LABORATORIO NFFT

María Alejandra Fonseca Zarrate.
Ingeniería de Telecomunicaciones
Universidad Militar Nueva Granada
Bogotá D.C.

Resumen - Este informe muestra el análisis de una señal senoidal que fue obtenida a través de un conversor analógico-digital (ADC) en una Raspberry Pi Pico. Con el fin de analizar la operación del proceso de muestreo y conversión al dominio de la frecuencia a través de la Transformada Rápida de Fourier (FFT), se producen diversas señales con frecuencias y propiedades cambiantes. Se examina cómo el número de puntos que se emplean en la FFT y el efecto de la ventana Hanning afectan la calidad del análisis espectral. Para analizar la reacción del sistema frente a distintas configuraciones, se utilizan MATLAB y Python para representar gráficamente los resultados.

Abstract - This report shows the analysis of a sinusoidal signal obtained through an analog-to-digital converter (ADC) on a Raspberry Pi Pico. In order to analyze the operation of the sampling and conversion process to the frequency domain through the Fast Fourier Transform (FFT), various signals with varying frequencies and properties are produced. The paper examines how the number of points used in the FFT and the effect of the Hanning window affect the quality of the spectral analysis. To analyze the system's response to different configurations, MATLAB and Python are used to graphically represent the results.

I. OBJETIVOS

- El objetivo principal de esta práctica de laboratorio es comprender cómo la variación del número de puntos de la FFT y la frecuencia de la señal afectan la representación espectral. Adicionalmente, se evalúa el efecto de la ventana Hanning en la minimización de errores en la transformación espectral. Estos análisis son esenciales para aplicaciones en telecomunicaciones, procesamiento digital de señales.

II. INTRODUCCIÓN

En los sistemas de comunicaciones digitales y procesamiento de señales, la conversión analógico-digital (ADC) es un proceso fundamental que permite transformar señales del mundo real en información procesable por sistemas digitales. La calidad de esta conversión depende de varios factores, como la tasa de muestreo, la resolución del ADC y el tratamiento de la señal en el dominio de la frecuencia.

En este laboratorio, se configura un generador de señales para producir una onda senoidal con una componente DC definida. Posteriormente, la señal es adquirida por la Raspberry Pi Pico mediante su ADC y procesada por un programa en Python que permite analizar su comportamiento en el tiempo y en la frecuencia utilizando la FFT

III. MARCO TEÓRICO

1. Conversión análogo digital (ADC) y su Implementación en la Raspberry Pi Pico W P

El proceso de conversión analógico-digital (ADC) permite transformar señales continuas en representaciones digitales discretas. Este proceso es esencial en sistemas embebidos y procesamiento de señales digitales, ya que muchas señales del mundo real, como audio y mediciones de sensores, son de naturaleza analógica.

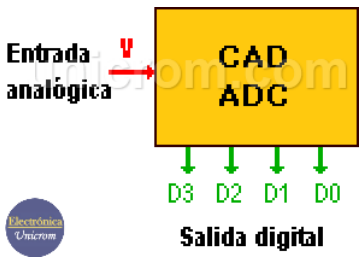
Funcionamiento del ADC

Un conversor A/D realiza tres pasos fundamentales:

Muestreo: La señal analógica se mide en intervalos regulares de tiempo.

Cuantización: Cada muestra se asigna a un valor discreto en un rango de niveles definidos por la resolución del ADC.

Codificación: El valor cuantizado se convierte en una secuencia binaria para su almacenamiento o procesamiento digital.



Img 1. Conversor A/D (tomado de https://unicrom.com/convertidor-analogico-digital-cad-adc/)

En este laboratorio, se utiliza el ADC integrado en la Raspberry Pi Pico, el cual posee una resolución de 12 bits y opera con un voltaje de referencia de 3.3V. Esto significa que cada nivel de cuantización representa un

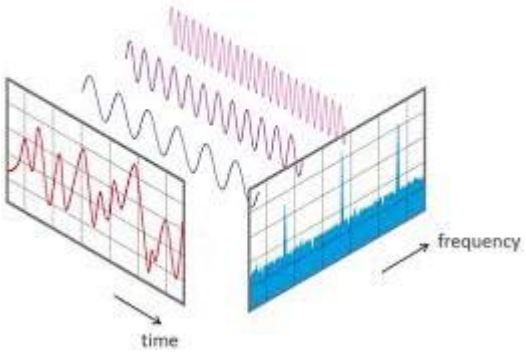
incremento de:

$$\frac{3.3V}{2^{12}} = 0.85mV \tag{1}$$

Si la señal supera este rango, se producirá saturación, generando mediciones incorrectas.

1.1 Transformada Rápida de Fourier (FFT) y Resolución Espectral

La Transformada de Fourier Discreta (DFT) permite analizar la frecuencia de una señal muestreada. Su cálculo directo es computacionalmente costoso, por lo que se usa la Transformada Rápida de Fourier (FFT), un algoritmo optimizado para reducir la complejidad computacional.



Img 2. FFT.

Efecto del número de puntos en la FFT

El número de puntos de la FTT (N_{FFT}) afecta la resolución espectral, que se define como:

$$\Delta f = \frac{f_s}{N_{FFT}} \tag{2}$$

donde f_s es la tasa de muestreo.

- Un mayor N_{FFT} proporciona una mejor resolución en frecuencia, permitiendo distinguir con mayor precisión los componentes espectrales.
- Los valores muy altos pueden aumentar el ruido espectral si la señal no es suficientemente larga o estable.

En este experimento se evalúan NFFT=(64,128,256,512,1024) para observar su impacto en la resolución de la señal.

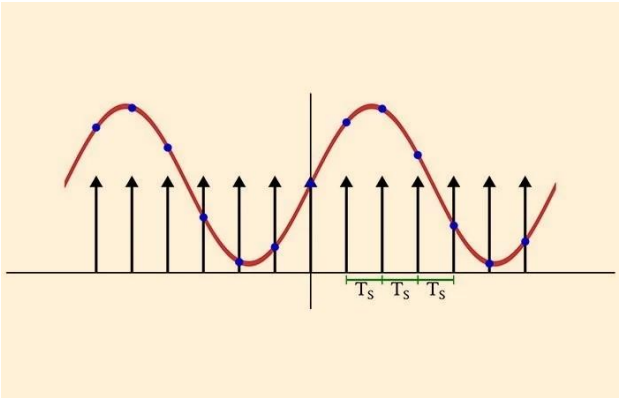
1.2. Tasa de Muestreo y Teorema de Nyquist.

Para evitar aliasing, la frecuencia de muestreo debe cumplir con el Teorema de Nyquist, que establece:

$$f_s \geq 2f_{max} \tag{3}$$

Donde f_{max} es la frecuencia más alta presente en la señal.

Si la tasa de muestreo es insuficiente, se superponen componentes de frecuencia, generando una señal distorsionada e irreconocible. En este laboratorio se estudia el comportamiento del sistema con señales de 100 Hz a 1800 Hz para verificar si la tasa de muestreo es adecuada en cada caso.



Img 3.Teorema de Nyquist (tomado de https://thonny.org/).

1.3. Ventana Hanning y uso en la FFT.

Cuando una señal muestreada no es periódica en la ventana de observación, pueden aparecer discontinuidades al aplicar la FFT. Estas discontinuidades generan lóbulos laterales en el espectro, lo que introduce errores en la interpretación de las frecuencias.

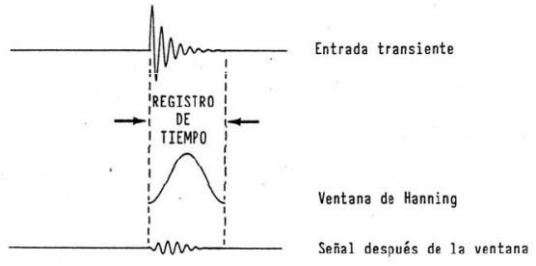
Para reducir este problema, se aplican funciones ventana, como la ventana Hanning, definida por:

$$w(n) = 0.5(\frac{2\pi n}{N-1}) \tag{4}$$

Efecto de la Ventana Hanning en el Espectro

Ventana rectangular (sin ventana): Mayor resolución espectral pero más lóbulos laterales.

Ventana Hanning: Reduce los lóbulos laterales, pero ensancha los picos espectrales.

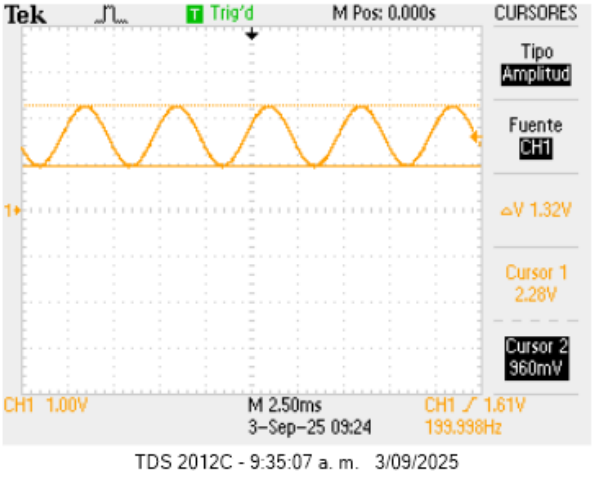


Img 3.Ventana de Hanning .

En este laboratorio, se evalúa el impacto de esta ventana en la representación espectral.

IV. DESCRIPCIÓN GENERAL DE LA PRÁCTICA.

Para empezar con esta práctica de laboratorio , el primer paso fue generar una señal senoidal desde el generador en la que se ajustó la tensión a 1.2v pico a pico junto con una componente DC de 1.6V y a una frecuencia de 200 Hz y con esto se pasó a verificar la señal en el osciloscopio antes de conectar las entradas al conversor A/D de la raspberry.



Img 4. Visualización señal generada.

Luego de esto se pasó a cargar el código entregado por el docente desde Github llamado ADC_testing.py como se logra ver en la Img. 5.

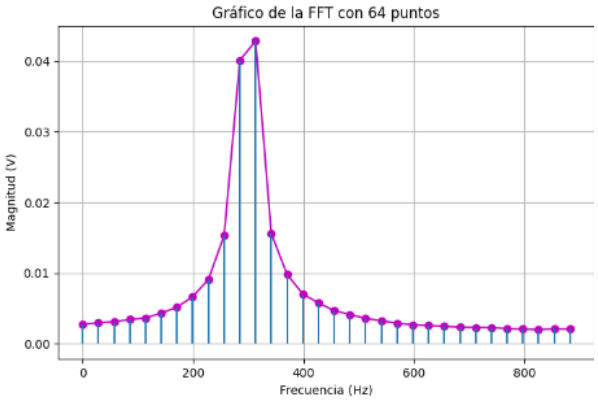
```
from machine import ADC, Pin
import utime
import array
import math
import cmath

# Configuración inicial
adc = ADC(Pin(27))
N = 512
N_FFT = 1024
f_muestreo = 2000  # Frecuencia objetivo de muestreo (Hz)
dt_us = int(1_000_000 / f_muestreo)
data = array.array('H', [0] * N)

# Adquisición de datos con tiempos uniformes
def acquire_data():
    tiempos = []
    muestras = []
    start = utime.ticks_us()

    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")

    # Guardar muestras en archivo
    with open("muestras.txt", "w") as f:
        f.write("Tiempo(s)\tVoltaje(V)\n")
        voltajes = [(x / 65535) * 3.3 for x in muestras]
        for t, v in zip(tiempos, voltajes):
            f.write(f"{t:.6f}\t{v:.5f}\n")
```

Img 5. Código ADC_testing cargado en thonny.

Por medio de este código se pasó a realizar la medición variando el valor de la cantidad de puntos a visualizar en la FFT , así mismo se guardaron los archivos generados al ejecutar el código muestras.txt y fft.txt para cada uno de los distintos valores para luego poder visualizarlos por medio de gráficas de la fft, para esto se inició con la frecuencia de 300Hz.

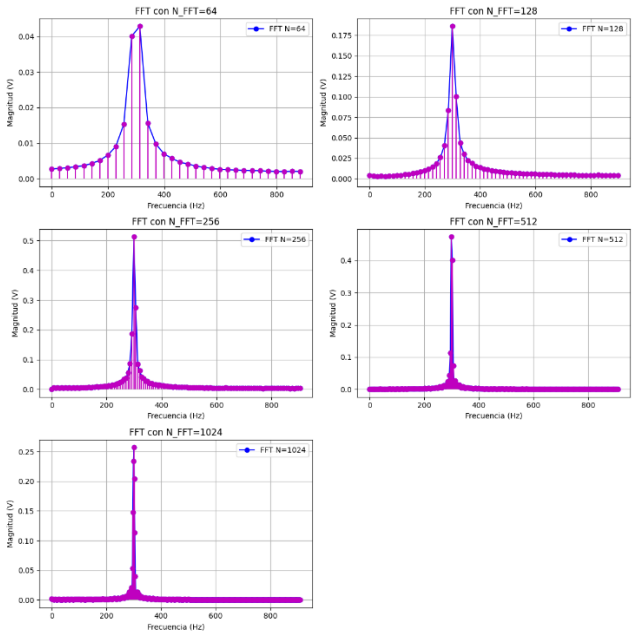
Frecuencia de 300 Hz.

Primero se realizó para un valor de N_FFT = 64.



Img 6. Señal visualizada en Python con N FFT =64.

Seguido de esto se repite el proceso para los demás valores del número de puntos de la FFT en el programa 128,256,512 y 1024, se almacenaron los archivos y luego se realizó un solo gráfico para los distintos números de puntos separados por medio de 5 subplots para poder visualizar el efecto que tiene la variación de estos puntos en la gráfica.

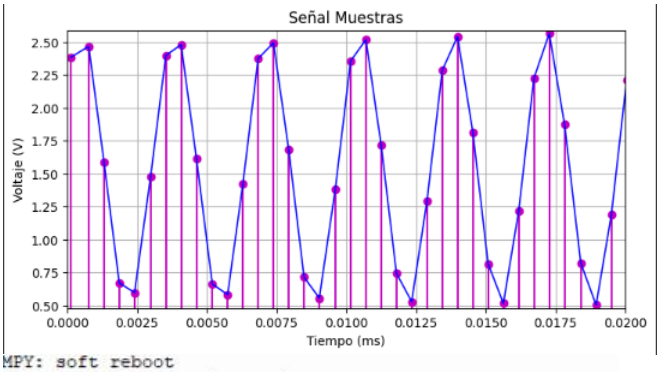


Img 7. Señal visualizada en Python para los distintos números de puntos a 300Hz.

En este caso se observa que para NFFT=64, la resolución

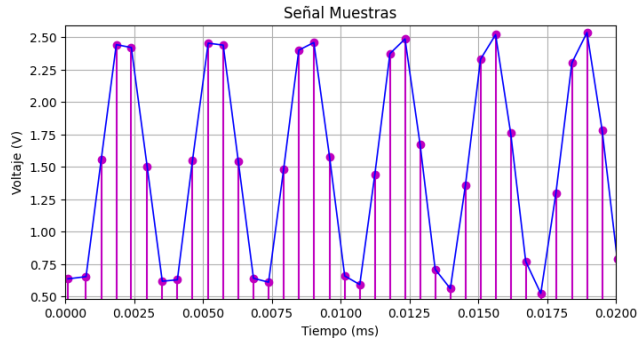
es baja y el pico en 300 Hz se ve ensanchado y con mayor dispersión y a medida que aumenta la resolución mejora como en el caso de NFFT=1024, el pico en 300 Hz es mucho más definido y con menor dispersión y a su vez que para NFFT bajos la amplitud del pico es menor debido a la menor resolución espectral y así mismo hay más dispersión de energía en varias frecuencias cercanas junto con una mayor dispersión en los lóbulos laterales que con valores mayores de NFFT.

Al ejecutar el código se obtuvieron parámetros como el piso de ruido, la relación señal-ruido (SNR) y la ENOB (número efectivo de bits) que es calculada por el mismo programa. Así que en la siguiente imagen se visualizan estos datos entregados y la señal senoidal continua y pura del archivo muestras.txt para cada uno de los distintos valores de puntos a esta frecuencia.



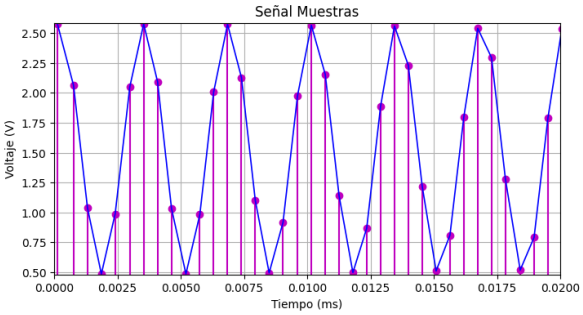

```
MPY: soft reboot
Iniciando adquisición y análisis...
Frecuencia deseada: 2000 Hz, frecuencia real: 1823.23 Hz
Offset DC removido: 1.535 V
Frecuencia dominante: 313.37 Hz
Amplitud señal: 0.043 V
Piso de ruido: 0.00574 V (-55.20 dB FS)
SNR: 17.47 dB, ENOB: 2.61 bits
```

Img 8. Señal senoidal 300 Hz 64 NFFT y datos entregados por el programa



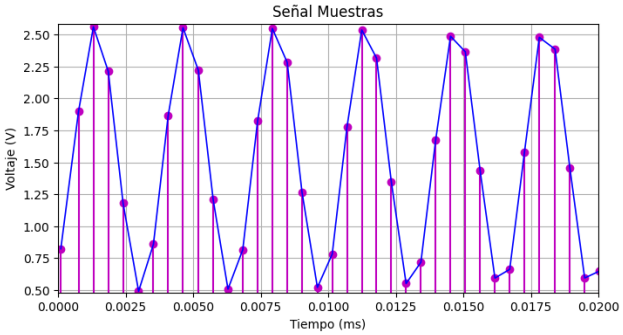

```
MPY: soft reboot
Iniciando adquisición y análisis...
Frecuencia deseada: 2000 Hz, frecuencia real: 1823.23 Hz
Offset DC removido: 1.530 V
Frecuencia dominante: 299.12 Hz
Amplitud señal: 0.186 V
Piso de ruido: 0.01158 V (-49.09 dB FS)
SNR: 24.11 dB, ENOB: 3.71 bits
```

Img 9. Señal senoidal 300 Hz 128 NFFT y datos entregados por el programa



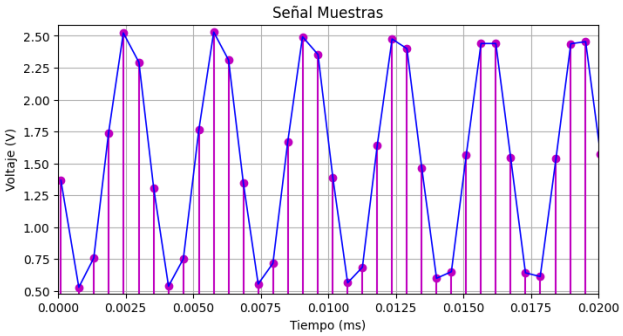

```
MPY: soft reboot
Iniciando adquisición y análisis...
Frecuencia deseada: 2000 Hz, frecuencia real: 1823.52 Hz
Offset DC removido: 1.535 V
Frecuencia dominante: 299.17 Hz
Amplitud señal: 0.513 V
Piso de ruido: 0.01417 V (-47.34 dB FS)
SNR: 31.17 dB, ENOB: 4.89 bits
```

Img 10. Señal senoidal 300 Hz 256 NFFT y datos entregados por el programa




```
MPY: soft reboot
Iniciando adquisición y análisis...
Frecuencia deseada: 2000 Hz, frecuencia real: 1823.07 Hz
Offset DC removido: 1.531 V
Frecuencia dominante: 299.10 Hz
Amplitud señal: 0.473 V
Piso de ruido: 0.00489 V (-56.59 dB FS)
SNR: 39.72 dB, ENOB: 6.30 bits
```

Img 11. Señal senoidal 300 Hz 512 NFFT y datos entregados por el programa

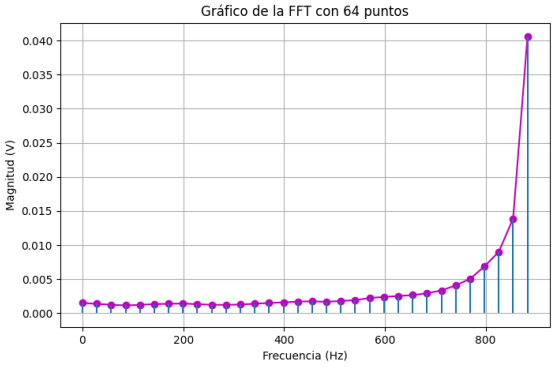



```
MPY: soft reboot
Iniciando adquisición y análisis...
Frecuencia deseada: 2000 Hz, frecuencia real: 1822.17 Hz
Offset DC removido: 1.530 V
Frecuencia dominante: 300.73 Hz
Amplitud señal: 0.257 V
Piso de ruido: 0.00295 V (-60.98 dB FS)
SNR: 38.81 dB, ENOB: 6.15 bits
```

Img 12. Señal senoidal 300 Hz 1024 NFFT y datos entregados por el programa

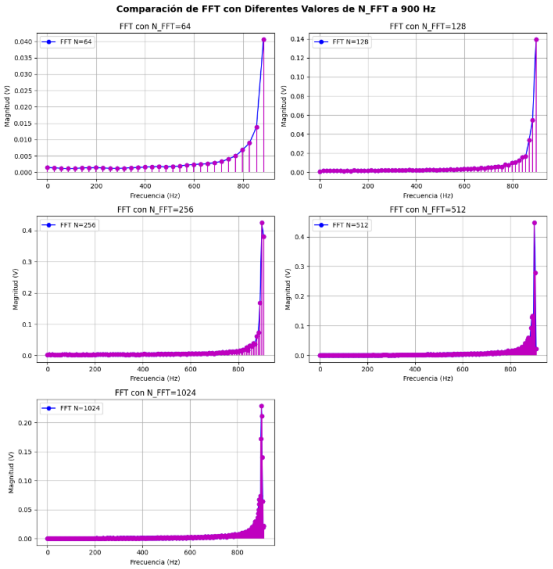
Este mismo proceso se repitió para las demás frecuencias sin embargo solo se mostrarán 2 más.

Frecuencia de 900 Hz.



Img 13. Señal visualizada en Python con 900Hz y N FFT 64.

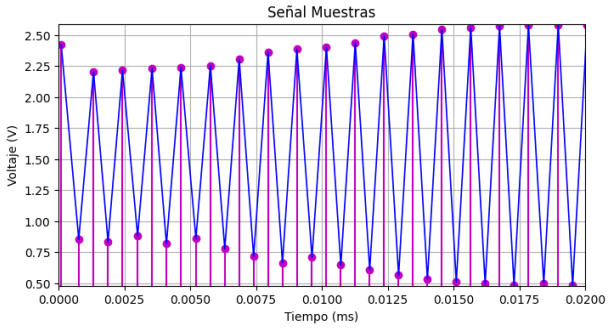
Luego se repite el proceso para los demás valores del número de puntos de la FFT en el programa 128,256,512 y 1024, se almacenaron los archivos y luego se realizó un solo gráfico para los distintos números de puntos separados por medio de 5 subplots para poder visualizar la variación de estos puntos en la gráfica.



Img 14. Señal visualizada en Python para los distintos números de puntos a 900Hz.

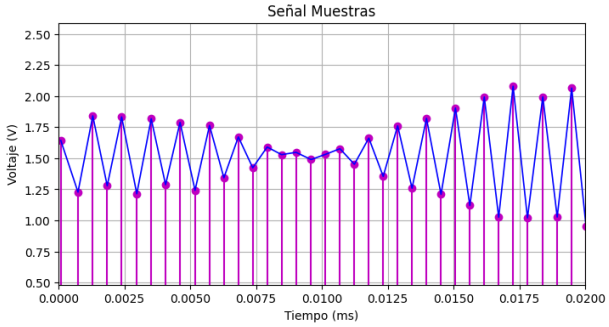
La comparación de las FFT con diferentes valores de NFFT muestra que, a medida que se incrementa el número de puntos, la resolución espectral mejora y la frecuencia dominante se identifica con mayor precisión. Con valores bajos como N=64 o N=128, el espectro aparece poco definido y la representación de la magnitud es limitada, mientras que con valores más altos como N=512 y N=1024 la forma espectral es más clara y el pico de frecuencia se concentra mejor en la componente fundamental.

Al ejecutar el código se obtuvieron parámetros como el piso de ruido, la relación señal-ruido (SNR) y la ENOB (número efectivo de bits) que es calculada por el mismo programa. Así que en la siguiente imagen se visualizan estos datos entregados y la señal senoidal continua y pura del archivo muestras.txt para cada uno de los distintos valores de puntos a esta frecuencia.



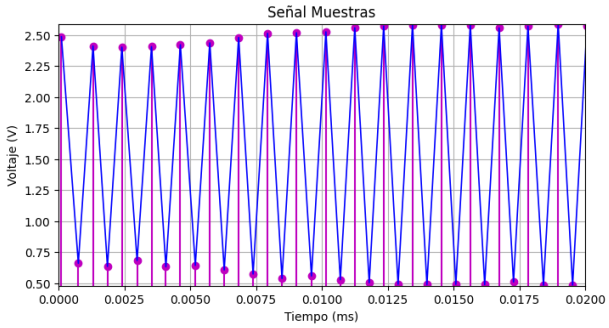

```
MPY: soft reboot
Iniciando adquisición y análisis...
Frecuencia deseada: 2000 Hz, frecuencia real: 1822.78 Hz
Offset DC removido: 1.533 V
Frecuencia dominante: 882.91 Hz
Amplitud señal: 0.041 V
Piso de ruido: 0.00269 V (-61.77 dB FS)
SNR: 23.56 dB, ENOB: 3.62 bits
```

Img 15. Señal senoidal 900 Hz 64 NFFT y datos entregados por el programa



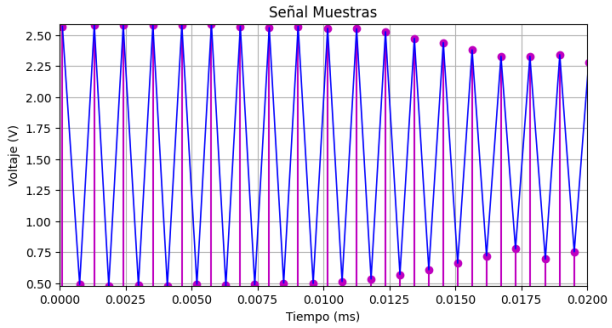

```
MPY: soft reboot
Iniciando adquisición y análisis...
Frecuencia deseada: 2000 Hz, frecuencia real: 1823.00 Hz
Offset DC removido: 1.531 V
Frecuencia dominante: 897.26 Hz
Amplitud señal: 0.139 V
Piso de ruido: 0.00494 V (-56.49 dB FS)
SNR: 29.00 dB, ENOB: 4.52 bits
```

Img 16. Señal senoidal 900 Hz 128 NFFT y datos entregados por el programa



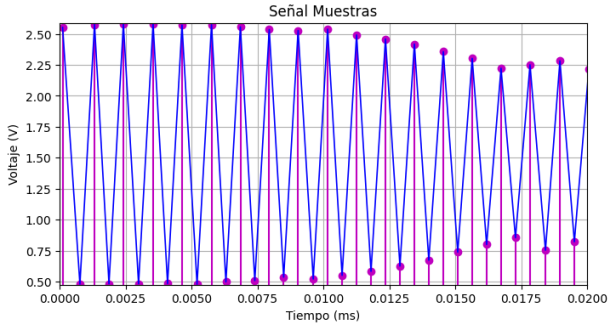

```
MPY: soft reboot
Iniciando adquisición y análisis...
Frecuencia deseada: 2000 Hz, frecuencia real: 1823.41 Hz
Offset DC removido: 1.533 V
Frecuencia dominante: 897.46 Hz
Amplitud señal: 0.425 V
Piso de ruido: 0.01145 V (-49.19 dB FS)
SNR: 31.38 dB, ENOB: 4.92 bits
```

Img 17. Señal senoidal 900 Hz 256 NFFT y datos entregados por el programa




```
MPY: soft reboot
Iniciando adquisición y análisis...
Frecuencia deseada: 2000 Hz, frecuencia real: 1822.89 Hz
Offset DC removido: 1.532 V
Frecuencia dominante: 900.77 Hz
Amplitud señal: 0.447 V
Piso de ruido: 0.00844 V (-51.84 dB FS)
SNR: 34.48 dB, ENOB: 5.44 bits
```

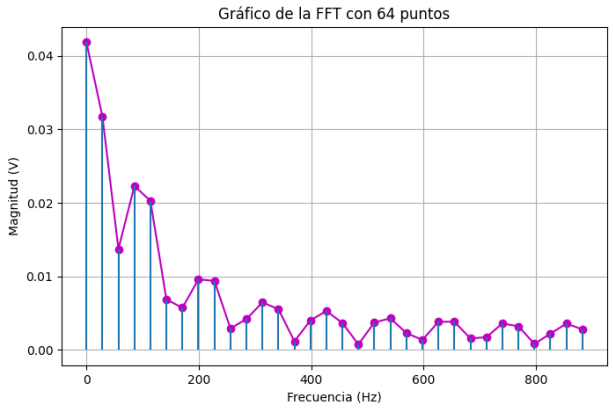
Img 18. Señal senoidal 900 Hz 512 NFFT y datos entregados por el programa




```
MPY: soft reboot
Iniciando adquisición y análisis...
Frecuencia deseada: 2000 Hz, frecuencia real: 1823.07 Hz
Offset DC removido: 1.532 V
Frecuencia dominante: 900.85 Hz
Amplitud señal: 0.229 V
Piso de ruido: 0.00459 V (-57.14 dB FS)
SNR: 33.95 dB, ENOB: 5.35 bits
```

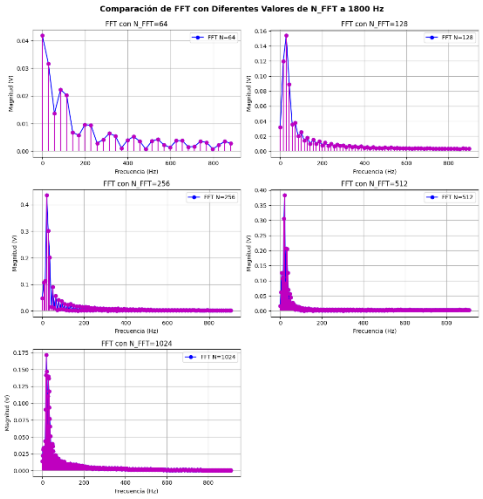
Img 19. Señal senoidal 900 Hz 1024 NFFT y datos entregados por el programa

Frecuencia de 1800 Hz.



I***mg 20. Señal visualizada en Python con 1800Hz y N FFT 64***

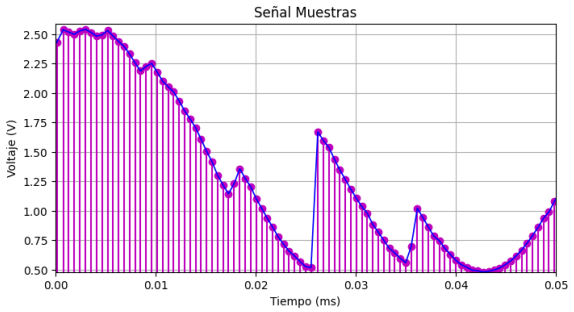
Luego se repite el proceso para los demás valores del número de puntos de la FFT en el programa 128,256,512 y 1024, se almacenaron los archivos y luego se realizó un solo gráfico para los distintos números de puntos separados por medio de 5 subplots para poder visualizar la variación de estos puntos en la gráfica.



Img 21. Señal visualizada en Python para los distintos números de puntos a 1800Hz.

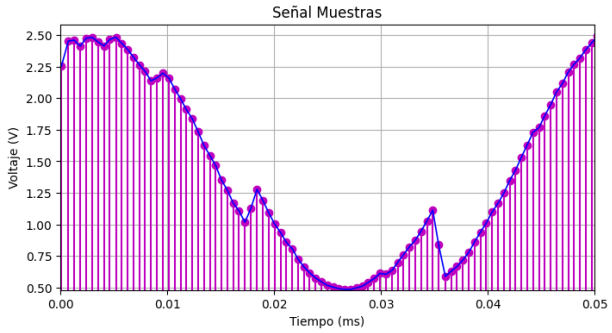
En esta comparación de la FFT con una señal de 1800 Hz, se aprecia que el valor de NFFT influye directamente en la claridad y definición del espectro. Con valores bajos como N=64 y N=128N, la resolución espectral es limitada y aparecen variaciones notables que dificultan identificar con precisión la frecuencia dominante. Al aumentar el número de puntos a N=256, N=512 y especialmente N=1024, la representación en frecuencia se vuelve mucho más detallada, reduciendo la dispersión de energía y mostrando de forma más nítida la concentración en las bajas frecuencias, lo que evidencia la mayor capacidad de análisis de la FFT con valores grandes de N, aunque a costa de un incremento en el procesamiento requerido.

Al ejecutar el código se obtuvieron parámetros como el piso de ruido, la relación señal-ruido (SNR) y la ENOB (número efectivo de bits) que es calculada por el mismo programa. Así que en la siguiente imagen se visualizan estos datos entregados y la señal senoidal continua y pura del archivo muestras.txt para cada uno de los distintos valores de puntos a esta frecuencia.



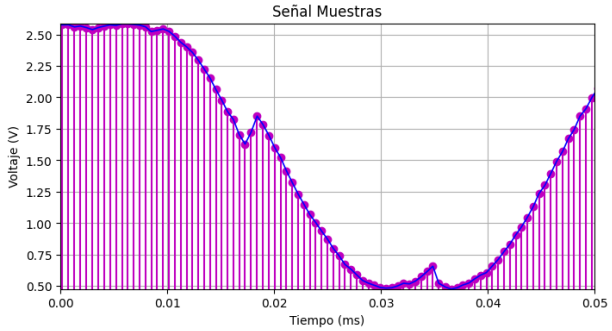

```
MPY: soft reboot
Iniciando adquisición y análisis...
Frecuencia deseada: 2000 Hz, frecuencia real: 1822.59 Hz
Offset DC removido: 1.475 V
Frecuencia dominante: 28.48 Hz
Amplitud señal: 0.032 V
Piso de ruido: 0.00652 V (-54.08 dB FS)
SNR: 13.74 dB, ENOB: 1.99 bits
```

Img 22. Señal senoidal 1800 Hz 64 NFFT y datos entregados por el programa



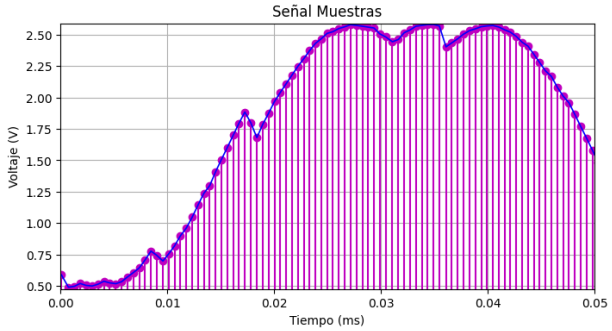

```
MPY: soft reboot
Iniciando adquisición y análisis...
Frecuencia deseada: 2000 Hz, frecuencia real: 1823.04 Hz
Offset DC removido: 1.468 V
Frecuencia dominante: 28.49 Hz
Amplitud señal: 0.154 V
Piso de ruido: 0.01090 V (-49.62 dB FS)
SNR: 23.00 dB, ENOB: 3.53 bits
```

Img 23. Señal senoidal 1800 Hz 128 NFFT y datos entregados por el programa



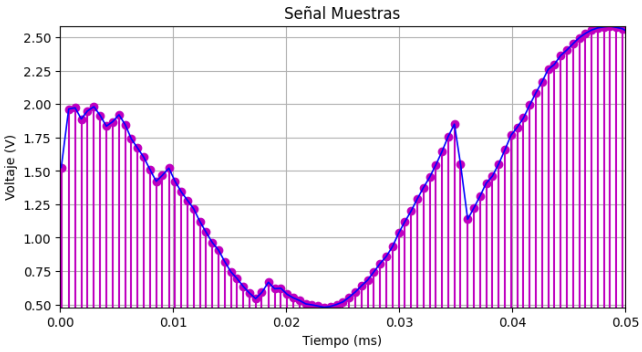

```
MPY: soft reboot
Iniciando adquisición y análisis...
Frecuencia deseada: 2000 Hz, frecuencia real: 1823.24 Hz
Offset DC removido: 1.505 V
Frecuencia dominante: 21.37 Hz
Amplitud señal: 0.435 V
Piso de ruido: 0.01262 V (-48.35 dB FS)
SNR: 30.76 dB, ENOB: 4.82 bits
```

Img 24. Señal senoidal 1800 Hz 256 NFFT y datos entregados por el programa




```
MPY: soft reboot
Iniciando adquisición y análisis...
Frecuencia deseada: 2000 Hz, frecuencia real: 1822.84 Hz
Offset DC removido: 1.605 V
Frecuencia dominante: 21.36 Hz
Amplitud señal: 0.383 V
Piso de ruido: 0.00992 V (-50.44 dB FS)
SNR: 31.74 dB, ENOB: 4.98 bits
```

Img 25. Señal senoidal 1800 Hz 512 NFFT y datos entregados por el programa




```
MPY: soft reboot
Iniciando adquisición y análisis...
Frecuencia deseada: 2000 Hz, frecuencia real: 1822.91 Hz
Offset DC removido: 1.489 V
Frecuencia dominante: 19.58 Hz
Amplitud señal: 0.172 V
Piso de ruido: 0.00670 V (-53.85 dB FS)
SNR: 28.17 dB, ENOB: 4.39 bits
```

Img 26. Señal senoidal 1800 Hz 1024 NFFT y datos entregados por el programa

Después de haber visualizado las gráficas y los datos que entrega el programa pasamos a tabular los datos obtenidos.

Frecuencia Real (Hz)	64	128	256	512	1024
300	313,37	299,12	299,17	299,1	300,73
900	882,91	897,26	897,46	900,77	900,85
1800	28,49	28,48	21,37	21,36	19,58

Tabla 1. Frecuencia real vs frecuencia dada por el programa con distintos números de puntos FFT.

Se observa que a medida que el número de puntos de la FFT (NFFT) aumenta, la frecuencia estimada tiende a estar más cercana a la frecuencia real.

En la frecuencia más alta se logra ver la desviación mucho más alta, esto debido a que la frecuencia ya está muy cercana a la frecuencia de muestreo y hay un efecto de aliasing por esta razón la medición de estas frecuencias es de un valor muy distinto al teórico ya que la frecuencia de muestreo no es suficiente para capturar la señal.

Frecuencia Deseada	64	128	256	512	1024
300	1823,23	1823,23	1823,52	1823,07	1822,17
900	1822,78	1823	1823,41	1822,89	1823,07
1800	1822,59	1823,04	1823,24	1822,84	1822,91

Tabla 2. Frecuencia de muestreo deseada vs frecuencia de muestreo dada por el programa con distintos números de puntos FFT.

La frecuencia de muestreo configurada en el código es 2000 Hz, pero los valores obtenidos están alrededor de 1823 Hz en todos los casos. Con esto se concluye que el numero de puntos no afecta este parámetro.

Amplitud Señal (V)	64	128	256	512	1024
300	0,043	0,186	0,513	0,473	0,257
900	0,041	0,139	0,425	0,447	0,229
1800	0,032	0,154	0,435	0,383	0,172

Tabla 3. Amplitud de señal calculada por el programa con distintos NFFT

Con estos resultados podemos decir que el número de puntos de la FFT afecta directamente la estimación de la amplitud de la señal, ya que con pocos puntos indica una baja resolución espectral y posibles pérdidas de energía en el análisis de frecuencia.

Piso de ruido (dB)	64	128	256	512	1024
300	-55,2	-49,09	-47,34	-56,59	-60,98
900	-61,77	-56,49	-49,19	-51,84	-57,14
1800	-54,08	-49,62	-48,35	-50,44	-53,85

Tabla 4. Piso de ruido en voltios calculado por el programa.

Se puede decir que el número de puntos de la FFT influye directamente en el nivel de ruido detectado ya que con FFT de pocos puntos se presenta un mayor piso de ruido en el caso de 64,128 y 256, es decir que la elección de este valor N debe tener un balance entre la resolución espectral y el nivel de ruido.

SNR (dB)	64	128	256	512	1024
300	17,47	24,11	31,17	39,72	38,81
900	23,56	29	31,38	34,48	33,95
1800	13,74	23	30,76	31,74	28,17

Tabla 5. SNR en dB calculado por el programa.

A medida que se aumenta el número de puntos se obtiene una mejor relación de señal a ruido esto se ve más que todo en la frecuencia baja ya que en las altas el beneficio de aumentar la cantidad de puntos es menor.

ENOB (bits)	64	128	256	512	1024
300	2,61	3,71	4,89	6,3	6,15
900	3,62	4,52	4,92	5,44	5,35
1800	1,99	3,53	4,82	4,98	4,39

Tabla 6. ENOB bits calculados por el programa

El ENOB es un indicador de la precisión del conversor A/D relacionado con la calidad de la señal digitalizada, así que se puede decir que el aumento en cantidad de puntos de FFT mejora la medición del ENOB en las frecuencias bajas o que no superan el cálculo teórico para la frecuencia de muestreo manejada en este caso.

Seguido se realizo el análisis del código para entender como se esta estableciendo la tasa de muestreo.

Inicialmente la tasa de muestreo se define como variable en el código f_muestreo, la cual se inicializa con un valor de 2000 Hz con este valor el programa realiza el cálculo del tiempo de muestreo establecido en la variable dt_us en el cual se realiza el siguiente cálculo:

$$dt\_us = \frac{1\_000\_000}{f\_muestreo} = 500\mu s \tag{5}$$

Con esto ya sabemos que el programa toma una muestra cada 500 us, esto en la funcion de acquire_data() y haciendo uso de una funcion utime.sleep que relaciona directamente la variable dt_us esto para que el intervalo de muestreo sea de 500 us,

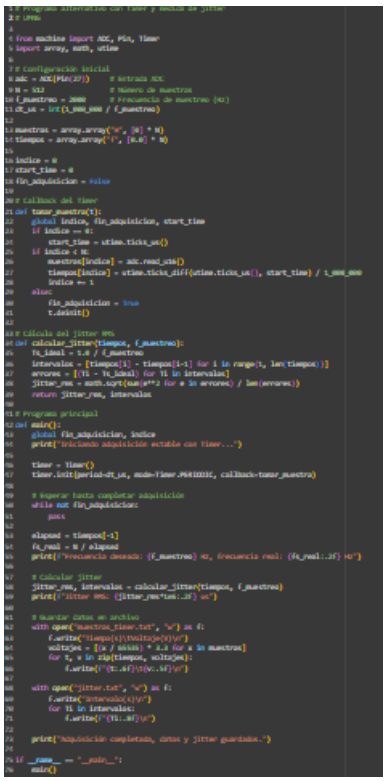
Este programa también calcula la frecuencia de muestreo real por medio de la variable fs_real esto lo hace por medio de esta operación en definida en el código:

$$fs\_real = \frac{N}{elapsedtime} \tag{6}$$

En este caso N es el número total de muestras tomadas sobre el tiempo total que se demoró en tomar estas muestras.

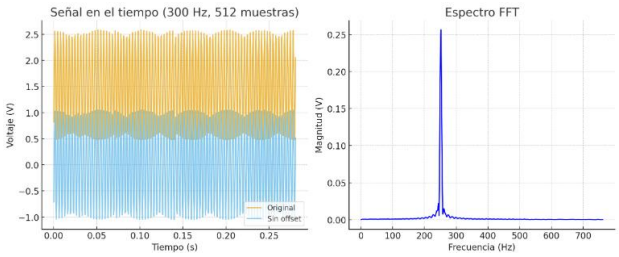
Como segunda parte de este laboratorio se creo con ayuda de herramientas de IA la creación de un código similar al (ADC_testing.py) que cumpliera con los requisitos que eran: muestrear una señal senoidal de 200 Hz, 1.2Vpp, y una componente DC de 1.6V con un tiempo de muestreo estable adicional de añadirle un jitter.

Por lo anterior se obtuvo el siguiente código:



Img 27. Codigo creado en base a (ADC_testing.py)

Este codigo usa un Timer periódico de hardware en lugar de sleep_us(), lo que permite un muestreo mucho más estable y reduce significativamente el jitter, es decir, las variaciones indeseadas en el tiempo entre muestras. Además, incluye el cálculo del jitter RMS, lo que permite cuantificar de manera objetiva la precisión temporal del sistema, También emplea buffers preasignados en lugar de listas dinámicas, haciendo la adquisición más eficiente y con menor sobrecarga. En conjunto, estas mejoras garantizan intervalos de muestreo más uniformes.



Img 28. Graficas con datos tomados con el nuevo código.

¿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 el instante en que se toman las muestras de una señal. En un sistema ideal, todas las muestras deberían capturarse con intervalos de tiempo perfectamente uniformes (por ejemplo, cada 500 µs si la frecuencia de muestreo es 2 kHz). Sin embargo, en la práctica, el reloj de muestreo puede presentar pequeñas fluctuaciones debidas a ruido, inestabilidad del oscilador, o limitaciones del hardware, y eso hace que las muestras no se tomen exactamente en los instantes esperados.

- El jitter provoca que cada muestra no represente exactamente el valor real de la señal en el instante deseado, lo cual se traduce en errores de cuantificación adicionales.
- A frecuencias altas o señales con variaciones rápidas, el jitter introduce distorsión y ruido de fase, degradando la fidelidad de la señal reconstruida.
- En el dominio de la frecuencia, puede aumentar el piso de ruido, reduciendo la relación señal/ruido (SNR) y por tanto el número efectivo de bits (ENOB) del conversor.
- En comunicaciones digitales, un exceso de jitter puede ocasionar que el receptor interprete incorrectamente los símbolos, lo cual implica errores de decodificación

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

En la Raspberry Pi Pico W, el muestreo de señales analógicas puede realizarse de distintas formas según las necesidades de estabilidad, velocidad y precisión. Para señales lentas, como una senoidal de 200 Hz, se puede usar el temporizador en MicroPython, logrando un muestreo sencillo y suficiente, aunque con cierto jitter. Si se requiere mayor precisión o frecuencias de muestreo más altas, el uso de ADC con DMA o el subsistema PIO en C/C++ permite reducir significativamente el jitter y descargar a la CPU. Otra alternativa es incorporar un ADC externo con reloj propio, lo cual mejora la resolución, el SNR y la estabilidad temporal. Además, se recomienda complementar con filtrado anti-alias analógico, un buffer de entrada de baja impedancia y una referencia de voltaje estable. Estas opciones permiten equilibrar complejidad, costo y calidad de muestreo según el objetivo del diseño (Raspberry Pi, 2023).

V. ANÁLISIS DE RESULTADOS Y CONCLUSIONES.

Se observó que a medida que aumenta NFFT, la resolución espectral mejora, permitiendo distinguir con mayor precisión las componentes la señal.

Al aplicar la ventana de Hanning se redujo la amplitud de los lóbulos laterales del espectro de la señal, pero así mismo la ventana de Hanning es la que ensancha los picos principales de la señal.

Se verificó que la señal adquirida y muestreada mantiene las características esperadas cuando se observa en el dominio del tiempo.

Se confirmó que aumentar N FFT mejora la resolución en el dominio de la frecuencia, pero con la desventaja de requerir mayor tiempo de procesamiento y mayor longitud de señal.

El análisis demostró que para obtener un muestreo exitoso en la Pico 2W es recomendable emplear herramientas avanzadas como DMA, PIO o incluso ADCs externos cuando se trabaja con señales de mayor frecuencia. Estas alternativas, junto con un filtrado adecuado y una fuente de referencia estable, permiten optimizar la captura de datos y asegurar un procesamiento digital más robusto y confiable en aplicaciones de telecomunicaciones.

VI. REFERENCIAS

- Administrador. (2023, 5 junio). Convertidor analógico digital. CAD - ADC - electrónica Unicrom. Electrónica Unicrom. https://unicrom.com/convertidor-analogico-digital-cad-adc

- Jrugeles. (s. f.). source_coding/sampling_2.py at main · jrugeles/source_coding. GitHub. https://github.com/jrugeles/source_coding/blob/main/ADC_testing.py
- Raspberry Pi Pico - El primer microcontrolador de Raspberry Pi Silicon. (s. f.). 330ohms. https://www.330ohms.com/es-co/products/raspberry-pi-pico
- Otero, C., & Otero, C. (2023, 16 julio). El increíble teorema de Nyquist: la clave para evitar la distorsión en las señales de audio y video. Tantalus. https://tantalus.es/el-increible-teorema-de-nyqui st-la-clave-para-evitar-la-distorsion-en-las-senal es-de-audio-y-video
- FFT. (s. f.). https://www.nti-audio.com/es/servicio/conocimi entos/transformacion-rapida-de-fourier-fft
- Guía para Análisis FFT. (2023, 10 mayo). Soluciones de Adquisición de Datos (DAQ). https://dewesoft.com/es/blog/guia-para-el-analisis-fft
- Analog Devices. (s. f.). *Maximum SNR vs. Clock Jitter*. Analog Devices. Recuperado el 9 de septiembre de 2025, de https://www.analog.com/en/resources/technical-articles/maximum-snr-vs-clock-jitter.html
- Raspberry Pi. (2023). *Getting started with Raspberry Pi Pico and RP2040*. Raspberry Pi Ltd. https://www.raspberrypi.com/documentation/microcontrollers/