UNIVERSIDAD MILITAR NUEVA GRANADA
https://www.umng.edu.co/

Laboratorio #5 Jitter

Daza Torres Brayan David
est.brayan.ddaza@unimilitar.edu.co
1401551
GitHub: Brayan-Daza21

***Resumen*— El laboratorio aborda la configuración y análisis de señales analógicas con el conversor A/D del Raspberry Pi Pico. Se estudia el muestreo de señales senoidales, el uso de la FFT con distintos tamaños, y el efecto de la frecuencia de entrada. También se propone un programa alternativo para muestreo estable, integrando el análisis de Jitter y explorando alternativas de muestreo en la Raspberry Pi Pico 2W.**

Palabras Clave*— *Muestreo, Conversor A/D, Raspberry Pi Pico 2W, FFT, Jitter, Ventana Hanning.

Abstract— The laboratory focuses on the configuration and analysis of analog signals using the Raspberry Pi Pico A/D converter. It examines sinusoidal signal sampling, FFT with different sizes, and the impact of input frequency. An alternative program for stable sampling is proposed, including jitter analysis and exploring sampling alternatives with the Raspberry Pi Pico 2W.

Keywords*— *Sampling, A/D Converter, Raspberry Pi Pico 2W, FFT, Jitter, Hanning Window.

I. INTRODUCCIÓN

Este laboratorio busca familiarizar al estudiante con las etapas de adquisición, muestreo y análisis espectral de señales mediante el uso del microcontrolador Raspberry Pi Pico. Inicialmente, se trabaja con el programa sugerido (ADC_testing.py) para realizar muestreo de una señal senoidal, analizar su representación en el dominio del tiempo y la frecuencia, y evaluar cómo influyen parámetros como el número de puntos en la FFT y la frecuencia de entrada. Posteriormente, se plantea el desarrollo de un programa propio que permita muestrear con estabilidad y analizar el impacto del Jitter. Con este enfoque, el laboratorio integra teoría y práctica para fortalecer la comprensión de los procesos de digitalización de señales en sistemas.

II. OBJETIVOS

- Configurar y muestrear señales analógicas en el Raspberry Pi Pico 2W.
- Analizar el comportamiento de la señal digitalizada mediante la FFT con diferentes resoluciones.
- Evaluar el efecto de la frecuencia de entrada en el proceso de muestreo.
- Implementar un programa propio de muestreo estable e incluir el análisis de Jitter.
- Explorar alternativas de muestreo en el dispositivo Raspberry Pi Pico 2W.

III. ASPECTOS TEÓRICOS

La conversión de señales analógicas a digitales (A/D) es un proceso fundamental en los sistemas de comunicaciones modernas, pues permite representar una señal continua en forma discreta para su procesamiento digital. Este proceso depende de parámetros clave como la frecuencia de muestreo, la resolución del conversor y el uso de técnicas de procesamiento digital de señales (DSP) [1].

El muestreo se basa en el teorema de Nyquist-Shannon, el cual establece que la frecuencia de muestreo debe ser al menos el doble de la frecuencia máxima de la señal para evitar aliasing [2]. Una vez muestreada la señal, es posible analizar su contenido espectral mediante la Transformada Rápida de Fourier (FFT), herramienta que permite estudiar la distribución de energía en el dominio de la frecuencia [3].

En el proceso de adquisición, es importante considerar el efecto del Jitter, definido como la variación temporal en los instantes de muestreo. Este fenómeno introduce distorsiones y errores en la reconstrucción de la señal, afectando directamente la calidad de la codificación de la fuente [4].

El uso de ventanas de análisis, como la ventana de Hanning, es esencial para reducir las fugas espectrales en el análisis con FFT, mejorando la resolución en frecuencia [5].

IV. PROCEDIMIENTOS Y ANALISIS

PARTE 1

Primero se pasó a generar una señal Senosoidal de 200 Hz, 1.2 Vpp y un DC de 1.6 voltios. Posteriormente se visualiza en el osciloscopio antes de usarla en la Raspberry Pi Pico 2W.

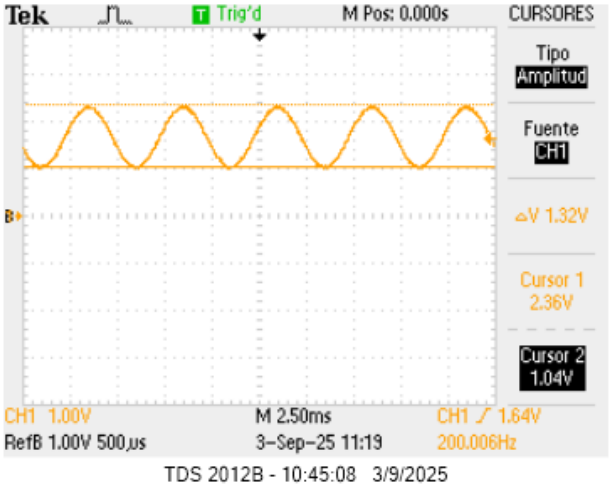

Figura 1. Señal Seno de 200Hz, 1.2 Vpp y DC 1.6

Se ejecuta el programa ADC_testing.py en el cual se toman los archivos .txt muestras y fft el cual muestra la siguiente información al ejecutarse:

```
MPY: soft reboot
Iniciando adquisición y análisis...
Frecuencia deseada: 2000 Hz, frecuencia real: 1903.71 Hz
Offset DC removido: 0.731 V
Frecuencia dominante: 200.78 Hz
Amplitud señal: 0.003 V
Piso de ruido: 0.00009 V (-91.31 dB FS)
SNR: 30.80 dB, ENOB: 4.82 bits
```

Figura 2. Información al ejecutar el codigo

En la figura 2 se notan datos como la frecuencia de muestreo utilizada en el programa, entre otros datos importantes como el SNR.

En un primer momento se obtiene los dos archivos muestras.txt y fft.txt los cuales se trabajan y por medio de codigo se grafican en MATLAB.

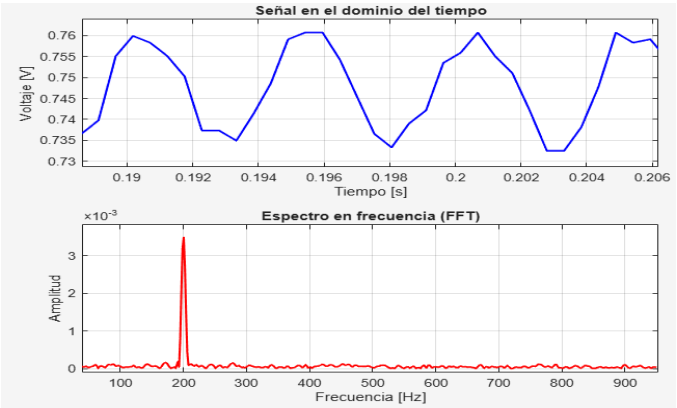

Figura 3. Gráficas en MATLAB

Se aprecia en la Figura 3 como se grafica en el dominio del tiempo la señal seno que muestreo el programa del docente y la FFT que se muestra parecido a un pulso en la frecuencia de 200 Hz la cual es la que se configuro en el generador de señales en un primer momento.

Siguiendo con la guía de laboratorio se pide que en el programa se modifique el número de puntos de la FFT (64,128,256,512,1024,2048) y en el generador de señales escogimos las frecuencias a usar de 300 Hz, 900 Hz y 1800 Hz.
Usando la función de subplot para poder visualizar las señales muestreadas:

Frecuencia de 300 Hz:

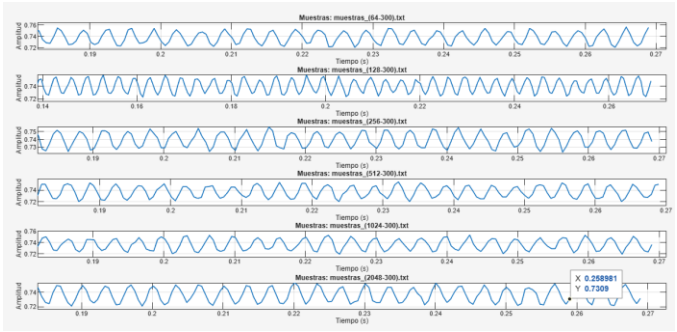

Figura 4. En el dominio del tiempo 300 Hz

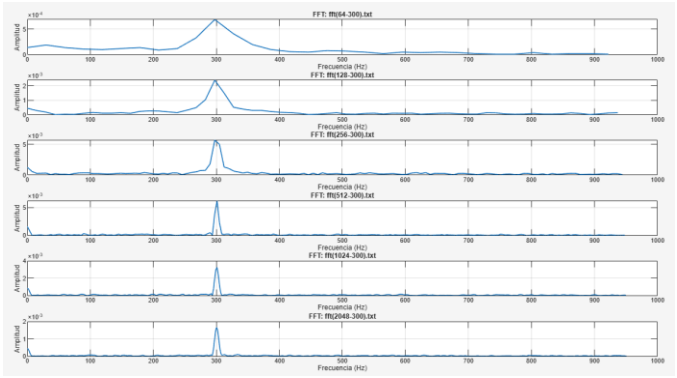

Figura 5. FFT 300 Hz

Empezando con la de 900 Hz se aprecia que en la figura 4 al cambiar los puntos de la FFT no se nota un cambio visible pero cuando se analiza la Figura 5 que corresponde a la FFT se nota que empezando con unos puntos de 64 la frecuencia en 300 Hz que es la frecuencia fundamental de la señal seno usada no se nota muy bien al tener un ancho considerable y a medida que se va aumentando los puntos de FFT hasta llegar a 2048 se nota un dato mas preciso y mas exacto al ser mas delgado o preciso en ese mismo punto de 300Hz.

Al aumentar los puntos de la FFT, se logra una mejor resolución en frecuencia, el pico de la señal se hace más definido, y se pueden diferenciar mejor señales cercanas en frecuencia.

Frecuencia de 900 Hz:

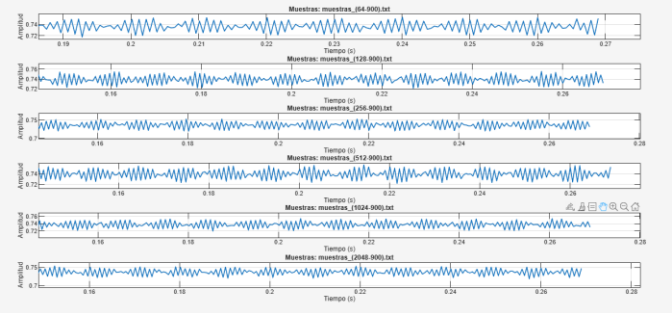

Figura 6. En el dominio del tiempo 900 Hz

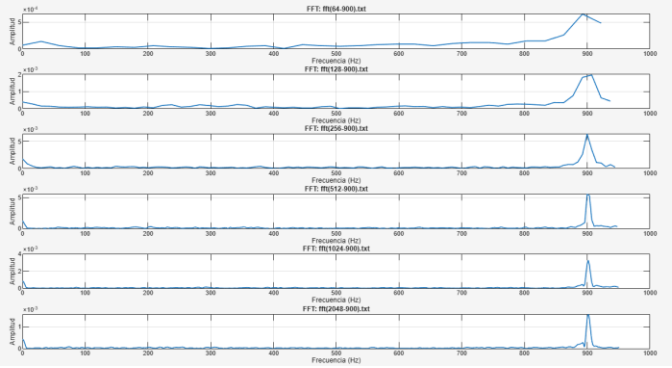

Figura 7. FFT 900 Hz

Un analisis similar ocurre en 900 Hz de acuerdo a la de 300 Hz donde el verdadero cambio cuando se va ajustando el numero de puntos de 64 a 2048 se puede observar en la Figura 7 donde a menor puntos en 64 no se ve bien la frecuencia fundamental de la señal seno usada y al aumentar se ve el pico de la señal mas definido al obtener una mayor resolución.

Frecuencia de 1800 Hz:

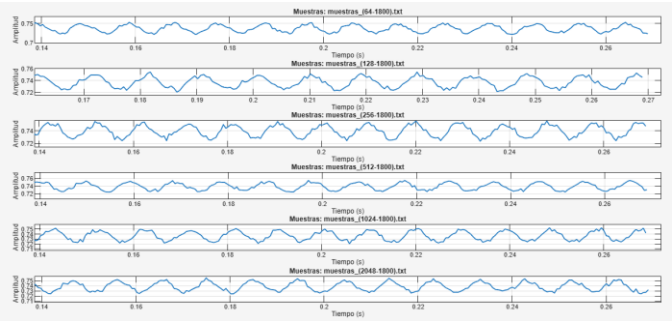

Figura 8. En el dominio del tiempo 1800 Hz

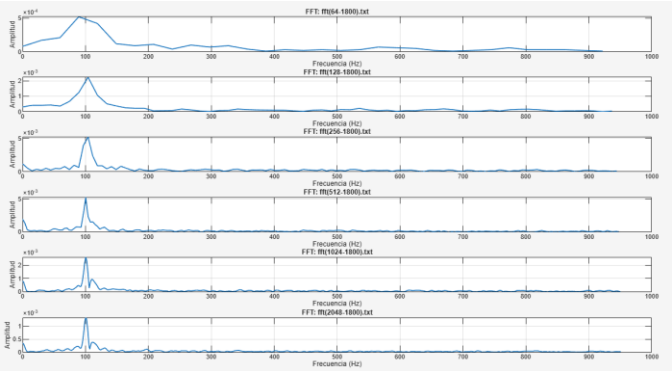

Figura 9. FFT 1800 Hz

Para la frecuencia de 1800 Hz se nota en la Figura 9 que el pico no esta en 1800 Hz, esto se debe a que en la teoria (Condicion de Nyquist), la fs (frecuencia de muestreo) debe ser dos veces la frecuencia de la señal que para 1800Hz su fs es de 3600 Hz, al no cambiar este parámetro dentro del programa del docente la captura de datos se ve como una señal aliased donde ocurrio la distorsion al no poder abarcar la frecuencia configurada en la señal seno por lo cual su pico no se encuentran en 1800 Hz al no poder abarcar ese rango.

Se nota lo mismo que en los anteriores analisis de las gráficas de FFT que a medida que se va aumentando el numero de puntos de la FFT se logra un pico mas definido al tener mayor resolución a la hora de tomar los datos y guardarlos en los archivos .txt.

¿Como se establece la tasa de muestreo en el programa?

Analizando el codigo del docente:

```
f_muestreo = 2000  # Frecuencia objetivo de muestreo (Hz)
dt_us = int(1_000_000 / f_muestreo)
```

Figura 10. Frecuencia de muestreo definida

En el programa, la tasa de muestreo se fija inicialmente con la variable f_muestreo, a partir de la cual se calcula el periodo de muestreo en microsegundos (dt_us = 1_000_000 / f_muestreo) que controla el retardo entre lecturas del ADC con utime.sleep_us ().

Sin embargo, debido a las limitaciones del temporizador y del ADC, la frecuencia de muestreo real se determina midiendo el tiempo total de adquisición y calculando:

$$fs_{real} = \frac{N}{elapsed\_time}$$

```
for i in range(N):
    t_actual = utime.ticks_diff(utime.ticks_us(), start) / 1_000_000
    tiempos.append(t_actual)
    muestras.append(adc.read_u16())
    utime.sleep_us(dt_us)

elapsed_time = tiempos[-1]
fs_real = N / elapsed_time
print(f"Frecuencia deseada: {f_muestreo} Hz, frecuencia real: {fs_real:.2f} Hz")
```

Figura 11. Calculo para determinar la frecuencia real de muestreo usada.

Y de acuerdo a ese calculo y la Figura 11 que corresponde a el codigo del docente el programa imprime el valor de frecuencia de muestreo como frecuencia real, la cual se puede mirar en la Figura 2 que sale la frecuencia real (Frecuencia real de muestreo usada).

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

La ventana Hanning atenúa los extremos de la señal, suavizando la transición en los bordes y concentrando mejor la energía en la frecuencia fundamental.

En el código se implementa en la función **apply_hanning_window(data),** donde se genera un vector de pesos:

```
def apply_hanning_window(data):
    N = len(data)
    window = [0.5 * (1 - math.cos(2 * math.pi * i / (N - 1))) for i in range(N)]
    return [d * w for d, w in zip(data, window)]
```

Figura 12. Ventana Hannig en el programa

y cada muestra como se nota en la Figura 12 se multiplica por su respectivo peso (d * w).

De esta forma, los datos ventaneados (senal_ventaneada) se pasan luego a la FFT, mejorando la resolución espectral y reduciendo artefactos en la estimación de amplitud y frecuencia.

PARTE 2

Se diseña un programa el cual se agrega a mi repositorio de GitHub como una alternativa para muestrar la misma señal de 200Hz, 1.2 Vpp y un DC de 1.6 voltios.

Se genera tambien dos archivos .txt de muestras y fft para graficarlos en MATLAB y poder hacer una comparación con el programa que dio el docente para la **PARTE 1**.

Quedando las gráficas de la siguiente manera:

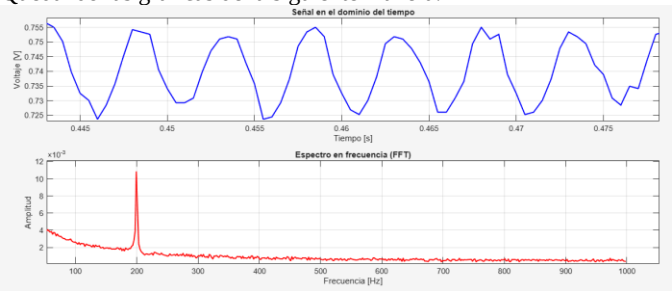

Figura 13. Gráficas en MATLAB con el codigo diseñado por los estudiantes.

Al comparar la Figura 13 con la Figura 3 se nota como una toma de datos similar respecto a los datos obtenidos para el dominio del tiempo y en la FFT se nota su pico en la frecuencia fundamental definida en el generador de señales de 200 Hz.

El programa utiliza muestreo uniforme periódico, implementado mediante un Timer en modo periódico que genera interrupciones a una frecuencia fija (fs). Esto garantiza intervalos de muestreo regulares y reduce el jitter respecto a soluciones basadas en retardos por software.

Ademas, en esta **PARTE 2**, se plantea la siguiente pregunta:

¿ Qué es el Jitter? ¿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 respecto a su posición ideal en el tiempo. En otras palabras, ocurre cuando el reloj de muestreo no es perfectamente uniforme, provocando que algunas muestras se tomen antes o después del instante correcto.

En el proceso de codificación de la fuente, el jitter afecta de la siguiente manera:

- Error en la amplitud muestreada: al no capturar la señal en el instante exacto, el valor digitalizado puede ser mayor o menor al real.

-Distorsión espectral: el jitter introduce ruido de fase, ensanchando los picos de frecuencia en el espectro.

-Degradación del SNR: una mayor variación en los tiempos de muestreo reduce la relación señal a ruido, afectando directamente la calidad de la señal reconstruida.

-Reducción de la ENOB (Effective Number of Bits): el jitter limita la precisión efectiva del ADC, aun si tiene buena resolución nominal.

En sistemas de telecomunicaciones, el jitter es especialmente crítico en señales de alta frecuencia, donde pequeños errores temporales generan desviaciones importantes en la amplitud digitalizada.

Se pidio agregar un informe del jitter en el codigo diseñado en la PARTE 2, el cual genero el siguiente reporte:

```
MPY: soft reboot
✅ Muestreo terminado.
Periodo ideal (us): 500.0
Periodo medio (us): 499.912032
Jitter RMS (us): 2.739951
✅ Reporte 'reporte.txt' guardado.
```

Figura 14. Reporte del Jitter

De acuerdo a la Figura 14 se analiza que:

El jitter RMS reportado (2.74 µs) es muy pequeño en comparación con el periodo de la señal de 200 Hz. Esto significa que el muestreo fue bastante estable y preciso, sin afectar de forma significativa la calidad de la señal capturada.

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

El Raspberry Pi Pico 2W, basado en el microcontrolador RP2040, ofrece varias alternativas para mejorar el muestreo de señales considerando sus características técnicas: **[6]**

-Uso de temporizadores internos (PWM/PIO): Se pueden configurar temporizadores o los bloques de I/O programable (PIO) para generar interrupciones periódicas estables. Esto permite muestrear con intervalos más uniformes que usando retardos por software (sleep_us()), reduciendo el jitter.

-DMA (Direct Memory Access): El RP2040 incluye soporte para DMA, lo que permite transferir automáticamente las muestras del ADC a memoria sin intervención constante del procesador. Esto mejora la estabilidad temporal y libera al CPU para análisis o transmisión.

-Buffers circulares de adquisición: Usar buffers circulares gestionados por interrupciones o DMA permite capturar bloques de datos de manera continua sin pérdidas, mejorando la coherencia en el muestreo.

-Sincronización externa: El Pico 2W puede recibir señales de reloj externas para sincronizar el muestreo del ADC con otros dispositivos, útil en sistemas de comunicaciones.

-Uso de periféricos adicionales: Si se requieren mayores resoluciones o velocidades, se pueden conectar ADC externos vía SPI o I2C, permitiendo superar las limitaciones del ADC interno (10–12 bits efectivos y ~500 kS/s). **[6]**

V. CONCLUSIONES

 El muestreo de señales en el Raspberry Pi Pico 2W permite comprender la relación entre frecuencia de entrada, resolución de la FFT y calidad espectral de la señal digitalizada.

 El aumento en el número de puntos de la FFT mejora la resolución en frecuencia, definiendo con mayor precisión los picos de la señal.

 Cuando la frecuencia de muestreo no cumple la condición de Nyquist, aparecen efectos de aliasing que distorsionan la representación espectral.

 La implementación de un programa propio basado en temporizadores periódicos reduce el Jitter y asegura un muestreo más estable.

 El Jitter medido (2.74 µs) resultó insignificante frente al periodo de la señal de 200 Hz, demostrando que el sistema fue preciso y confiable en este caso.

 El Raspberry Pi Pico 2W ofrece alternativas como temporizadores, DMA y sincronización externa que permiten optimizar la adquisición de señales y reducir errores de muestreo.

VI. REFERENCIAS

[1] A. V. Oppenheim and A. S. Willsky, *Signals and Systems*, 2nd ed. Prentice Hall, 1997.

[2] C. E. Shannon, "Communication in the presence of noise," *Proceedings of the IRE*, vol. 37, no. 1, pp. 10–21, Jan. 1949.

[3] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Mathematics of Computation*, vol. 19, no. 90, pp. 297–301, 1965.

[4] M. Pelgrom, *Analog-to-Digital Conversion*, 3rd ed. Springer, 2017.

[5] F. J. Harris, "On the use of windows for harmonic analysis with the discrete Fourier transform," *Proceedings of the IEEE*, vol. 66, no. 1, pp. 51–83, Jan. 1978.

[6] Raspberry Pi Foundation, *Raspberry Pi Pico 2 W Datasheet*, Nov. 26, 2024.