# LAB 3 - Report 3

## Introduction

In this laboratory work, we implemented digital filtering techniques—specifically **Finite Impulse Response (FIR)** and **Infinite Impulse Response (IIR)** low-pass filters—on an embedded system using FreeRTOS. The ADC and DAC functionalities used in this lab are based on the working implementation from Laboratory Work 2, where an ESP32-S3 microcontroller sampled and transmitted analog signals.

## FIR Filter Design and Implementation

A 16th-order low-pass FIR filter was designed using the Hamming window method. The filter design was implemented in Python using the scipy.signal.firwin() function.

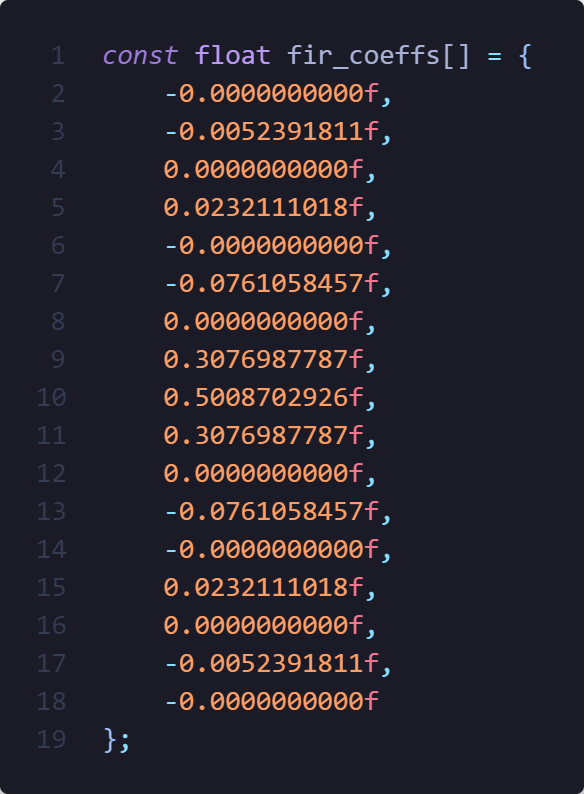
Sampling frequency (fs): 1000 Hz

Cutoff frequency (fc): 250 Hz (0.25 \* fs)

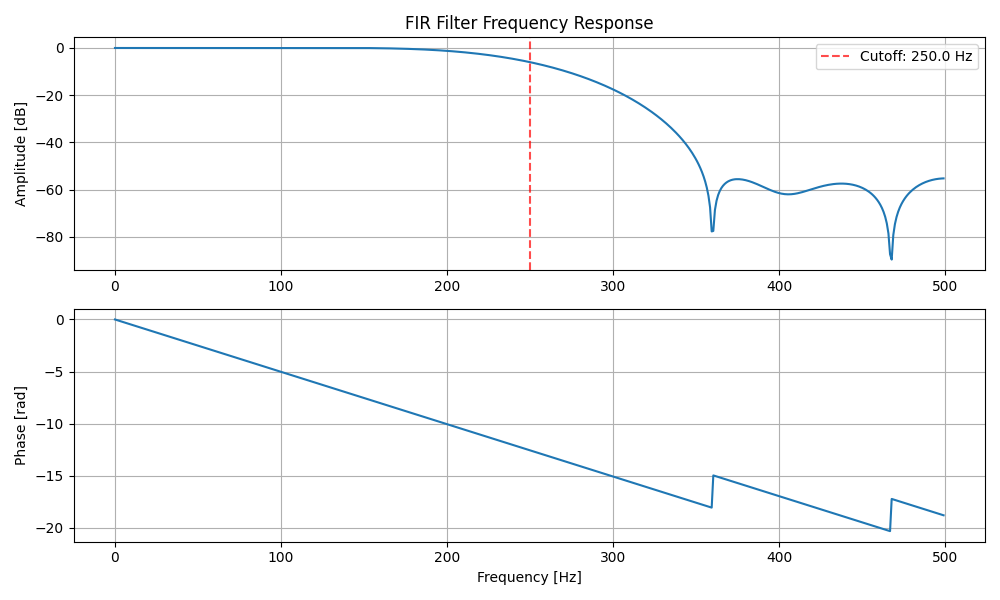
Normalized cutoff frequency (wc): 0.5

Filter order: 16 (i.e., 17 coefficients)

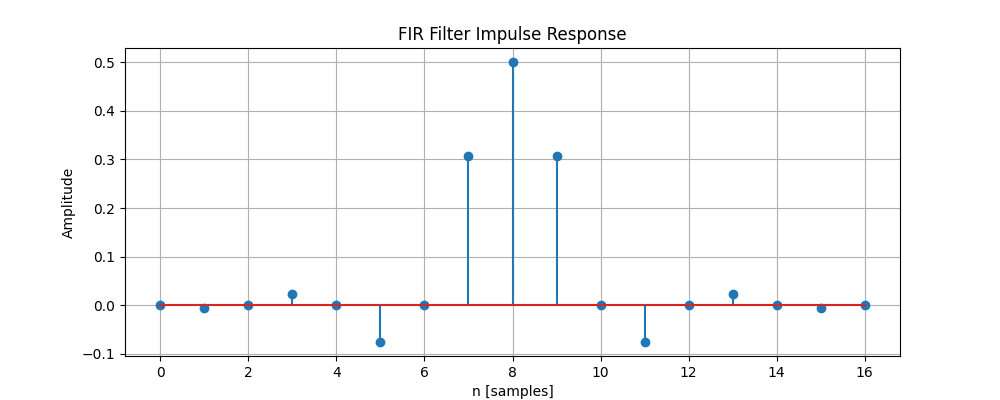
The coefficients were printed for embedding in the ESP32 firmware. Below is the C-style array for use in the ADC/DAC FreeRTOS task:



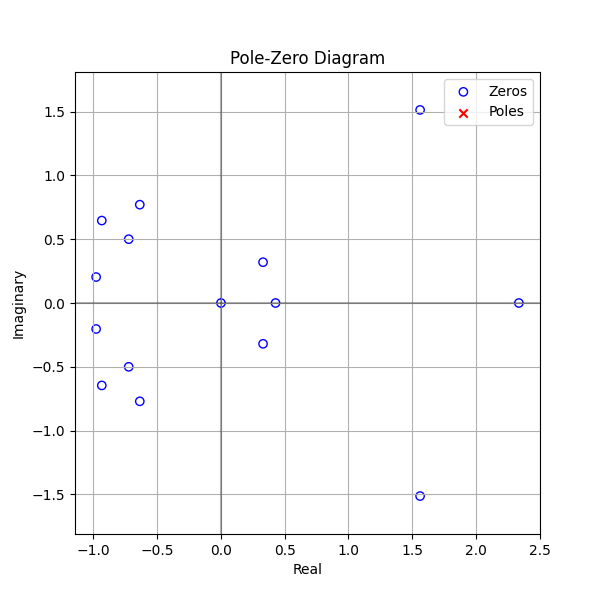
The filter's frequency response cutoff near 250 Hz, effectively attenuating frequencies above that. The phase response is linear, typical of FIR filters and beneficial for signal integrity.



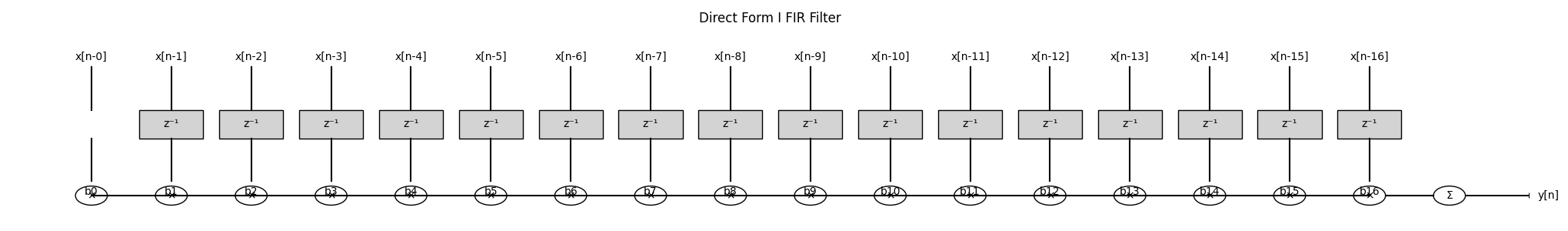
The symmetric impulse response shows the expected windowed sinc shape, indicating linear phase.



Pole-Zero Diagram



The filter was visualized using a Direct Form I block diagram. Each coefficient multiplies the delayed input, and all products are summed.



## Real-Time Embedded Implementation

The FIR filter was implemented on the ESP32-S3 XIAO Sense in FreeRTOS as a dedicated task.

ADC Sampling and PWM-based DAC were reused from Lab Work 2.The filtering task reads raw ADC samples into a buffer, applies the FIR filter using the above coefficients, and outputs filtered samples via DAC (PWM).

Filtered signals were analyzed using an oscilloscope and/or captured via serial for Python analysis.

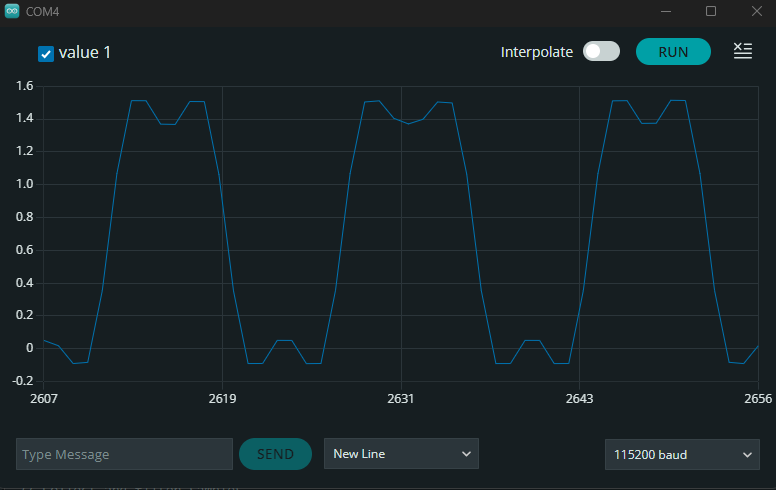
## Signal Evaluation and Results

The filter was tested using a 100 Hz square wave input generated by a function generator. The input and output signals were captured using both the serial monitor and analyzed further in Python.

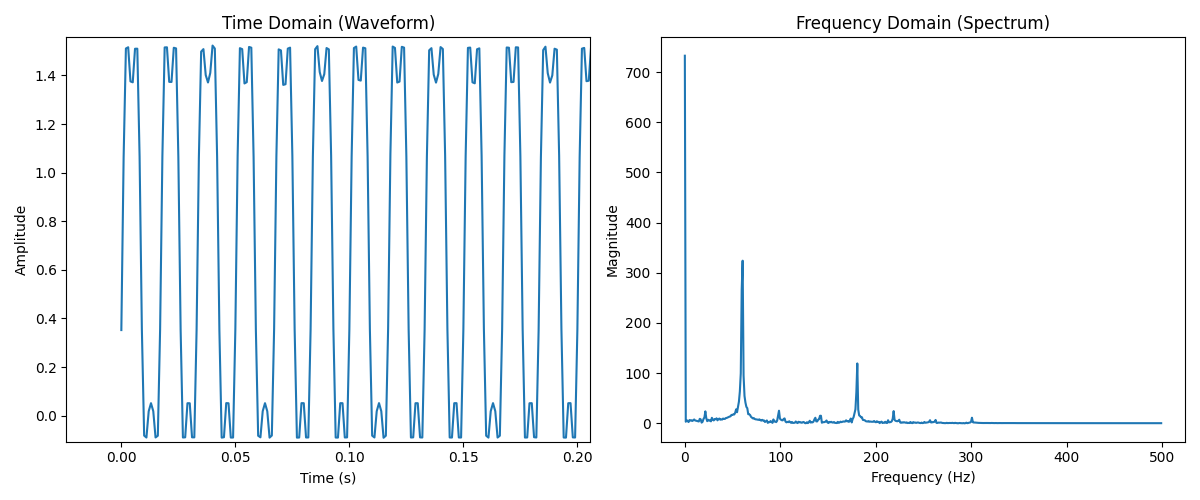
The input spectrum showed a series of harmonics, while the output spectrum revealed strong attenuation beyond the filter’s cutoff frequency (~250 Hz). The filtered output matches the theoretical frequency response plotted during the design phase, demonstrating that the implemented filter performs accurately under real-time conditions.



After filtering:



Filtered signal analysis:



As seen from both serial monitor and python analysis, the filter works properly and gives accurate and expected results.

## IIR Filter Design and Implementation

An Infinite Impulse Response (IIR) low-pass filter was designed using the Butterworth approximation method. The design was performed in Python using the scipy.signal.butter() function. The Butterworth design was chosen for its maximally flat passband and smooth response.

**Filter Type:** Low-Pass

**Design Method:** Butterworth

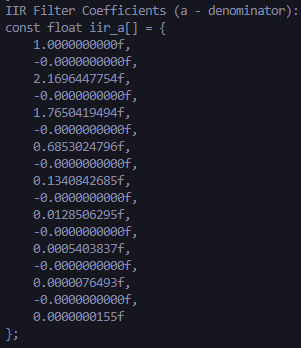
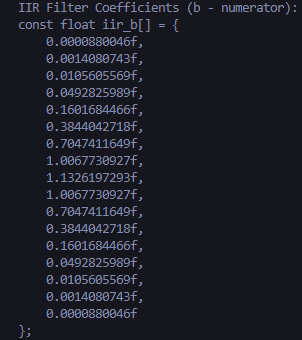
**Filter Order:** 16

**Sampling Frequency (fs):** 1000 Hz

**Cutoff Frequency (fc):** 250 Hz

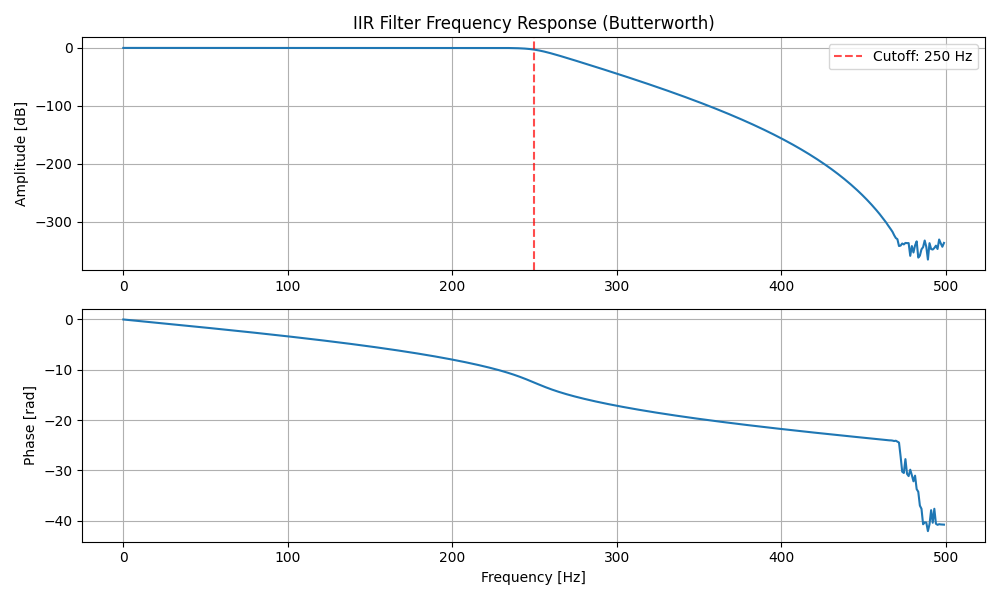
**Normalized Cutoff Frequency:** 0.5

The filter coefficients for implementation on the ESP32 are as follows:

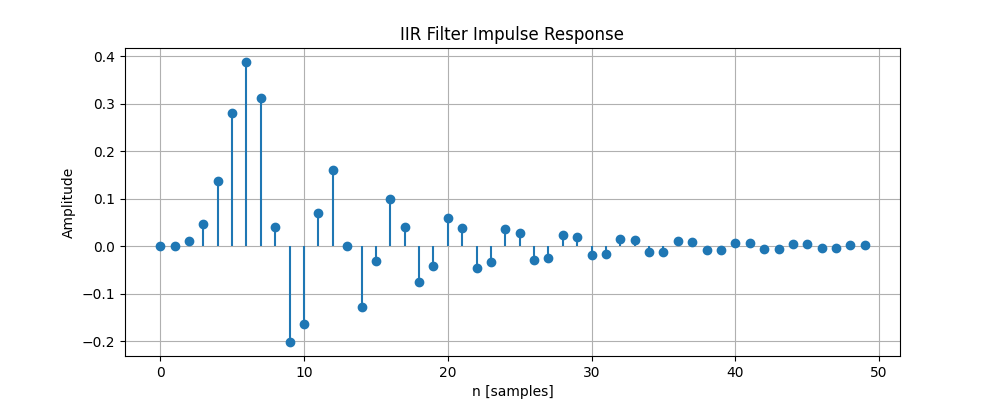


Filter Characteristics

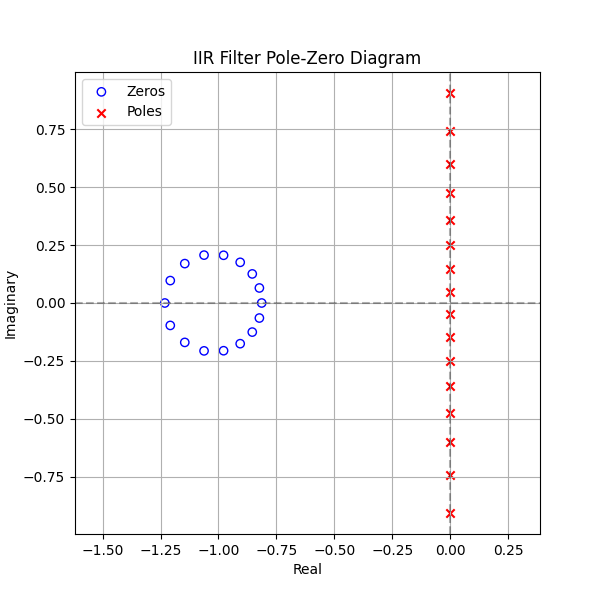
The IIR filter's frequency response confirms strong attenuation above the cutoff frequency of 250 Hz. The Butterworth design yields a smooth roll-off without ripples in the passband. The phase response, while non-linear, is continuous and typical for high-order IIR filters.



The impulse response shows an exponentially decaying oscillation, characteristic of a stable IIR filter.



The pole-zero plot shows the distribution of poles and zeros. Poles are lined up on 0 on real and distributed on imaginary while zeros circle the 0 imaginary and -1 real values.



Due to time constraints, the IIR filter was not implemented on the embedded system. However, all design steps, visual analyses, and coefficient generation for potential real-time use have been completed successfully. The filter is ready to be integrated into an RTOS task in a manner similar to the FIR implementation, using the printed coefficients.

## Conclusion

In this laboratory work, digital filters were designed and analyzed for implementation on an embedded real-time system using the ESP32-S3 microcontroller under FreeRTOS. Both Finite Impulse Response (FIR) and Infinite Impulse Response (IIR) low-pass filters were designed using Python.

The FIR filter was fully implemented on the embedded system and tested using ADC sampling and DAC output through PWM. The input signal—captured from a function generator—was processed in real time, and the filtered output was analyzed using both the serial monitor and Python-based spectrum analysis. Results clearly demonstrated that the FIR filter accurately attenuated high-frequency components while preserving the shape of the low-frequency signal, validating the correctness of both the design and its embedded implementation.

The IIR filter, based on a high-order Butterworth design, was analyzed in Python. Its frequency, phase, and impulse responses were visualized, and implementation-ready coefficients were generated. Although it was not implemented on the embedded platform in this lab session, all preparatory steps were completed, and the filter is ready for future integration.

This lab successfully demonstrated the end-to-end process of digital filter design, real-time embedded implementation, and performance evaluation. It provided practical experience with real-time data acquisition and signal processing using embedded systems and confirmed that Python is a powerful tool for digital filter prototyping and validation prior to deployment.