Informe de Laboratorio: Exploración de Conceptos de Conversión A/D con Raspberry Pi Pico

Miguel Andrey Peña Cárdenas
Programa de Ingeniería en Telecomunicaciones
Universidad Militar Nueva Granada
Bogotá, Colombia
est.miguela.pena@unimilitar.edu.co

***Resumen*—Este informe detalla el procedimiento y los resultados obtenidos en la práctica de laboratorio sobre la conversión analógico-digital (A/D) utilizando el microcontrolador Raspberry Pi Pico. Se exploraron los conceptos fundamentales de resolución, voltaje de referencia, muestreo de señales y análisis estadístico de las lecturas del conversor. Se realizaron tres experimentos principales: la lectura de voltajes DC variables, el muestreo de una señal senoidal y el análisis estadístico de la estabilidad del conversor. Los resultados permitieron validar el comportamiento teórico del ADC del RP2040, comprender la relación entre los valores crudos de 16 bits y los valores reales de 12 bits, y analizar la calidad de la señal muestreada y la precisión de las lecturas.**

***Index Terms*—Conversión A/D, Raspberry Pi Pico, RP2040, MicroPython, Muestreo de Señales, Cuantización, Análisis Estadístico.**

I. Introducción y Marco Teórico

En el centro de todo sistema de comunicación digital que debe interactuar con el mundo real está el Conversor Analógico-Digital (ADC). Este dispositivo actúa como un puente que traduce señales analógicas continuas como el voltaje o la temperatura al lenguaje discreto que entienden los microcontroladores y procesadores. El propósito de esta práctica es hacer más claro y cercano este proceso, utilizando la Raspberry Pi Pico.

II. Marco Teórico

II-A. Conversión Analógico-Digital (A/D)

El proceso de A/D consta de dos pasos principales:

- **Muestreo:** Tomar ”fotografías”(muestras) del valor de la señal analógica a intervalos de tiempo regulares. La frecuencia a la que se toman estas muestras se conoce como frecuencia de muestreo (f_s). El teorema de Nyquist-Shannon establece que para reconstruir una señal sin pérdida de información (aliasing), f_s debe ser al menos el doble de la frecuencia más alta presente en la señal ($f_s \geq 2f_{max}$).
- **Cuantización:** Asignar un valor digital (un número) a cada muestra. La precisión de esta asignación depende de la **resolución** del ADC, medida en bits. Un ADC de N bits puede representar 2^N niveles de voltaje distintos.

II-B. El ADC del Raspberry Pi Pico (RP2040)

El microcontrolador RP2040 de la Pi Pico tiene un ADC con una **resolución de 12 bits**. Esto significa que puede distinguir $2^{12} = 4096$ niveles de voltaje, que corresponden a valores enteros de 0 a 4095.

Una peculiaridad de MicroPython en la Pico es que la función ‘adc.read_u16()‘ no devuelve directamente el valor de 12 bits, sino un valor de 16 bits (0-65535). Esto se hace por compatibilidad y eficiencia. El valor real de 12 bits se alinea a la izquierda dentro del registro de 16 bits, rellenando con ceros los 4 bits menos significativos. Para obtener el valor real de 12 bits (‘code12‘), es necesario hacer un corrimiento de 4 bits a la derecha del valor leído (‘raw16‘):

$$\text{code12} = \text{raw16} \gg 4 \tag{1}$$

Una vez obtenido el valor de 12 bits, se puede convertir a voltaje usando el voltaje de referencia (‘VREF‘), que es el voltaje máximo que el ADC puede medir, típicamente 3.3V en la Pico.

$$\text{Voltaje} = \frac{\text{code12}}{4095} \times V_{REF} \tag{2}$$

III. Desarrollo del Laboratorio

La práctica se dividió en tres partes principales para explorar diferentes facetas del ADC.

III-A. Parte A: Uso Básico del Conversor A/D

En esta fase inicial, el objetivo era familiarizarse con la lectura de un voltaje DC constante. Se utilizó una fuente de voltaje para aplicar un voltaje variable entre 0V y 3.3V al pin GP26 del ADC. El siguiente código en MicroPython fue empleado para leer y mostrar el voltaje.

```
import machine
import utime

ADC_PIN = 26
VREF = 3.323  # Voltaje medido en el pin 3V3
PERIOD = 0.05

adc = machine.ADC(ADC_PIN)

```

```
while True:
    raw16 = adc.read_u16()
    # Corrimiento a la derecha para obtener 12 bits
    code12 = raw16 >> 4
    # Conversion a voltaje
    volts = (code12 * VREF) / 4095.0
    print(f"{volts:.4f}")
    utime.sleep(PERIOD)
```

Listing 1: Código para lectura de voltaje DC.

Se realizaron varias pruebas ajustando el voltaje de referencia ('VREF') en el código para que coincidiera con voltajes de entrada específicos. La Figura 1 muestra el resultado de una de estas mediciones usando el plotter de Thonny, donde la entrada era un voltaje DC estable de aproximadamente 3.2V. Se puede observar una pequeña fluctuación en las lecturas, lo cual es normal y se debe al ruido inherente del sistema.

Figura 1: Lectura de un voltaje DC cercano a 3.2V con VREF=3.323V.

III-B. Análisis de la Conversión de 16 a 12 bits

Para una comprensión completa de los resultados, es fundamental detallar cómo la función `read_u16()` de MicroPython presenta el valor nativo de 12 bits del ADC del RP2040. Aunque el ADC tiene una resolución de 12 bits (4096 niveles), la función devuelve un valor de 16 bits (65536 niveles).

III-C. Fórmulas de la Conversión

Definimos dos variables clave:

- **`code12`**: El valor real del ADC, un entero de 12 bits en el rango [0, 4095].
- **`raw16`**: El valor crudo que leemos de la función `read_u16()`, un entero de 16 bits en el rango [0, 65535].

III-C1. Alineación a la izquierda (Realizada por el sistema): El sistema realiza un **desplazamiento de 4 bits a la izquierda** ($\ll 4$), lo cual es matemáticamente equivalente a multiplicar por $2^4 = 16$.

$$\text{raw16} = \text{code12} \ll 4 \quad \equiv \quad \text{raw16} = \text{code12} \times 16 \tag{3}$$

III-C2. Recuperación del valor (Realizada en nuestro código): Para revertir el proceso, aplicamos un **desplazamiento de 4 bits a la derecha** ($\gg 4$), lo que equivale a una división entera por 16.

$$\text{code12} = \text{raw16} \gg 4 \quad \equiv \quad \text{code12} = \left\lfloor \frac{\text{raw16}}{16} \right\rfloor \tag{4}$$

III-C3. Ejemplo 1: Media Escala (code12 = 2048): Este valor corresponde a la mitad del rango de medición.

- **Binario**: `1000 0000 0000`
- **Alineación**: El sistema calcula `2048 « 4 = 32768`. `El valor de 16 bits es 1000 0000 0000` **`0000.`**
- »**Recuperación:** `Nuestro código calcula 32768 » 4 = 2048`, obteniendo el valor central correcto.

Para poder obtener un valor correcto en nuestro codigo tenemos que cambiar el Vref con la formula 5, reemplazamos code12 por 2048 que es nuestro ejemplo 1, así:

$$V = \frac{\texttt{code12} \times V_{\text{REF}}}{4095} \tag{5}$$

$$V = \frac{2048 \times 3{,}323}{4095} = 1{,}61V \tag{6}$$

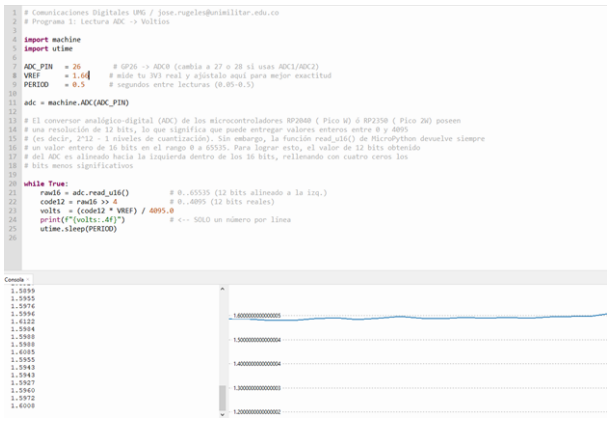


Figura 2: Ejemplo 2048 con un valor de Vref=1.61.

III-C4. Ejemplo 2: Un Cuarto de Escala (code12 = 1024): Este valor representa un cuarto del rango total del ADC.

- **Binario**: `0100 0000 0000`
- **Alineación**: El sistema calcula `1024 « 4 = 16384`. `El valor de 16 bits es 0100 0000 0000` **`0000.`**
- »**Recuperación:** `Nuestro código calcula 16384 » 4 = 1024`, restaurando el valor original.

En la figura 3 se observa un Vref de 0.831, este valor se calcula con la formula 5, pero ahora reemplazamos code12 por 1024, en el ploter se observa una salida de 0.8, valor muy cercano al teórico.

Figura 3: Ejemplo 1024 con un valor de Vref=0.831.

III-C5. Ejemplo 3: Un Octavo de Escala (code12 = 512): Este valor corresponde a un octavo del rango.

- **Binario**: `0010 0000 0000`
- **Alineación**: El sistema calcula `512 « 4 = 8192`. `El valor de 16 bits es 0010 0000 0000` **`0000.`**
- **Recuperación:** `Nuestro código calcula 8192 » 4 = 512`, obteniendo el valor correcto.

Figura 4: Ejemplo 512 con un valor de Vref=0.415.

III-D. Parte B: Muestreo de Señales

El siguiente paso fue capturar una señal que varía en el tiempo. Se utilizó un generador de señales para crear la señal senoidal de 3V pico a pico con un offset DC de 1.6V, y una frecuencia de 625 Hz. Esta señal fue conectada al ADC.

Primero, se realizó una simulación en MATLAB para visualizar el concepto de muestreo (Figura 5).

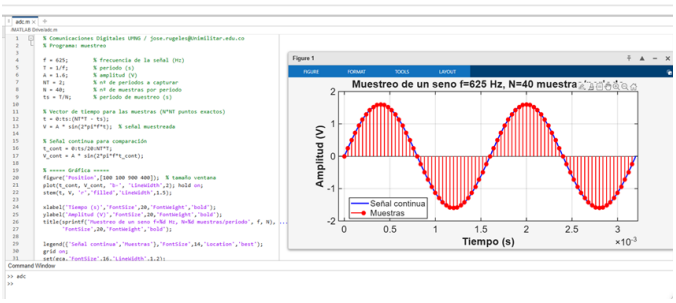


Figura 5: Simulación en MATLAB del muestreo de una señal senoidal.

Cargue el programa ADC_Sampling.py en el microcontrolador. Analice el código. ¿ Que nombre tiene el archivo .CSV generado por el programa?

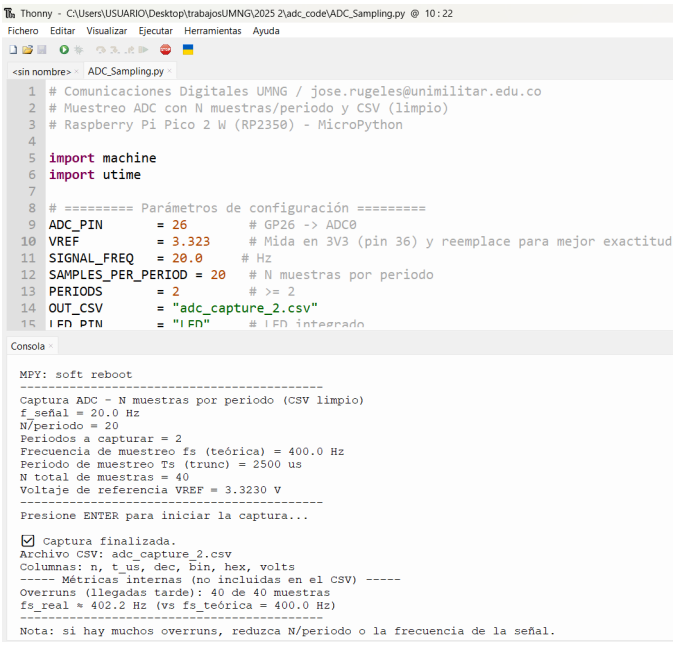


Figura 6: Codigo ADC_Samplig.py ejecutado.

El archivo generado por el codigo ADC_Sampling.py tiene el nombre de adc_capture_2.csv. Despues configuramos la señal seno en el osciloscopio y conectamos la señal al microcontrolador

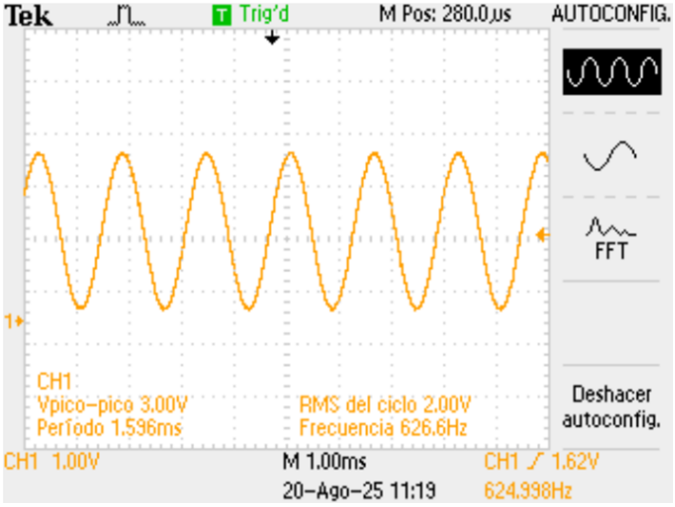


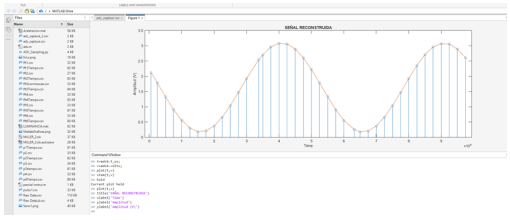
Figura 7: Señal seno en el osciloscopio.

Continuando con la guía, analizamos los datos del .csv generado por el código anterior en MATLAB, con estos datos reconstruimos la señal seno generada por el código adc.m, para esto se usaron los siguientes comandos en MATLAB:

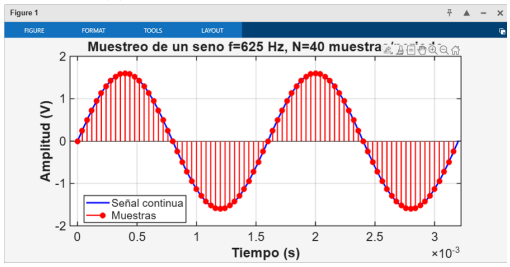
```
t=adc.t_us;
v=adc.volt;
plot(t,v)
stem(t,v)
hold
plot(t,v)
title('SENAL RECONSTRUIDA')
xlabel('Time')
ylabel('Amplitud (V)')
```

Listing 2: Comandos MATLAB.

La señal reconstruida se puede visualizar en la Figura 8.a, esta se ve de manera invertida a comparación de la gráfica generada por el código adc.m(Figura 8.b), esto se debe a que se le agrego un DC = 1.6V, esto DC se agrega debido a que la raspberry no puede medir voltajes negativos.



(a) Señal senoidal reconstruida.



(b) Señal original.

Figura 8: Comparativa de las gráficas de muestreo.

III-E. Parte C: Análisis Estadístico

Finalmente, para cuantificar la precisión y el ruido del ADC, se aplicó un voltaje DC constante y se tomaron 10,000 muestras a alta velocidad.
Seleccione 5 valores diferentes en un rango entre 0 y 3.2 voltios. Póngalos en la Cuadro 1.

Cuadro I: Tabla de resultados.

Test	V in (DC)	Media	Desviación estándar	Nombre de archivos(.txt)
1	0.8	0.68580 V	0.00178 V	mu08 y hi08
2	1.5	1.28589 V	0.00303 V	mu15 y hi15
3	1.9	1.62892 V	0.00396 V	mu19 y hi19
4	2.0	1.71464 V	0.00477 V	mu20 y hi20
5	2.7	2.31384 V	0.00569 V	mu27 y hi27

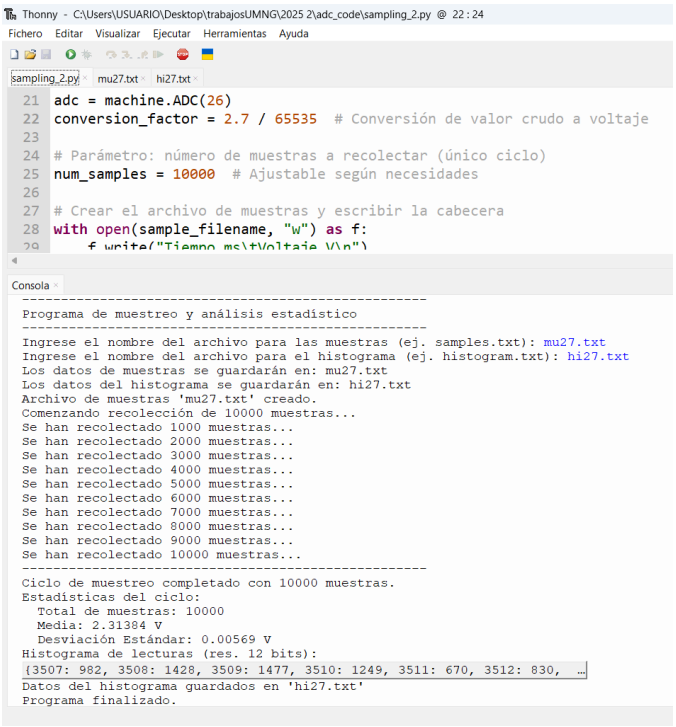


Figura 9: Salida de la consola del script de análisis estadístico.

Los resultados (mostrados en la Figura 9) fueron:

- **Muestras totales:** 10000
- **Media:** 2.31384 V
- **Desviación Estándar:** 0.00569 V

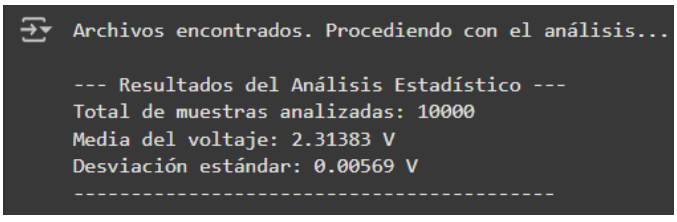


Figura 10: Salida de la consola del análisis estadístico con V = 2.7V.

Se puede observar en la Figura 10 que los resultados son iguales a los obtenidos por el código entregado por el profesor, los demás resultados se encuentran en el cuadro 1.

También se generó un histograma de las lecturas crudas de 12 bits, mostrando la distribución de los valores medidos. Se observa que la mayoría de las lecturas se concentran en unos pocos valores discretos alrededor de la media, lo cual es el comportamiento esperado.
En el Listing 3, que se encuentra en el apéndice esta el código con el cual se calculo la desviación estándar y la media junto con las respectivas gráficas.

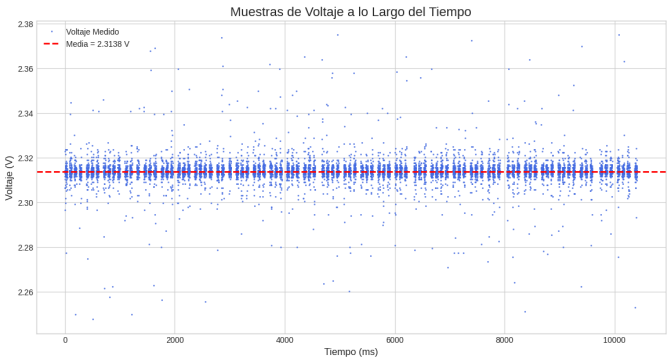


Figura 11: Gráfica de muestras de voltaje con 2.7V.

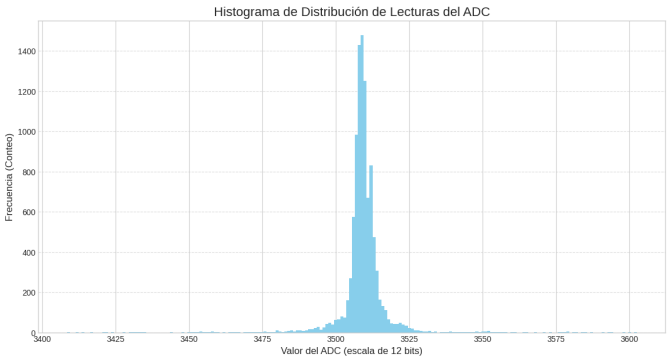


Figura 12: Gráfica del historigrama pertenenciente al voltaje 2.7.

Estas dos gráficas muestran una medición de voltaje DC estable. En la Figura 11(tiempo), se ve cómo el voltaje se mantuvo firme, siempre alrededor de un valor medio de 2.3138 V el cual es la media que se pedía calcular, sin cambios bruscos durante todo el muestreo. Eso mismo se confirma en la Figura 12(histograma), donde los datos forman una curva con forma de campana casi perfecta (distribución Gaussiana), lo que indica que las variaciones fueron mínimas y muy regulares.

IV. Análisis de Resultados

Parte A: La conversión de ‘raw16‘ a ‘code12‘ mediante el corrimiento de 4 bits a la derecha fue fundamental. Sin este paso, los cálculos de voltaje serían incorrectos por un factor de 16. Los experimentos confirmaron que el ADC se comporta de manera lineal y que ajustar el ‘VREF‘ en el código al valor real medido mejora significativamente la exactitud de la conversión.

Parte B:

La reconstrucción de la señal senoidal fue de gran importancia para estudiar el **Teorema de Muestreo de Nyquist-Shannon**. Se analizaron escenarios con una frecuencia de muestreo teórica (f_s) de 1000 Hz.

Frecuencia de Señal (20 Hz): Se realizó una prueba con una frecuencia de señal de 20 Hz, manteniendo la frecuencia de muestreo de 1000 Hz. Al verificar la condición de Nyquist:

$$1000 \text{ Hz} \geq 2 \times 20 \text{ Hz} \quad (1000 \text{ Hz} \geq 40 \text{ Hz}) \tag{7}$$

La condición se cumple, la reconstrucción de la señal de 20 Hz fue exitosa. Los puntos de muestreo fueron suficientemente densos para capturar la forma y periodicidad correctas de la onda senoidal.

En conclusión, la Parte B validó experimentalmente que el cumplimiento del teorema de Nyquist no es opcional, sino un requisito fundamental para la correcta digitalización y reconstrucción de una señal analógica.

Parte C: El análisis estadístico proporciona una medida cuantitativa del rendimiento del ADC. La media (2.31384 V) es nuestra mejor estimación del voltaje DC real aplicado. La desviación estándar (0.00569 V) nos informa sobre el ruido y la estabilidad de las lecturas. Este valor representa la dispersión típica de una medida alrededor de la media. Para muchas aplicaciones, esta precisión es más que suficiente, pero en sistemas de alta sensibilidad, este ruido podría ser un factor a considerar.

V. Conclusiones

A través de esta serie de experimentos, se logró una comprensión práctica y profunda del funcionamiento del conversor A/D en el microcontrolador Raspberry Pi Pico.

Se verificó con éxito el método para convertir los valores de 16 bits de ‘read_u16()‘ a los valores reales de 12 bits del ADC, un paso crucial para obtener mediciones de voltaje precisas.

El muestreo de señales dinámicas mostró que la Pico tiene la capacidad de digitalizar bien las formas de onda, pero también dejó en evidencia algunos retos prácticos: por ejemplo, respetar el teorema de Nyquist y manejar bien los tiempos para evitar que se acumulen más datos de los que puede procesar (los famosos overruns).

En resumen, el análisis estadístico permitió entender con claridad la precisión y el nivel de ruido del ADC, mostrando que es una herramienta bastante confiable para distintas aplicaciones, ya que su ruido es predecible y medible. Más allá de la teoría, este laboratorio dio la oportunidad de vivir la práctica y enfrentarse a problemas reales que aparecen al conectar el mundo analógico con el digital.

Referencias

[1] Raspberry Pi Foundation, Raspberry Pi Pico Datasheet,"Jan. 2021. [Online]. Available: https://datasheets.raspberrypi.com/pico/pico-datasheet.pdf

[2] J. G. Proakis and M. Salehi, *Digital Communications,* 5th ed., McGraw-Hill, 2008.

[3] A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing,* 3rd ed., Pearson, 2010.

[4] Raspberry Pi Foundation, RP2040 Datasheet,"May 2022. [Online]. Available: https://datasheets.raspberrypi.com/rp2040/rp2040-datasheet.pdf

[5] The MicroPython Team, .ºfficial MicroPython Documentation,"2023. [Online]. Available: https://docs.micropython.org/

[6] Raspberry Pi Foundation, *Get Started with MicroPython on Raspberry Pi Pico,* 2nd ed., 2021. [Online]. Available: https://rptl.io/pico-get-started

Apéndice
Códigos de Programación

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import os
# Importamos 'os' para verificar que los archivos existan

# Por favor, aseg rate de haber subido los archivos 'hi27.txt' y 'mu27.txt' a la
# carpeta '/content/' usando el panel de archivos a la izquierda antes de
# ejecutar esta celda.

# Definimos las rutas de los archivos
ruta_muestras = '/content/mu27.txt'
ruta_histograma = '/content/hi27.txt'

# Verificamos si los archivos existen en la ruta especificada
if not os.path.exists(ruta_muestras) or not os.path.exists(ruta_histograma):
    print("Error: No se encontraron los archivos 'M1.txt' o 'H1.txt' en /content/")
    print("Por favor, s belos usando el panel de archivos a la izquierda y vuelve a ejecutar.")
else:
    print("Archivos encontrados. Procediendo con el an lisis...")

    try:
        # Leemos el archivo con las muestras de voltaje
        df_voltajes = pd.read_csv(ruta_muestras, sep='\t')

        # Extraemos la columna de voltajes
        voltajes = df_voltajes['Voltaje_V']

        # Calculamos la media y la desviaci n est ndar
        media = np.mean(voltajes)
        desviacion_estandar = np.std(voltajes)

        print("\n--- Resultados del An lisis Estad stico ---")
        print(f"Total de muestras analizadas: {len(voltajes)}")
        print(f"Media del voltaje: {media:.5f} V")
        print(f"Desviaci n est ndar: {desviacion_estandar:.5f} V")
        print("-----------------------------------------\n")

    except Exception as e:
        print(f"Ocurri  un error al procesar M1.txt: {e}")


    # --- Grafica 1: Voltaje vs. Tiempo ---
    try:
        plt.style.use('seaborn-v0_8-whitegrid')
        fig, ax = plt.subplots(figsize=(14, 7))

        ax.plot(df_voltajes['Tiempo_ms'], df_voltajes['Voltaje_V'], label='Voltaje Medido', color='royalblue', alpha=0.8, marker='.', linestyle='None', markersize=2)
        ax.axhline(y=media, color='red', linestyle='--', linewidth=2, label=f'Media = {media:.4f} V')

        ax.set_title('Muestras de Voltaje a lo Largo del Tiempo', fontsize=16)
        ax.set_xlabel('Tiempo (ms)', fontsize=12)
        ax.set_ylabel('Voltaje (V)', fontsize=12)
        ax.legend()
        ax.grid(True)
        plt.show()

    except Exception as e:
        print(f"No se pudo generar la gr fica de Voltaje vs. Tiempo. Error: {e}")


    # --- Grafica 2: Histograma de Lecturas del ADC ---
    try:
        df_histograma = pd.read_csv(ruta_histograma, sep='\t')
        fig, ax = plt.subplots(figsize=(14, 7))
        ax.bar(df_histograma['Valor_ADC_12bits'], df_histograma['Frecuencia'], color='skyblue', width=1.0)

        ax.set_title('Histograma de Distribuci n de Lecturas del ADC', fontsize=16)
        ax.set_xlabel('Valor del ADC (escala de 12 bits)', fontsize=12)
        ax.set_ylabel('Frecuencia (Conteo)', fontsize=12)
        ax.grid(axis='y', linestyle='--', alpha=0.7)
        plt.show()

    except Exception as e:
        print(f"No se pudo generar el histograma. Error: {e}")
```

Listing 3: Codigo Python para calcular desviación estándar y media.