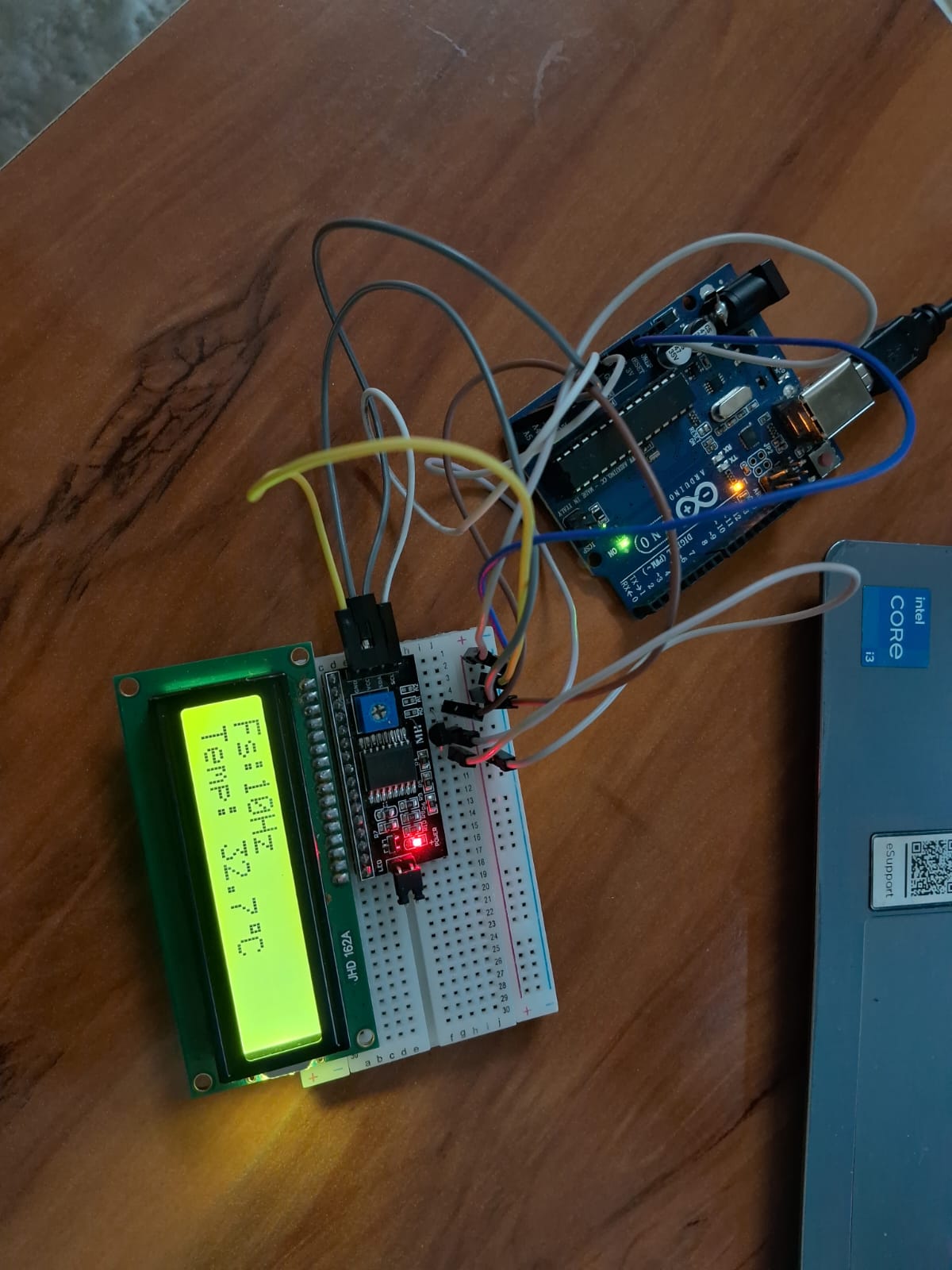
### **Project Overview: "Analog to Digital Conversion and Spectrum Analysis of LM35 Signal"**

In this project, we are converting the analog signal from the LM35 temperature sensor (which provides a voltage output proportional to the temperature) to a digital signal using the Arduino's ADC (Analog to Digital Converter). We then analyze the signal in both the time domain and frequency domain using Fast Fourier Transform (FFT) for spectrum analysis.  
  


#### **Steps Involved:**

1. **Analog to Digital Conversion (ADC):**
   * The LM35 outputs an analog voltage (10 mV per °C). This voltage is read by the Arduino's ADC (in the range 0–1023, corresponding to 0–5V).
   * The analog signal is converted to a digital signal by the ADC, and the resulting data is displayed on an LCD and sent to the Serial Monitor in CSV format for further analysis.
2. **Data Analysis:**
   * The collected data is used to analyze the signal in both the **time domain** and the **frequency domain**.
   * **Time Domain:** Displays how the signal varies over time.
   * **Frequency Domain:** Uses FFT to convert the time-domain signal into the frequency domain, providing insights into the signal's frequency components.
3. **Visualization:**
   * The analog and digital signals are plotted in the time domain to visualize their behavior.
   * The frequency spectrum of the analog and digital signals is analyzed using FFT.

### **Sampling Theory and Concepts:**

#### **1. Sampling:**

* **Sampling** refers to the process of converting a continuous-time signal into a discrete-time signal. In this project, the analog voltage from the LM35 sensor is sampled at regular intervals by the Arduino's ADC.
* **Sampling Rate (Fs):** The rate at which samples are taken from the continuous signal. In this case, the sampling rate is 10 Hz, meaning one sample is taken every 100 ms.
* **Nyquist Theorem:** This states that to accurately represent a continuous signal, the sampling rate must be at least twice the highest frequency present in the signal (i.e., the Nyquist rate). For example, if the highest frequency component of the LM35 signal is 5 Hz, a minimum sampling rate of 10 Hz is required to avoid aliasing.

#### **2. Quantization:**

* **Quantization** is the process of mapping the continuous amplitude of the analog signal to discrete values. In this project, we used an 8-bit quantization, meaning the analog signal is mapped to values between 0 and 255.

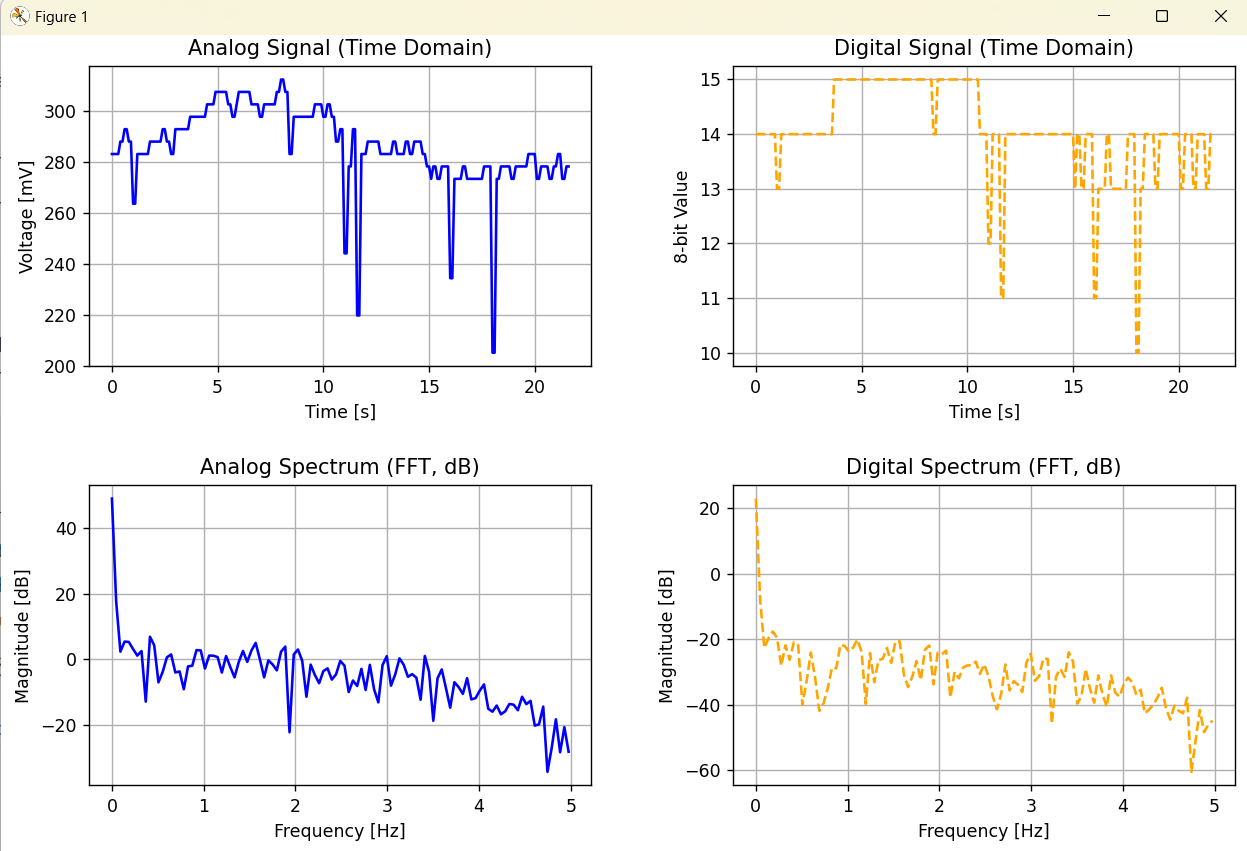
#### **3. Aliasing:**

* **Aliasing** occurs when the sampling rate is too low to accurately capture the signal, leading to distortion. If the sampling rate is below the Nyquist rate, high-frequency components will appear as low-frequency components in the sampled signal.

#### **4. FFT (Fast Fourier Transform):**

* **FFT** is a mathematical algorithm used to convert a signal from the time domain to the frequency domain. The result of an FFT is a spectrum that shows the magnitude of different frequency components present in the signal.
* In this project, FFT is used to analyze the frequency components of both the analog and digital signals.

### **Detailed Explanation of Each Graph:**



#### **1. Analog Signal (Time Domain):**

* The first graph shows the analog signal (in millivolts) as a function of time. This represents the variation of the temperature (from the LM35 sensor) over time.
* **Comment:** The analog signal is continuous and reflects the temperature changes of the LM35 sensor. The curve is relatively smooth, indicating stable temperature fluctuations without sudden jumps (since the LM35 provides a steady analog output).

#### **2. Digital Signal (Time Domain):**

* The second graph displays the digital representation of the analog signal, where the continuous voltage is quantized to discrete levels.
* **Comment:** The digital signal looks like a staircase because it has been quantized to 8-bit levels. Each step corresponds to the ADC's conversion of the continuous voltage to a discrete value (ranging from 0 to 255). This quantization is a natural artifact of the ADC process.

#### **3. Analog Spectrum (FFT, dB):**

* The third graph shows the **frequency spectrum** of the analog signal using FFT. The x-axis represents the frequency (in Hz), and the y-axis represents the magnitude (in dB).
* **Comment:** The analog spectrum gives insight into the frequency content of the LM35 signal. Since the LM35's output is a low-frequency signal (due to slow temperature changes), we would expect to see a dominant peak at low frequencies (for example, around 0–1 Hz, reflecting gradual temperature variations). Higher frequency components should be absent, which is expected since the LM35 is a low-frequency analog sensor.

#### **4. Digital Spectrum (FFT, dB):**

* The fourth graph shows the **frequency spectrum** of the digital signal. Similar to the analog spectrum, this graph displays the magnitude of frequency components present in the digital signal.
* **Comment:** The digital spectrum should have similar characteristics to the analog spectrum, but we may notice higher-frequency noise due to the quantization process. Since the signal is discrete, small fluctuations or rounding errors in the digital signal may introduce high-frequency components.

### **Spectrum Analysis and Comment:**

1. **Analog Signal Spectrum:**
   * In the case of the analog signal, we expect the frequency spectrum to primarily contain low-frequency components. The LM35's temperature-related signal is slow, and thus its frequency content should predominantly lie at lower frequencies (e.g., below 1 Hz). The magnitude of the spectrum at higher frequencies should be low, indicating little to no high-frequency noise or unwanted signals.
2. **Digital Signal Spectrum:**
   * In the digital signal's spectrum, you may notice additional high-frequency components due to **quantization noise**. Quantization errors (from the ADC) introduce small fluctuations that show up as high-frequency noise in the spectrum. This is typical for signals that have been sampled and quantized.
3. **Aliasing:**
   * If the sampling rate were too low, we'd observe **aliasing**, where higher-frequency components from the analog signal would be "folded" into lower frequencies in the digital spectrum. This could distort the digital signal's spectrum and make it hard to distinguish between real and aliased components. However, since the sampling rate here is sufficient (10 Hz), aliasing is unlikely to be a problem.

### **Conclusion:**

This project demonstrates the process of converting an analog signal (LM35 temperature sensor output) to a digital signal using an ADC, visualizing it in the time domain, and performing spectrum analysis using FFT. The analog signal reflects slow temperature changes, while the digital signal introduces quantization errors that result in high-frequency noise. The frequency spectrum of both signals helps identify their frequency content and reveals the impact of quantization on the digital signal.

### **How FFT (Fast Fourier Transform) Was Performed:**

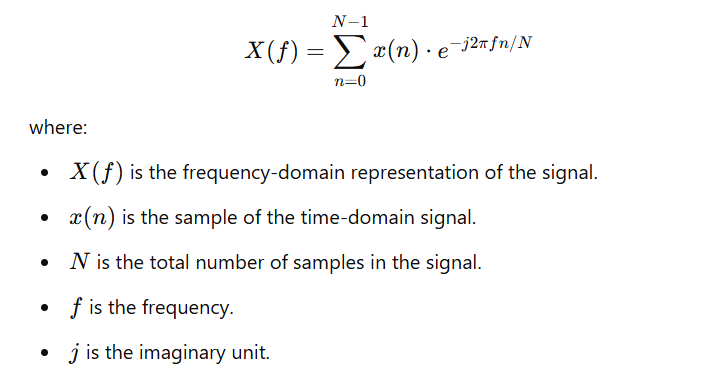
The **Fast Fourier Transform (FFT)** is an efficient algorithm for calculating the **Discrete Fourier Transform (DFT)** of a signal. The DFT is used to convert a signal from the time domain into the frequency domain, revealing the frequency components that make up the signal. In our case, we used FFT to analyze both the **analog** and **digital** signals obtained from the LM35 temperature sensor.

Here’s a step-by-step explanation of how we performed the FFT in the context of this project:

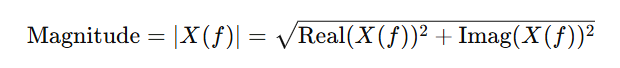
### **1. Data Collection:**

* We collected the analog and digital signals from the LM35 sensor using an Arduino’s ADC. The analog signal is continuously monitored, and the digital version is derived from it using quantization (8-bit resolution).
* Both the **analog signal** and the **digital signal** are stored in separate arrays (analog and digital) for FFT processing.

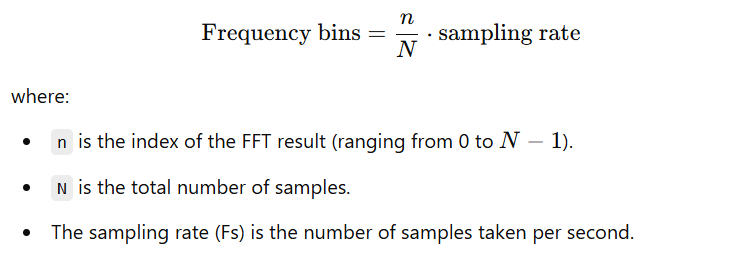
### **2. FFT Calculation:**

* **FFT Function:**
  + We used the fft function from the **SciPy** library in Python, which implements the FFT algorithm.
  + The fft function computes the **discrete Fourier transform (DFT)** of the input signal and returns the complex result, where each value represents the amplitude and phase information for each frequency component of the signal.
  + The formula for DFT is as follows:  
    
  + The FFT algorithm efficiently computes the DFT in O(Nlog⁡N)O(N \log N)O(NlogN) time, which is much faster than the direct DFT computation.

### **3. Handling Frequency Data:**

* After performing the FFT, we get an array of **complex numbers** (real and imaginary parts). To analyze the **magnitude** of the frequency components, we calculate the **absolute value** of each complex number in the FFT output.  
  
* This magnitude represents the strength (amplitude) of each frequency component in the signal.

### **4. Frequency Axis Setup:**

* The FFT output contains both **positive** and **negative** frequencies, but for most real-world signals, we are only interested in the positive frequencies.
* The **frequency bins** are calculated using the fftfreq function from the scipy.fft module, which generates the frequencies corresponding to the FFT results. These are based on the **sampling rate** and the number of samples (N).  
  
* Only the positive frequencies are kept, as they correspond to the actual frequency components of the signal.

### **5. Normalization and Scaling:**

* To properly interpret the FFT results, we **normalize** the magnitude of the frequency components by dividing by the total number of samples (N).
* We also convert the magnitude to **decibels (dB)** for better visualization. The dB scale is often used in signal processing to represent the ratio between the signal strength and a reference level (usually the lowest possible value).  
  
* This scaling helps to visualize frequency components with better contrast, especially when the magnitude varies over many orders of magnitude.

### **6. Plotting the Spectrum:**

* After calculating the FFT magnitudes, we plot the frequency spectrum. The x-axis represents the **frequency (Hz)**, and the y-axis represents the **magnitude in dB**.
* This allows us to see which frequencies are most prominent in the signal.  
  + **Peaks in the spectrum** indicate the presence of specific frequency components.
  + The shape of the spectrum provides insight into the nature of the signal, such as its periodicity or any noise present.

### **7. Handling Aliasing and Sampling:**

* According to the **sampling theorem (Nyquist-Shannon theorem)**, we need to sample the signal at a frequency higher than twice the highest frequency present in the signal (the Nyquist rate).
* In this project, the sampling rate is 10 Hz, which is sufficient for the slow-changing temperature data from the LM35 sensor. However, if the signal had high-frequency components, we might need to sample at a higher rate to avoid **aliasing** (where high-frequency signals are incorrectly represented as lower frequencies).

### **Conclusion:**

FFT allows us to analyze the frequency content of the LM35 temperature sensor's signal. By converting the signal from the time domain to the frequency domain, we can identify the frequency components of the signal and determine if there are any high-frequency noise or unwanted components. In this project, FFT analysis helps visualize the underlying frequency behavior of both the analog and digital signals.

CODES

1. Arduino Code

#include <Wire.h>

#include <LiquidCrystal\_I2C.h>

// I2C LCD address, adjust to your module's

LiquidCrystal\_I2C lcd(0x27, 16, 2);

const int lm35Pin = A0;

const float referenceVoltage = 5.0;

const int adcResolution = 1024;

unsigned long prevSampleTime = 0;

const int samplingIntervalMs = 100; // 10 Hz → 100ms

float samplingFreq = 1000.0 / samplingIntervalMs;

byte degreeSymbol[8] = {

0b00111,

0b00101,

0b00111,

0b00000,

0b00000,

0b00000,

0b00000,

0b00000

};

void setup() {

Serial.begin(9600);

lcd.begin(16,2);

lcd.createChar(0, degreeSymbol); // create custom char at location 0

lcd.backlight();

lcd.setCursor(0, 0);

lcd.print("Fs:");

lcd.setCursor(0, 1);

lcd.print("Temp:");

}

void loop() {

unsigned long currentMillis = millis();

if (currentMillis - prevSampleTime >= samplingIntervalMs) {

prevSampleTime = currentMillis;

int rawADC = analogRead(lm35Pin); // 10-bit ADC value (0–1023)

float mV = (rawADC \* referenceVoltage \* 1000.0) / adcResolution;

float temperature = mV / 10.0; // LM35 gives 10mV/°C

// 8-bit quantization (0–255)

int digital8bit = map(rawADC, 0, 1023, 0, 255);

// Show on LCD

lcd.setCursor(3, 0);

lcd.print(" ");

lcd.setCursor(3, 0);

lcd.print(samplingFreq, 0);

lcd.print("Hz");

lcd.setCursor(6, 1);

lcd.print(" ");

lcd.setCursor(6, 1);

lcd.print(temperature, 1);

lcd.write(byte(0)); // write custom degree symbol

lcd.print("C");

// Serial Monitor Output (CSV format)

Serial.print(mV, 2); // mV

Serial.print(",");

Serial.println(digital8bit); // Digital (8-bit)

// Serial Plotter Output (for graphing purposes)

Serial.print(mV); // analog signal (in mV)

Serial.print(",");

Serial.println(digital8bit); // digital signal

}

}

1. Python code to visualize plots

import numpy as np

import matplotlib.pyplot as plt

from scipy.fft import fft, fftfreq

import pandas as pd

# Load data

data = pd.read\_csv("data\_corrected.csv", names=["mV", "digital"])

analog = data["mV"].values

digital = data["digital"].values

# Sampling info

Fs = 10 # Hz

N = len(analog)

T = 1.0 / Fs

time = np.arange(N) \* T

# FFT computation

analog\_fft = fft(analog)

digital\_fft = fft(digital)

freqs = fftfreq(N, T)

# Only positive frequencies

pos\_mask = freqs >= 0

freqs = freqs[pos\_mask]

analog\_fft\_mag = np.abs(analog\_fft[pos\_mask]) / N

digital\_fft\_mag = np.abs(digital\_fft[pos\_mask]) / N

# Convert to dB scale for better spectral view

analog\_db = 20 \* np.log10(analog\_fft\_mag + 1e-6) # avoid log(0)

digital\_db = 20 \* np.log10(digital\_fft\_mag + 1e-6)

# Plotting

plt.figure(figsize=(12, 10))

# 1. Analog Time Domain

plt.subplot(2, 2, 1)

plt.plot(time, analog, color='blue')

plt.title("Analog Signal (Time Domain)")

plt.xlabel("Time [s]")

plt.ylabel("Voltage [mV]")

plt.grid(True)

# 2. Digital Time Domain

plt.subplot(2, 2, 2)

plt.plot(time, digital, color='orange', linestyle='--')

plt.title("Digital Signal (Time Domain)")

plt.xlabel("Time [s]")

plt.ylabel("8-bit Value")

plt.grid(True)

# 3. Analog Spectrum

plt.subplot(2, 2, 3)

plt.plot(freqs, analog\_db, color='blue')

plt.title("Analog Spectrum (FFT, dB)")

plt.xlabel("Frequency [Hz]")

plt.ylabel("Magnitude [dB]")

plt.grid(True)

# 4. Digital Spectrum

plt.subplot(2, 2, 4)

plt.plot(freqs, digital\_db, color='orange', linestyle='--')

plt.title("Digital Spectrum (FFT, dB)")

plt.xlabel("Frequency [Hz]")

plt.ylabel("Magnitude [dB]")

plt.grid(True)

plt.tight\_layout()

plt.show()