Perfecto. Realizaré una investigación completa sobre el teorema de muestreo, el teorema de Nyquist, y la Transformada de Fourier, con explicaciones teóricas y aplicaciones prácticas. También incluiré un ejemplo detallado del uso del ADC con ESP32 y STM32, utilizando Arduino IDE, ESP-IDF y STM32CubeIDE. Te presentaré toda la información en español con explicaciones claras, ejemplos y código cuando sea relevante. En cuanto esté lista, te aviso.

Investigación: Teorema de muestreo, Teorema de Nyquist, Transformada de Fourier y ADC en ESP32/STM32

Teorema de muestreo (Shannon-Nyquist)

El **teorema de muestreo de Nyquist-Shannon** establece que una señal analógica de banda limitada puede **reconstruirse exactamente a partir de sus muestras** si la frecuencia de muestreo fsf_s es **mayor al doble de su ancho de banda (frecuencia máxima)**. En otras palabras, si la frecuencia más alta presente en la señal original es BB, necesitamos muestrearla con fs>2Bf_s > 2B. Bajo esta condición, *toda la información de la señal continua original queda capturada en la secuencia de muestras*, sin pérdida de contenido en frecuencia. Una señal que cumpla esto (limitada en BB Hz y muestreada a más de 2B2B Hz) se puede reconstruir perfectamente mediante interpolación **sinc** (función sinc): por ejemplo usando g(t)=sin⁡(2πBt)2πBtg(t) = \frac{\sin(2\pi B t)}{2\pi B t} como núcleo de interpolación. La reconstrucción teórica consiste en sumar cada muestra escalada por una sinc desplazada; así, la señal continua x(t)x(t) se obtiene como x(t)=∑n=−∞∞x(nT) g(t−nT)x(t)=\sum_{n=-\infty}^{\infty} x(nT)\,g(t-nT) con T=1/fsT=1/f_s. En la práctica, esta idea justifica que un convertidor digital-analógico (DAC) pueda recuperar una onda continua a partir de muestras, siempre que se haya cumplido el criterio de muestreo adecuado.

Importancia en procesamiento de señales: Este teorema es fundamental en la digitalización de señales analógicas. Permite saber a qué velocidad muestrear para no perder información, habilitando el procesamiento digital de audio, vídeo, telecomunicaciones, etc. **Ejemplos:** Un caso típico es el audio digital: el estándar CD usa fs=44.1 kHzf_s = 44.1~kHz para capturar fielmente audio de hasta ~20 kHz (doble de la máxima frecuencia audible). Así, cualquier contenido de audio por debajo de 20 kHz puede ser reconstruido a partir de las muestras. Otro ejemplo es en instrumentación: un osciloscopio digital debe muestrear al menos al doble del ancho de banda de la señal medida para representarla correctamente. En resumen, el teorema de muestreo brinda el **criterio de Nyquist** para el diseño de sistemas de adquisición: fija la frecuencia mínima de muestreo necesaria para **evitar pérdida de información o distorsiones** en la conversión analógico-digital.

Teorema de Nyquist, frecuencia de Nyquist y aliasing

En este contexto, suele llamarse **teorema de Nyquist** al criterio antes mencionado: la frecuencia de muestreo debe exceder el doble de la frecuencia máxima de la señal. Se define la **frecuencia de Nyquist** fN=fs/2f_N = f_s/2, que es la máxima frecuencia que puede representarse de forma unívoca con una dada frecuencia de muestreo. Si intentamos muestrear una señal que contenga componentes por encima de fNf_N, ocurre el fenómeno de **aliasing**: las componentes altas se "mezclan" y aparecen falsas frecuencias bajas en la señal muestreada. En otras palabras, al reconstruir obtendríamos frecuencias que **no estaban en la señal original**, producto de haber submuestreado. Por ejemplo, al muestrear una onda senoidal de 800 kHz a solo 1 MS/s, esta aparece en el dominio digital como una senoidal de 200 kHz (un alias) que no corresponde a la original. La Figura 7 ilustra este caso: la señal continua de alta frecuencia (línea gris) es muestreada insuficientemente y los puntos tomados (flechas negras) producen una forma reconstruida errónea (línea punteada roja) de frecuencia mucho más baja. En general, **si fs<2Bf_s < 2B, distintas señales continuas pueden dar las mismas muestras**, haciendo imposible distinguirlas tras la digitalización.

Para **evitar el aliasing**, el muestreo debe cumplir la condición fs≥2Bf_s \ge 2B (criterio de Nyquist), o equivalentemente T≤1/(2B)T \le 1/(2B) en el período de muestreo. En la práctica real, suele tomarse un margen mayor (sobre-muestreo) para facilitar el filtrado. Además, es **obligatorio filtrar** la señal analógica antes de muestrear: se utiliza un **filtro antialias** de tipo pasa-bajos cortado en fNf_N para eliminar componentes por encima de esa frecuencia. De este modo, el ADC no recibirá contenido prohibido que pueda plegarse (alias) al digitalizar. Tras la conversión, el proceso inverso (reconstrucción mediante DAC) también aplica un filtro pasa-bajos de reconstrucción para suavizar la onda y remover replicas espectrales. En resumen, el teorema de Nyquist implica que **en sistemas digitales debemos limitar el ancho de banda de la señal de entrada al ADC o muestrear suficientemente rápido**. Sus implicaciones son críticas en audio digital, video, radio digital, instrumentación y cualquier sistema de tiempo discreto: garantizan que la señal digital represente fielmente a la analógica original en la banda de interés, evitando distorsiones por solapamiento espectral (aliasing).

Transformada de Fourier (continua y discreta)

La **Transformada de Fourier** es una operación matemática que descompone una señal en las **frecuencias sinusoidales** que la componen. En su forma continua, la transformada de Fourier de una función f(t)f(t) se define como una integral de la señal multiplicada por un exponencial complejo: F(ω)=∫−∞∞f(t) e−iωt dtF(\omega) = \int_{-\infty}^{\infty} f(t)\,e^{-i\omega t}\,dt. Esta integral (y su inversa) permiten pasar del dominio del tiempo al **dominio de la frecuencia** y viceversa. Intuitivamente, Fourier nos dice que cualquier señal (incluso no periódica) puede

expresarse como suma de senos y cosenos de distintas frecuencias, amplitudes y fases. La transformada continua F(ω)F(\omega) nos da el **espectro de frecuencia** de f(t)f(t), indicando cuánto de cada frecuencia contiene la señal. Por ejemplo, en procesamiento de audio podríamos ver qué componentes tonales (Hz) están presentes en una grabación, o en imágenes podríamos analizar las componentes de alta y baja frecuencia de la intensidad de píxeles.

Cuando la señal es *discreta* (tiempo discreto), usamos la **Transformada Discreta de Fourier (DFT)**. La DFT toma una secuencia finita de N muestras x(0),x(1),...,x(N−1)x(0), x(1), ..., x(N-1) y la convierte en **N valores complejos** X(0),...,X(N−1)X(0), ..., X(N-1) que representan las frecuencias de esa señal muestreada. La definición matemática de la DFT es:

X(k)=∑n=0N−1x(n) e−i2πkn/N,k=0,1,...,N−1:contentReference[oaicite:21]index=21. X(k) = \sum_{n=0}^{N-1} x(n)\, e^{-i 2\pi k n / N}, \quad k=0,1,\dots,N-1:contentReference[oaicite:21]{index=21}.

Cada término X(k)X(k) corresponde a una componente sinusoidal de frecuencia k/Nk/N del muestreo (es decir, kk ciclos en la ventana de N puntos). Los valores de X(k)X(k) son complejos: su magnitud |X(k)||X(k)| indica la amplitud de esa frecuencia y su argumento la fase. La transformada inversa (IDFT) permite reconstruir x(n)x(n) a partir de los X(k)X(k) de forma similar (suma de exponenciales). En la práctica, para señales de tiempo discreto de duración infinita o periódicas, existen variantes como la Transformada de Fourier de Tiempo Discreto (DTFT) o las Series de Fourier. Además, algoritmos eficientes como la **Transformada Rápida de Fourier (FFT)** permiten calcular la DFT rápidamente incluso para N muy grandes, siendo fundamentales en aplicaciones en tiempo real.

Aplicaciones en procesamiento de señales: La transformada de Fourier, continua o discreta, es una **herramienta esencial** en ingeniería. Permite analizar y filtrar señales de forma más sencilla en frecuencia que en tiempo. Por ejemplo, el filtrado lineal se vuelve multiplicación en el dominio de Fourier (según el Teorema de Convolución). En procesamiento de audio, la DFT se usa para **ecualización**, reducción de ruido y **compresión** mediante el aislamiento de bandas frecuenciales. En compresión de imágenes (por ejemplo JPEG), se aplica una transformación similar a Fourier (como la DCT) para conservar componentes importantes y descartar las de alta frecuencia que el ojo no percibe con detalle. En telecomunicaciones, la DFT permite modular/demodular señales (OFDM, por ejemplo) y analizar el espectro para una transmisión eficiente. En el ámbito médico e industrial, Fourier se usa para **análisis de vibraciones**, detección de fallas, análisis de espectros en resonancia magnética, etc.. En resumen, la Fourier continua brinda fundamentos teóricos (análisis de sistemas, solución de ecuaciones diferenciales, etc.), mientras que la DFT es la base del **procesamiento**

digital de señales: desde los algoritmos de sonido e imagen, hasta las comunicaciones digitales y la visión por computador, aprovechando nuestra capacidad de manipular la representación frecuencial de una señal para *filtrar, comprimir o sintetizar* información de manera más efectiva.

Ejemplos prácticos: ADC en microcontroladores ESP32 y STM32

A continuación, se muestra cómo usar en la práctica un **convertidor analógico-digital (ADC)** tanto en un ESP32 como en un STM32. Veremos ejemplos de código y configuraciones en cada caso, ilustrando cómo leer señales analógicas en estos microcontroladores.

Lectura ADC en ESP32 con Arduino IDE

En el entorno Arduino, usar el ADC del ESP32 es **tan sencillo como en un Arduino UNO** tradicional, gracias a la función de alto nivel analogRead(). El ESP32 cuenta con dos módulos ADC SAR de 12 bits (ADC1 y ADC2), cada uno multiplexado a varios pines GPIO. Antes de leer, debemos asegurarnos de conectar la señal al pin ADC adecuado (y nunca superar 3.3V, porque el ADC del ESP32 no tolera más voltaje). Arduino por defecto configura el ADC a **12 bits** de resolución (0-4095) y usa una atenuación que permite leer aproximadamente 0–3.3V. Podemos ajustar la resolución con analogReadResolution(bits) si se desea, pero normalmente no es necesario.

Ejemplo: Supongamos un potenciómetro conectado al pin A0 (GPIO36, canal ADC1_0 en ESP32). El siguiente código lee el valor ADC y lo envía por serial:

```
void setup() {
  Serial.begin(9600);
}


void loop() {
  int sensorValue = analogRead(A0);  // Leer pin analógico A0
  Serial.println(sensorValue);      // Imprimir valor (0-4095)
  delay(100);
}
```

En este código, analogRead(A0) retorna el valor digital de 0 a 4095 correspondiente al voltaje medido en A0. Un valor cercano a 0 indica ~0V y cercano a 4095 indica ~3.3V. El ESP32, a diferencia de arduinos de 10 bits, ofrece 12 bits de resolución (~0.8 mV por nivel). La lectura se realiza de manera *bloqueante* (espera el resultado

inmediatamente) y podemos realizarla en el loop() periódicamente. Cabe destacar que si el WiFi del ESP32 está activo, **no se deben usar los pines del ADC2** para analogRead, ya que comparten hardware con WiFi y pueden dar resultados erróneos. Por seguridad, use siempre los pines ADC1 si WiFi está en uso.

Lectura ADC en ESP32 con ESP-IDF (C/C++ nativo)

En el SDK nativo ESP-IDF de Espressif, el uso del ADC requiere algunos pasos de configuración, pero ofrece más control. En ESP-IDF (versión 4.x), típicamente se usa el *driver* ADC en modo de lectura única (one-shot). Los pasos básicos son:

1. **Configurar ancho de bits:** El ADC del ESP32 soporta hasta 12 bits de resolución. Se configura con adc1_config_width(ADC_WIDTH_BIT_12); para el ADC1.
2. **Configurar atenuación del canal:** La atenuación define el rango de voltaje medible. Por ejemplo, adc1_config_channel_atten(ADC1_CHANNEL_0, ADC_ATTEN_DB_11); configurería el canal 0 (GPIO36) con atenuación 11dB, que extiende el rango aprox. a 0~3.3V. (0dB sería ~0-1.1V, 6dB ~0-2.0V, etc., según la tabla de ESP32).
3. **Leer valor ADC:** Se invoca la lectura con int valor = adc1_get_raw(ADC1_CHANNEL_0);, que retorna el conteo digital (0–4095 a 12 bits). Este valor crudo puede convertirse a milivoltios si se aplica calibración, usando esp_adc_cal_raw_to_voltage() con los coeficientes de calibración si están disponibles en eFuse.

Con el enfoque anterior (APIs de driver/adc.h), un ejemplo mínimo dentro de app_main() sería:

```
#include "driver/adc.h"

...

// Configuración ADC1

adc1_config_width(ADC_WIDTH_BIT_12);

adc1_config_channel_atten(ADC1_CHANNEL_0, ADC_ATTEN_DB_11);

while (1) {

  int lectura = adc1_get_raw(ADC1_CHANNEL_0);

  printf("ADC lectura: %d\n", lectura);

  vTaskDelay(pdMS_TO_TICKS(1000));

}
```

En este código de ESP-IDF, primero se establece la resolución a 12 bits y la atenuación a 11dB en el canal deseado. Luego en el bucle, adc1_get_raw realiza una conversión simple y obtenemos el resultado inmediatamente. Es importante no usar funciones ADC2 simultáneamente con WiFi activo (misma restricción mencionada). Las APIs de ESP-IDF brindan más funcionalidades, como *modo continuo*, lectura del sensor de efecto Hall interno, o configuración de múltiples canales con DMA, pero el ejemplo mostrado cubre el caso más común de leer un valor analógico puntual bajo demanda. Para mayor precisión, Espressif provee un driver de **calibración ADC** que corrige variaciones de referencia interna, así como recomendaciones de hardware (ej. colocar un condensador de 0.1µF en la entrada para reducir ruido).

Lectura ADC en STM32 con STM32CubeIDE (HAL)

En los microcontroladores STM32, el ADC también es de tipo SAR (aproximación sucesiva) con resoluciones típicas de 12 bits (algunas familias permiten hasta 16 bits mediante sobremuestreo). Usaremos la biblioteca **HAL** generada por CubeMX/CubeIDE para configurar y leer el ADC de forma sencilla. Los pasos generales son:

1. **Configuración con CubeMX:** En STM32CubeIDE, habilitar el periférico ADC en el microcontrolador seleccionado, elegir el canal (pin analógico) que leeremos, y configurar parámetros como resolución (ej. 12 bits), modo de conversión (simple o continuo), alineación de datos, etc. Por ejemplo, para leer el canal ADC_IN0 en modo sencillo, se puede activar *Continuous Conversion* si queremos lecturas periódicas automáticas, o usar modo single-shot para lecturas bajo demanda. También podemos activar el **modo escaneo** si queremos muestrear múltiples canales en secuencia, o habilitar el **DMA** para transferir automáticamente muestras a memoria – pero mantendremos el ejemplo simple por ahora.
2. **Código HAL de lectura (bloqueante):** La HAL proporciona funciones para controlar el ADC. Las principales son: HAL_ADC_Start() para iniciar una conversión, HAL_ADC_PollForConversion() para esperar (polling) hasta que haya terminado, y HAL_ADC_GetValue() para obtener el resultado convertido. Opcionalmente, HAL_ADC_Stop() para detener el ADC si no está en continuo.

Ejemplo: Supongamos un STM32 con un potenciómetro en el canal ADC1_IN0. Tras la configuración inicial generada por CubeIDE (que crea una estructura ADC_HandleTypeDef hadc1), podemos leer el valor así:

```
HAL_ADC_Start(&hadc1);                    // Iniciar conversion ADC
```

HAL_ADC_PollForConversion(&hadc1, HAL_MAX_DELAY); // Esperar fin de conversión

uint32_t valor = HAL_ADC_GetValue(&hadc1); // Leer resultado (por ej., 0-4095)

En este fragmento, la primera llamada inicia la conversión en el ADC1, la segunda bloquea hasta que la conversión finalice (aquí con espera indefinida usando HAL_MAX_DELAY, aunque podríamos especificar un timeout en milisegundos), y la tercera obtiene el valor digital convertido. Después de HAL_ADC_GetValue, el ADC típicamente queda listo para una nueva conversión si se llama nuevamente a Start. Si configuramos el ADC en *modo continuo*, tras el Start el hardware muestreará de forma repetida y podremos leer valores sucesivos sin tener que iniciar cada vez. En modo de *inyección única* (single conversion mode), cada Start produce solo una muestra.

Consideraciones adicionales en STM32: Muchos STM32 incluyen características avanzadas en el ADC, como canales internos (temperatura, voltaje de referencia), modo de **inyección** (conversiones disparadas por eventos externos o timer), **watchdogs analógicos** (monitoreo de niveles fuera de rango), etc.. En nuestro ejemplo básico no profundizamos en ellas. También es común usar **DMA** para adquirir muestras continuamente sin carga de CPU: por ejemplo con HAL_ADC_Start_DMA(&hadc1, buffer, longitud) se pueden obtener lecturas periódicas a un buffer, ideal para señales a alta velocidad. En aplicaciones de tiempo real, a veces se usan interrupciones (HAL_ADC_ConvCpltCallback) en lugar de polling para no bloquear la CPU mientras el ADC convierte. Sin embargo, para *leer un sensor lentamente* (ej. un potenciómetro cada tanto), el método mostrado es suficiente y claro.

Diagrama simplificado del funcionamiento interno de un ADC SAR en STM32 (ejemplo general para STM32 L4). El multiplexor selecciona la entrada analógica, un circuito de seguimiento y retención toma la muestra, y el convertidor SAR la digitaliza comparando con un DAC interno bit a bit. El resultado de 12 bits se almacena en registros de datos (regular e inyectado), pudiendo generar interrupciones o peticiones DMA cuando la conversión termina.

En resumen, para leer un ADC en STM32 via HAL: **(a)** se configura el periférico y canal en CubeMX, **(b)** se inicia la conversión con HAL_ADC_Start, **(c)** se espera el resultado (vía *polling* o por interrupción/DMA), y **(d)** se obtiene el valor con HAL_ADC_GetValue. A partir de ese valor digital (0 a $2^{N_bits}-1$), podemos convertir a unidades de ingeniería sabiendo el voltaje de referencia y la resolución. Por ejemplo, con 12 bits y Vref=3.3V, el valor 4095 equivale aprox. 3.3V y 2048 ~1.65V, de forma lineal. Si se requiere más precisión, se puede calibrar el ADC (muchos STM32 permiten calibración interna para ajustar offset y ganancia).

Aplicación típica: Imaginemos que en un STM32 leemos un sensor de temperatura analógico LM35 (0-100°C mapeado a 0-1V). Configuramos el ADC a 12 bits, Vref=3.3V. Al leer el canal, obtendremos un número entre 0 y 4095. Podemos convertirlo a voltaje: V=valor4095×3.3VV = \frac{\text{valor}}{4095} \times 3.3\text{V}. Luego, dado que 10 mV = 1°C en el LM35, la temperatura sería T(°C)=V×100T(°C) = V \times 100. Este simple flujo muestra cómo del **dato crudo ADC** pasamos a una magnitud física utilizable. Tanto en ESP32 como en STM32 (y en general cualquier microcontrolador con ADC) los pasos son análogos: configurar, muestrear periódicamente o bajo demanda, y procesar el valor digital para obtener la información del mundo real que deseamos (sensor, voltaje, etc.).

Referencias técnicas: Para profundizar, es recomendable consultar la documentación oficial. Espressif proporciona el *ESP32 Technical Reference Manual* y la guía del API ESP-IDF (sección ADC), donde se discuten temas como precisión, ruido, calibraje y uso del ADC en bajo consumo. STMicroelectronics ofrece notas de aplicación y el capítulo de ADC en el Reference Manual de cada familia STM32, explicando detalles de modos de conversión, tiempos de muestreo, y limitaciones específicas (ej. impedancia de fuente recomendada, etc.) que son cruciales para un diseño robusto con ADC. Estas fuentes oficiales, junto con bibliografía IEEE y textos de procesamiento digital de señales, respaldan los conceptos teóricos y prácticos presentados en este informe.