Informe IV –Exploración de conceptos de conversión A/D con Raspberry Pi Pico RS232

Dylan Ferney Vasquez Rojas
1401597
Comunicaciones Digitales

***Resumen*— La presente práctica de laboratorio tuvo como propósito analizar la comunicación serial bajo el estándar RS232, empleando una Raspberry Pi Pico 2W y el lenguaje MicroPython. Se estudió la relación entre tiempo de bit y tasa de transmisión, así como la estructura de tramas, identificando bits de inicio, datos, paridad y parada. Mediante el uso de un osciloscopio digital se realizaron mediciones con diferentes configuraciones de baudios y paridad, comparando resultados teóricos y experimentales. Adicionalmente, se implementó la interconexión de dos dispositivos para comunicación Half-Duplex y Full-Duplex, verificando su funcionamiento y registrando datos recibidos. Finalmente, se efectuó una comparación técnica entre los modelos Raspberry Pi Pico W y Pico 2W.**

***Abstract*— This laboratory focused on the application of analog-to-digital conversion (ADC) using the Raspberry Pi Pico. Voltage signals were acquired and processed through MicroPython functions to analyze the relation between 12-bit resolution and 16-bit readings. Signal sampling and data reconstruction were performed with Python and MATLAB, and a statistical analysis of DC inputs validated the accuracy of measurements. The practice reinforced the role of quantization and sampling in digital communication systems..**

I. Introducción

En el marco de las comunicaciones digitales, la conversión de señales analógicas a digitales constituye un proceso fundamental para garantizar la transmisión, almacenamiento y procesamiento eficiente de la información. El uso de microcontroladores modernos, como la Raspberry Pi Pico, permite implementar de manera práctica estos conceptos mediante su conversor analógico-digital (ADC), el cual posibilita la cuantificación y digitalización de señales continuas.

El presente laboratorio tuvo como propósito explorar los principios de la conversión A/D, el muestreo de señales y el análisis estadístico de los datos adquiridos. Para ello, se realizaron ejercicios prácticos que incluyeron la conexión de señales de voltaje variable, la visualización de resultados en tiempo real a través de software de apoyo (Thonny, MATLAB y Python), así como el procesamiento de archivos generados por el microcontrolador.

De esta manera, se busca no solo afianzar los conocimientos teóricos sobre cuantización, resolución y muestreo, sino también comprender su aplicación en sistemas de comunicación digital, donde la correcta interpretación de las señales digitalizadas es esencial para asegurar la fidelidad y eficiencia en la transmisión de la información.

DESARROLLO DE LA PRÁCTICA

En esta primera parte de la práctica se realizó la conexión de un voltaje variable utilizando una fuente de tensión DC dentro del rango permitido por el microcontrolador (0 a 3.3 V). Con el fin de garantizar una mayor precisión en los cálculos, se midió previamente el voltaje de referencia fijo de la Raspberry Pi Pico, obteniendo un valor de 3.287 V. Posteriormente, se empleó el pin ADC0 como entrada principal del conversor analógico-digital (ADC), lo que permitió adquirir los valores digitales correspondientes al voltaje aplicado. Estos datos fueron procesados mediante la función read_u16() en MicroPython, analizando su relación con la resolución real de 12 bits del conversor.

Se implementó un programa en MicroPython para realizar la conversión analógica-digital a través del pin ADC0 de la Raspberry Pi Pico. El código permitió adquirir valores digitales mediante la función read_u16(), que entrega un dato de 16 bits alineado a la izquierda. Para obtener el valor real de 12 bits, se aplicó un corrimiento de 4 bits hacia la derecha.

Posteriormente, el valor digital fue convertido a voltaje aplicando la ecuación:

$$V = \frac{code12 \times V_{REF}}{4095}$$

Donde VREF corresponde al voltaje de referencia medido en la tarjeta, en este caso 3.287 V, lo que permitió una mayor exactitud en la conversión.

El código empleado fue el siguiente:

```
import machine
import utime

ADC_PIN = 26        # GP26 -> ADC0
VREF = 3.300        # Vref ajustado (se reemplaza por 3.287 medido en la práctica)
PERIOD = 0.05       # Segundos entre lecturas (0.05–0.5)

adc = machine.ADC(ADC_PIN)

while True:
    raw16 = adc.read_u16()      # Valor de 16 bits (alineado a
```

```
la izquierda)
    code12 = raw16 >> 4          # Conversión a 12 bits reales
(0–4095)
    volts = (code12 * VREF) / 4095
    print(f"{volts:.4f}")        # Muestra un valor en volts
  utime.sleep(PERIOD)
```

Con este procedimiento se visualizaron las mediciones en tiempo real y se verificó la correcta correspondencia entre el voltaje aplicado y los valores digitalizados obtenidos.

Se aplicó un voltaje de 2.7 V a la entrada del conversor A/D de la Raspberry Pi Pico mediante la fuente DC. Al ejecutar el programa, se obtuvo el valor digital correspondiente a través de la función read_u16().

La función read_u16() entrega un número entero de 16 bits, en el rango 0–65535, debido a que el resultado de 12 bits del ADC se alinea a la izquierda rellenando con ceros los 4 bits menos significativos. Por esta razón, los valores que se observan directamente en raw16 no representan el código real del conversor. Para obtener el valor correcto, es necesario desplazar el dato 4 bits a la derecha (raw16 >> 4), recuperando así el valor en 12 bits (0–4095).

Finalmente, al convertir dicho valor digital a voltaje usando la relación:

$$V = \frac{code12 \times V_{REF}}{4095}$$

Donde VREF=3.287V, se comprobó que el voltaje reconstruido correspondía aproximadamente al valor aplicado (0.5 V), validando el funcionamiento del proceso de cuantización y la utilidad de la función read_u16() en la adquisición de señales analógicas.

Posteriormente, se activó la herramienta Plotter en el entorno de programación Thonny, lo que permitió visualizar en tiempo real el comportamiento del voltaje digitalizado por el conversor A/D de la Raspberry Pi Pico. Para lograr una gráfica más estable y legible, se ajustó el parámetro PERIOD del programa a 0.5 segundos, lo que definió el intervalo de muestreo entre cada lectura del ADC.

De esta manera, se pudo observar en la ventana del Plotter una señal continua correspondiente al voltaje aplicado (2.7 V en la primera prueba). La visualización permitió comprobar que el sistema registraba correctamente el valor medido y que los cambios en el voltaje de entrada se reflejaban en tiempo real en la gráfica.

Ilustración. 1. Period de 0.5

Luego la lectura obtenida con adc.read_u16() devuelve un valor entero de 16 bits sin signo, es decir, en el rango de 0 a 65535. Para convertir este valor a una resolución de 12 bits (rango de 0 a 4095), se realiza un desplazamiento de 4 bits a la derecha:

$$code12 = \frac{raw16}{2^4} = raw16 \gg 4$$

Esto equivale a descartar los 4 bits menos significativos, manteniendo los 12 bits más representativos.

Caso	Valor raw16	Valor (12 bits)	code12	Voltaje Calculado (V)
1	32776	2048		1.6439
2	16836	1052		0.8444
3	65535	4095		3.2870

```
import machine
import utime

ADC_PIN = 26
VREF = 3.287
PERIOD = 1

adc = machine.ADC(ADC_PIN)

while True:
    raw16 = adc.read_u16()
    code12 = raw16 >> 4
    volts = (code12 * VREF)/4095
    print(f"{volts:.4f}")
    utime.sleep(PERIOD)
    print(f"{code12:.4f}")
    print(f"{raw16:.4f}")
    print()
```

```
Shell
1.6439
2048.0000
32776.0000
```

Ilustración. 2. Código caso 1

Sustituyendo la velocidad de transmisión de 9600 baudios, se obtiene:

```
<untitled> *
1  import machine
2  import utime
3
4  ADC_PIN = 26
5  VREF = 3.287
6  PERIOD = 1
7
8  adc = machine.ADC(ADC_PIN)
9
10 while True:
11     raw16 = adc.read_u16()
12     code12 = raw16 >> 4
13     volts = (code12 * VREF)/4095
14     print(f"{volts:.4f}")
15     utime.sleep(PERIOD)
16     print(f"{code12:.4f}")
17     print(f"{raw16:.4f}")
18     print()

Shell
0.8444
1052.0000
16836.0000
```

Ilustración. 3. Código caso 2

```
<untitled> *
1  import machine
2  import utime
3
4  ADC_PIN = 26
5  VREF = 3.287
6  PERIOD = 1
7
8  adc = machine.ADC(ADC_PIN)
9
10 while True:
11     raw16 = adc.read_u16()
12     code12 = raw16 >> 4
13     volts = (code12 * VREF)/4095
14     print(f"{volts:.4f}")
15     utime.sleep(PERIOD)
16     print(f"{code12:.4f}")
17     print(f"{raw16:.4f}")
18     print()

Shell
3.2870
4095.0000
65535.0000
```

Ilustración. 4. Código caso 3

En esta etapa se ejecutó el programa adc.m suministrado por el profesor, con el fin de analizar el proceso de muestreo y la reconstrucción de una señal periódica. Para la simulación, se configuró una señal de pulso con frecuencia de 1 kHz, amplitud de ±1.5 V y ciclo de trabajo del 80%, mientras que la frecuencia de muestreo fue fijada en 100 kHz.

Del análisis de la señal en el dominio del tiempo se verificó que, por cada período de la onda (1 ms), se obtenían aproximadamente 100 muestras, lo cual confirma la relación:

$$f_s = N \cdot f_{\text{señal}} = 100 \times 1000 = 100\ \text{kHz}$$

Este resultado es coherente con el valor de muestreo establecido en el código.

Posteriormente, se aplicó la Transformada Rápida de Fourier (FFT) para examinar el espectro de la señal. Se observaron armónicos en múltiplos de 1 kHz, y la presencia de nulos espectrales en los múltiplos de 5 kHz, fenómeno esperado debido al ciclo de trabajo de 80% (D=4/5) de la señal rectangular. La resolución en frecuencia fue de 20 Hz, suficiente para discriminar los armónicos de interés.

Finalmente, se utilizó la Transformada Inversa de Fourier (IFFT) para reconstruir la señal en el dominio del tiempo, comprobando que su forma se mantenía consistente con la onda original de entrada. Esto permitió validar experimentalmente la coherencia entre la tasa de muestreo, el análisis espectral y la reconstrucción de la señal digitalizada.

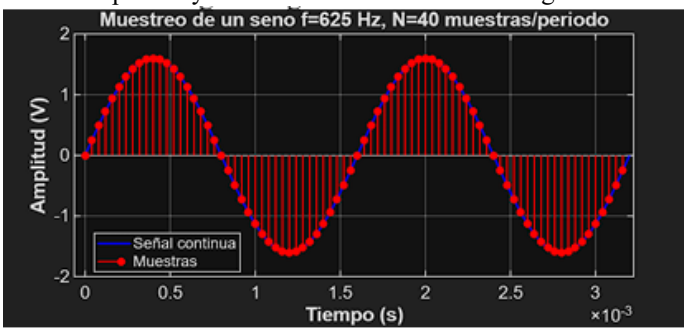

Ilustración. 5. Grafica obtenida a partir del código.

Al ejecutar el programa ADC_Sampling.py, se verificó que los datos adquiridos se almacenan automáticamente en el archivo adc_capture_2.csv. Dicho archivo contiene la numeración de cada muestra, el tiempo relativo en microsegundos, el valor digital en diferentes representaciones (decimal, binario y hexadecimal) y la conversión a voltaje real en función de la referencia medida en la tarjeta.

Antes de realizar la captura con la Raspberry Pi Pico, se ajustó en el generador de señales una onda senoidal de 3 Vpp con un nivel de DC = 1.6 V, garantizando que la señal permaneciera dentro del rango permitido por el conversor A/D (0 a 3.3 V).

La señal fue verificada con un osciloscopio Tektronix TDS 2012C, como se muestra en la Figura 1. En la pantalla se observan los cursores:

Cursor 1 en aproximadamente 3.20 V, correspondiente al valor máximo de la señal.

Cursor 2 en aproximadamente 80 mV, correspondiente al valor mínimo.

La diferencia de voltaje medida (ΔV=3.12 V) coincide con la amplitud de la señal generada, confirmando que cumple con las condiciones establecidas para la práctica. Con esta verificación se aseguró que la entrada no excediera el límite máximo del ADC del microcontrolador, evitando daños y garantizando mediciones correctas.

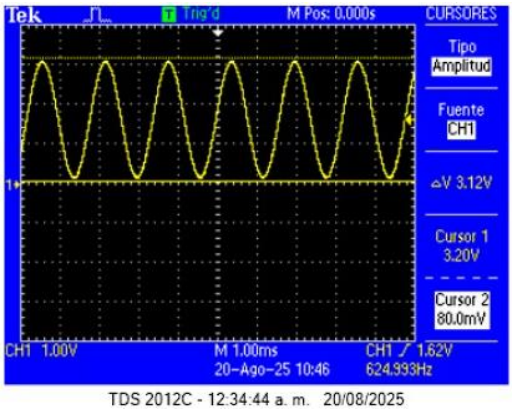

Ilustración. 6. Grafica obtenida a partir del microcontrolador.

Se cargaron los archivos adc_capture_3.csv, adc_capture_4.csv y adc_capture_5.csv, correspondientes a señales senoidales de 50 Hz, 25 Hz y 20 Hz, respectivamente. Cada archivo contiene los valores muestreados por el ADC de la Raspberry Pi Pico en formato digital (12 bits) y su conversión a voltaje real.

Posteriormente, se procesaron los datos en MATLAB/Python, graficando la señal en función del tiempo. En la Figura 2 se observan las formas de onda adquiridas. Se aprecia que las señales mantienen la amplitud y el desfase esperado, cumpliendo con las condiciones establecidas en el generador de señales.

Al ajustar los parámetros del programa adc.m, fue posible comparar la señal original con su reconstrucción a partir de las muestras digitalizadas. Los resultados confirmaron la coherencia entre la señal de entrada y la forma reconstruida, validando la correcta aplicación del muestreo.

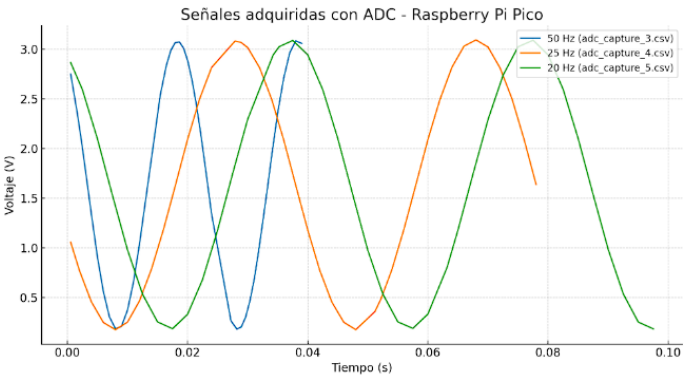

Ilustración. 7. Grafica comparando las diferentes reconstrucciones de csv

Se utilizó una fuente DC del laboratorio conectada al ADC de la Raspberry Pi, configurando cinco voltajes distintos. Para cada uno de ellos se ejecutó el programa sampling_2.py proporcionado por el docente. Dicho código generó dos archivos: uno con los valores de voltaje en función del número de muestras y otro con el histograma correspondiente. Los resultados incluían el total de muestras, la media, la desviación estándar y la ubicación de los datos almacenados.

El procedimiento se repitió para los cinco niveles de voltaje, obteniéndose la Tabla 1, donde se observan los valores medios y las desviaciones estándar junto con los nombres de los archivos generados.

Prueba	V in (DC)	Media	Desviacion Estandar	Name .txt
1	0.8	0.86379	0,23576	“0,8.txt” “h0,8.txt”
2	1.4	1.47668	0,24125	"1,4.txt” “h1,4.txt”
3	1.6	1.69987	0,23936	"1,6.txt” “h1,6.txt”
4	2	2.14733	0,23315	"2.txt” “h2.txt”
5	2.6	2.74544	0,23420	"2,6.txt” “h2,6.txt”

Tabla 1. Datos del programa

Posteriormente, se procesaron los datos con un código en MATLAB que leyó el archivo de cada voltaje, extrajo la columna correspondiente a los valores medidos y calculó la media y la desviación estándar mediante las funciones mean y std. Para el caso de 0,8 V, los resultados confirmaron los obtenidos en el programa en MicroPython, y este análisis se repitió para los demás voltajes. La Tabla 2 presenta los valores experimentales de media y desviación estándar.

Test	V in (DC)	Media experimental	Desviación estándar experimental
1	0.8	0,9672	0,2510
2	1,4	1.5133	0.2535
3	1,6	1.7835	0.2540
4	2	2.1460	0.2488
5	2,6	2.7839	0.2503

Tabla 2. Datos obtenidos a partir del programa creado.

A partir de allí, se compararon las medias calculadas con MATLAB frente a las arrojadas por la Raspberry Pi, calculando el porcentaje de error. Los resultados se recopilan en la Tabla 3, donde se aprecia que el error en la media fue mayor en el voltaje más bajo (11% para 0,8 V), mientras que para niveles más altos se redujo a valores entre 0,06% y 2,4%. Una comparación similar se hizo para la desviación estándar (Tabla 4), mostrando errores entre el 5% y el 7%.

Prueba	V in (DC)	Media	Media Experimental	% de Error
1	0.8	0.86379	0.9672	11%
2	1.4	1.47668	1.5133	2.40%
3	1.6	1.69987	1.7835	4.90%
4	2	2.14733	2.146	0.06%
5	2.6	2.74544	2.7839	1.40%

Tabla 3. Porcentaje de error con respecto a la media.

Prueba	V in (DC)	Desviacion Estandar	Desviacion Estandar Experimental	% de error
1	0.8	0,23576	0.251	6.40%
2	1.4	0,24125	0.2535	5%
3	1.6	0,23936	0.254	6%
4	2	0,23315	0.2488	6.70%
5	2.6	0,23420	0.2503	6.80%

Tabla 4. Porcentaje de error con respecto a la desviación estándar.

Luego, se graficaron histogramas para cada voltaje, analizando la distribución de los datos digitalizados por el ADC. En todos los casos, los valores se concentraron alrededor de la media, aunque con cierta dispersión atribuida principalmente al ruido eléctrico y a la inestabilidad de la fuente de alimentación. Dicha dispersión se refleja en las desviaciones estándar, que si bien son bajas, evidencian un margen de error constante del 5–7%.

Finalmente, se observó que los errores en las medias son más notorios en los voltajes bajos debido a la resolución limitada del ADC y a la precisión de la fuente. A medida que el voltaje aumenta, el error disminuye, validando el comportamiento esperado del convertidor. En conclusión, los histogramas muestran que el ADC realiza su función de cuantización, pero la exactitud de los resultados depende de la estabilidad de la fuente, la resolución del conversor y la cantidad de muestras adquiridas.

- 0.8V

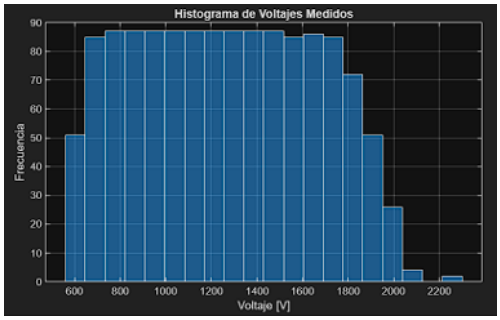

Ilustración. 7. Histograma 0,8V.

- 1.4 V

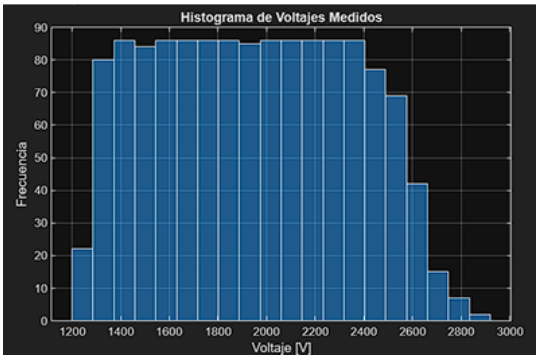

Ilustración. 8. Histograma 1.4V.

- 1.6V

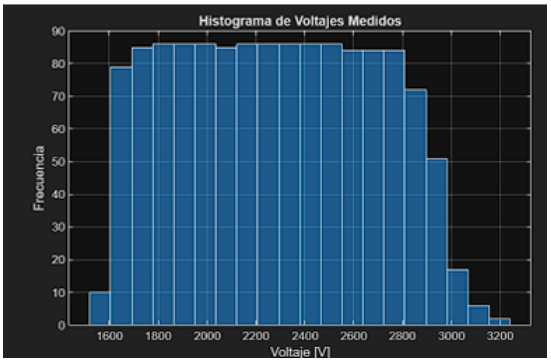

Ilustración. 9. Histograma 1.6V.

- 2V

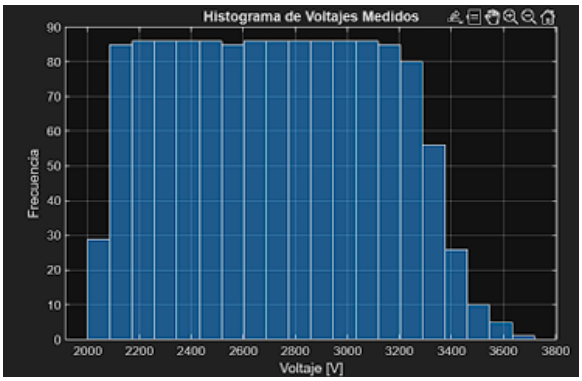

Ilustración. 10. Histograma 2V.

- 2.60V

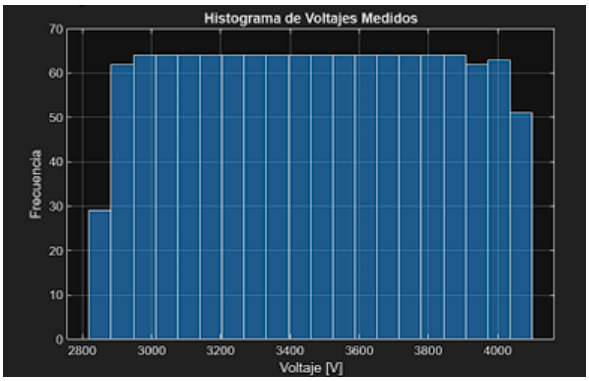

Ilustración. 10. Histograma 2.6V.

CONCLUSIONES

[1] La estructura de la trama RS232 (bit de inicio, bits de datos, bit de paridad y bits de parada) se identificó claramente en las capturas de osciloscopio, validando la correcta generación de las secuencias binarias según el carácter transmitido y la configuración de paridad seleccionada.

[2] Los histogramas y los cálculos de desviación estándar evidencian que las mediciones presentan una dispersión relativamente constante (5–7%), atribuida al ruido eléctrico y a la resolución limitada de 12 bits del ADC. Esto confirma que, aunque el conversor es funcional para fines académicos, su exactitud no es comparable con equipos de mayor precisión.

[3] La conversión analógico–digital es un proceso fundamental en los sistemas de comunicación digital, pues permite representar señales continuas en forma discreta y cuantificada, garantizando que la información pueda ser procesada y transmitida eficientemente por dispositivos digitales.

[4] El uso de herramientas como Python/MicroPython y Matlab facilita el análisis, almacenamiento y procesamiento de los datos muestreados, permitiendo validar experimentalmente conceptos como la media, la desviación estándar, el muestreo y la reconstrucción de señales.

[5] El análisis estadístico de las muestras evidencia la importancia de comprender la resolución y limitaciones del conversor A/D, ya que la precisión de las mediciones y la fidelidad de la reconstrucción de la señal dependen directamente de la correcta interpretación de los valores entregados por el ADC.

REFERENCIAS

[1] MicroPython, machine — functions related to the hardware, Documentación oficial, [En línea]. Disponible en: https://docs.micropython.org/en/latest/library/machine.html. [Accedido: 14-ago-2025].

[2] Raspberry Pi Foundation, Getting started with MicroPython on Raspberry Pi Pico, Documentación oficial, [En línea]. Disponible en: https://www.raspberrypi.com/documentation/microcontrollers/micropython.html. [Accedido: 14-ago-2025].

[3] W. Stallings, Data and Computer Communications, 10th ed. Boston, MA, USA: Pearson, 2014.

[4] ANSI/TIA, TIA-232-F: Interface Between Data Terminal Equipment and Data Circuit-Terminating Equipment Employing Serial Binary Data Interchange, Telecommunications Industry Association, 1997.

[5] A. S. Tanenbaum and D. Wetherall, Computer Networks, 5th ed. Boston, MA, USA: Pearson, 2011.