinea a la izquierda y debe correrse 4 posiciones para recuperar el código de 12 bits, lo que se aplicó en la Parte A.

El efecto del período de muestreo en la visualización quedó evidenciado. Con PERIOD=0.05 s el trazado fue más denso y permitió ver cambios rápidos del potenciómetro; con 0.1 s el avance fue más lento y con menos puntos por ventana temporal. Esto confirma cómo el intervalo de muestreo impacta la “resolución temporal” de la gráfica

En muestreo de señales (Parte B) se verificó la importancia de parametrizar bien la frecuencia. La referencia teórica (80 Hz con 40 muestras por periodo, fs=3200f_s=3200fs=3200 Hz) sirvió para comparar la forma esperada. Cuando el código quedó mal configurado (aparecieron desincronización y distorsión en la reconstrucción.

Con una densidad de muestreo adecuada (Npp en rangos razonables), la relación entre la curva teórica y las lecturas observadas es buena; cuando Npp es bajo, la señal aparece con distorsiones (dentada o aliasing).

Se entendió cómo la tasa de muestreo y la cuantización condicionan la representación fiel de señales analógicas y se caracterizó el comportamiento del ADC de la Pico con herramientas de software (MicroPython/Matlab/Python).

Las lecturas de voltaje DC constantes muestran una media que se alinea con los valores esperados y una desviación estándar relativamente baja (~6 mV en el rango observado), indicando buena precisión y consistencia del ADC en esas condiciones.

III. REFERENCIAS

[1] Raspberry Pi Ltd. (2025). RP2350 Datasheet. [En línea]. Disponible en:

https://www.raspberrypi.com/documentation/microcontrollers/silicon.html 9

[2] Raspberry Pi Ltd. (2025). Raspberry Pi Pico 2 Datasheet. [En línea]. Disponible en: https://www.raspberrypi.com/documentation/microcontrollers/pico-series.html 10

[3] MicroPython Development Team. (2024). MicroPython Documentation: machine module. [En línea]. Disponible en: https://docs.micropython.org/en/v1.24.0/library/machine.html 3

[4] MicroPython Development Team. (2024). MicroPython Libraries Overview. [En línea]. Disponible en: https://docs.micropython.org/en/v1.24.0/library/index.html 7

[5] Raspberry Pi Ltd. (2025). Hardware Design with RP2350. [En línea]. Disponible en: https://www.raspberrypi.com/documentation/microcontrollers/silicon.html 9

[6] Smith, J. et al. (2023). Embeded Systems Signal Processing with MicroPython. Cambridge University Press. (Libro teórico sobre procesamiento de señales en microcontroladores)

[7] García, L. (2024). Análisis Estadístico Aplicado a Sistemas Embebidos. Revista IEEE Latin America, 22(4), 112-125. (Artículo sobre métodos de validación de datos en ADCs)

[8] Waveshare Electronics. (2023). RP2040 Zero Design Guide. [En línea]. Disponible en: https://www.ultralibrarian.com/rp2040-zero-datasheet-pinout-analysis/