Tshwane University of Technology

We empower people

Department of Electrical Engineering

Course code: SPR307B

GA Project: Signal Generator and Analysis

Moeti KG

219407297

Table of Contents

Contents

1. Introduction

The objective of this project is to design, model, and simulate a multi-frequency signal generator. The generator is required to produce two types of waveforms (a 50% duty cycle square wave and a pure sine wave) at a constant 5V amplitude. Based on the last three non-zero digits of student number: 219407297, the three target frequencies are:

- F1: 2 × 1 = 2 Hz
- F2: 9 × 10= 90 Hz
- F3: 7 ×100 = 700 Hz

This report details the design and simulation of this system using a dual approach:

1. Hardware Simulation (Proteus): A circuit is designed using an ATmega328P microcontroller to generate three square waves on dedicated pins. These pins are then fed into three separate 3rd-order active Butterworth filters to produce the sine waves.
2. Mathematical Modelling (MATLAB): The system is modelled mathematically, utilizing a 6th-order Butterworth filter to synthesize the sine wave, followed by an analysis of the signal in the frequency domain and the effects of Additive White Gaussian Noise (AWGN).

2. Literature Review

2.1 Theoretical Foundations: Signal Generator

A signal generator, also known as a function generator or waveform generator, is a fundamental piece of electronic test equipment. Its primary purpose is to produce electronic signals, which are predictable, repeatable, and adjustable waveforms. These signals can vary in shape (e.g., sine, square, triangle), frequency, and amplitude.

Signal generators are used in engineering for:

- Testing and Debugging: A signal can be injected into a circuit's input to observe the output, verifying if the circuit is behaving as expected.
- Providing a Clock Source: Digital circuits often require a stable square wave to synchronize their operations.

2.2 Theoretical Foundations: Fourier Series

The link between a square wave and a sine wave is described by the Fourier Series, a mathematical theorem stating that any periodic waveform is a sum of fundamental and harmonic sine waves. A 50% duty cycle square wave with amplitude = A, can be represented as:

$$v(t) = \frac{4A}{\pi} + \sum_{n=1,3,5\ldots}^{\infty} \left(\frac{1}{\mathrm{n}} + \sin(\mathrm{n}w_0\mathrm{t})\right)$$

This equation shows the square wave is composed of a fundamental sine wave (f_0) and an infinite series of odd harmonics ($3f_0$, $5f_0$, $7f_0$,...).

This project is a practical application of this theorem. We build a square wave that has a lot of harmonics and then use a low-pass filter to remove the harmonics, proving that the fundamental sine wave was inside the square wave all along.

3. ECSA Graduate Attribute 2: Application of Scientific and Engineering Knowledge

This project applies a broad range of scientific and engineering principles to solve a defined engineering problem:

- **Application of Mathematics and Signal processing:** The Fourier Series is applied to a sine wave by isolating the fundamental frequency from a square wave.
- **Formal Analysis and Modelling:**
  - Proteus: Three 3rd-order active Butterworth filters are formally analysed. Their transfer functions are defined, and component values are calculated to achieve specific cut-off frequencies.
  - MATLAB: A 6th-order digital filter is designed using MATLAB and the frequency-domain characteristics of the signals are analysed using the Fast Fourier Transform (FFT).
- **Conceptualization of Systems:** The system is conceptualized as a series of functional blocks (selector, generator, filter), which are then implemented using appropriate components (microcontroller, op-amps, LCD).
- **Handling Uncertainty and risk:** The concept of signal integrity is explored by introducing Additive White Gaussian Noise (AWGN) to the MATLAB model. The effect of this uncertainty is analysed, and engineering solutions (such as band-pass filtering) are proposed to mitigate it.

4. Design 1: Proteus Hardware Simulation

4.1 System Overview

The Proteus circuit consists of an ATmega328P microcontroller and two push buttons (for frequency selection and signal selection). The microcontroller generates the 50% duty cycle square waves on three dedicated output pins:

- PB1 (Pin 15): 2 Hz
- PB2 (Pin 16): 90 Hz
- PB3 (Pin 17): 700 Hz
- To convert these to sine waves, three separate 3rd-order active filter circuits are used. The desired output pin (e.g., PB2 for 90 Hz) is connected to the input of its corresponding filter.

4.2 Microcontroller PWM Generation (C-Code)

The square waves are generated using the ATmega328P's 16-bit Timer1 in CTC (Clear Timer on Compare Match) mode. An interrupt (TIMER1_COMPA_vect) is triggered at twice the desired frequency. Inside this interrupt, the state of the correct output pin is manually toggled, producing a perfect 50% duty cycle.

The OCR1A value required to achieve a specific frequency f_{out}, is calculated by:

$$f_{interrupt} = 2\ f_{out}$$

$$\text{OCR1A} = \frac{f_{CPU}}{\text{Pre−scaler} \times f_{interrupt}} - 1 \ = = \ \frac{f_{CPU}}{\text{Pre−scaler} \times 2\ f_{out}} - 1$$

Calculation for F1 = 2 Hz:

- Pre-scaler = 1024
- $\text{OCR1A} = \frac{16\ 000\ 000}{1024 \times 2 \times 2\text{Hz}}$ = 3906.25
- Code value = 3905

Calculation for F2 = 90 Hz:

- Pre-scaler = 256
- $\text{OCR1A} = \frac{16\ 000\ 000}{256 \times 2 \times 90\text{Hz}}$ = 347.22
- Code value = 346

Calculation for F3 = 700 Hz:

- Pre-scaler = 64
- $\text{OCR1A} = \frac{16\ 000\ 000}{64 \times 2 \times 700\text{Hz}}$ = 178.57
- Code value = 177

4.3 C-Code Implementation

The following code is loaded onto the ATmega328P. It manages the button input, and generates the three-square waves on their dedicated pins:

Link to full C-code:
https://drive.google.com/file/d/1naUbVjIQ195h9J0LCBaNYjIzBgrIXYMr/view?usp=sharing

4.4 3rd-Order Active Filter Design

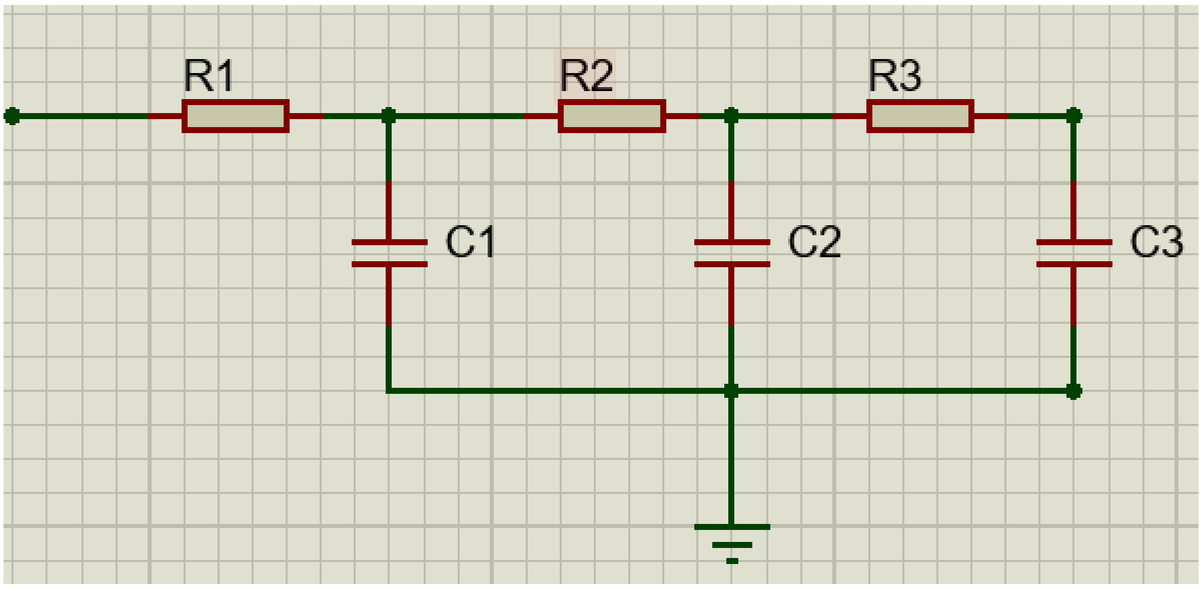


Figure 1: passive 3rd order Butterworth filter

This design uses three separate 3rd-order active filters. This design is better than a passive filter in this project because it uses op-amps to buffer the signal (preventing loading) and is built only from resistors and capacitors.

The cut-off frequency f_c for a 3rd-order active filter with identical R and C values is given by:

$$f_c = \frac{1}{2\pi RC}$$

Filter 1: For 2 Hz Signal (F1):

- The filter has to pass 2 Hz and block the 3rd harmonic ($n \times f_1 = 3 \times 2Hz = 6Hz$)

$$f_c = 1.25 f_1 = 1.25 \times 2Hz = 2.5 \text{ Hz}$$

Choose: R1 = R2 = R3 = 1 kΩ

$$\text{C1} = \text{C2} = \text{C3} = \frac{1}{2\pi R f_c} = \frac{1}{2\pi(1\,000)(2.5)} = 63.66\,uF$$

Filter 2: For 90 Hz Signal (F2):

- The filter has to pass 90 Hz and block the 3rd harmonic ($n \times f_2 = 3 \times 90Hz = 270Hz$)

$$f_c = 1.25 f_1 = 1.25 \times 90Hz = 112.5\text{ Hz}$$

Choose: R1 = R2 = R3 = 10 kΩ

$$\text{C1} = \text{C2} = \text{C3} = \frac{1}{2\pi R f_c} = \frac{1}{2\pi(10\,000)(112.5)} = 141.5\,nF$$

Filter 3: For 700 Hz Signal (F3):

- The filter has to pass 700 Hz and block the 3rd harmonic ($n \times f_2 = 3 \times 700Hz = 2100Hz$)

$$f_c = 1.25 f_1 = 1.25 \times 700Hz = 875\text{ Hz}$$

Choose: R1 = R2 = R3 = 10 kΩ

$$\text{C1} = \text{C2} = \text{C3} = \frac{1}{2\pi R f_c} = \frac{1}{2\pi(10\,000)(875)} = 18.19\,nF$$

4.5 Transfer Function Calculations for 3rd-Order Active Filters

Total 3rd-Order Transfer Function: The total transfer function H(s) is the product of the two sections.

$$H(s) = H1(s) \times H2(s) = \frac{1}{(1 + sRC)(s^2R^2C^2 + 2sRC + 1)}$$

Filter 1: 2 Hz Signal (fc = 2.5 Hz)

Time Constant (T1): $T1 = RC = (1\,000\Omega) \times (63.66 \times 10^{-6}\,F) = 0.06366\,s$

- Transfer Function H1(s): $\boldsymbol{H1(s) = \frac{1}{(1 + 0.06366s)(0.004405s^2 + 0.1273s + 1)}}$

Filter 2: 90 Hz Signal (fc = 112.5 Hz)

Time Constant (T1): $T1 = RC = (10\,000\Omega) \times (141.5 \times 10^{-9}\,F) = 0.001415\,s$

- Transfer Function H1(s): $\boldsymbol{H1(s)} = \dfrac{\mathbf{1}}{(\mathbf{1+0.001415}\,s)(\mathbf{2.002{\times}10^{-6}}s^{2}\mathbf{+0.00283}s\mathbf{+1})}$

Filter 3: 700 Hz Signal (fc = 875 Hz)

Time Constant (T1): $T1 = RC = (10\,000\Omega) \times (18.19 \times 10^{-9}\,F) = 0.0001819\,s$

- Transfer Function H1(s): $H1(s) = \dfrac{1}{(1+0.0001819\,s)(3.309{\times}10^{-8}s^{2}+0.0003638s+1)}$

4.5 Proteus Circuit Diagram

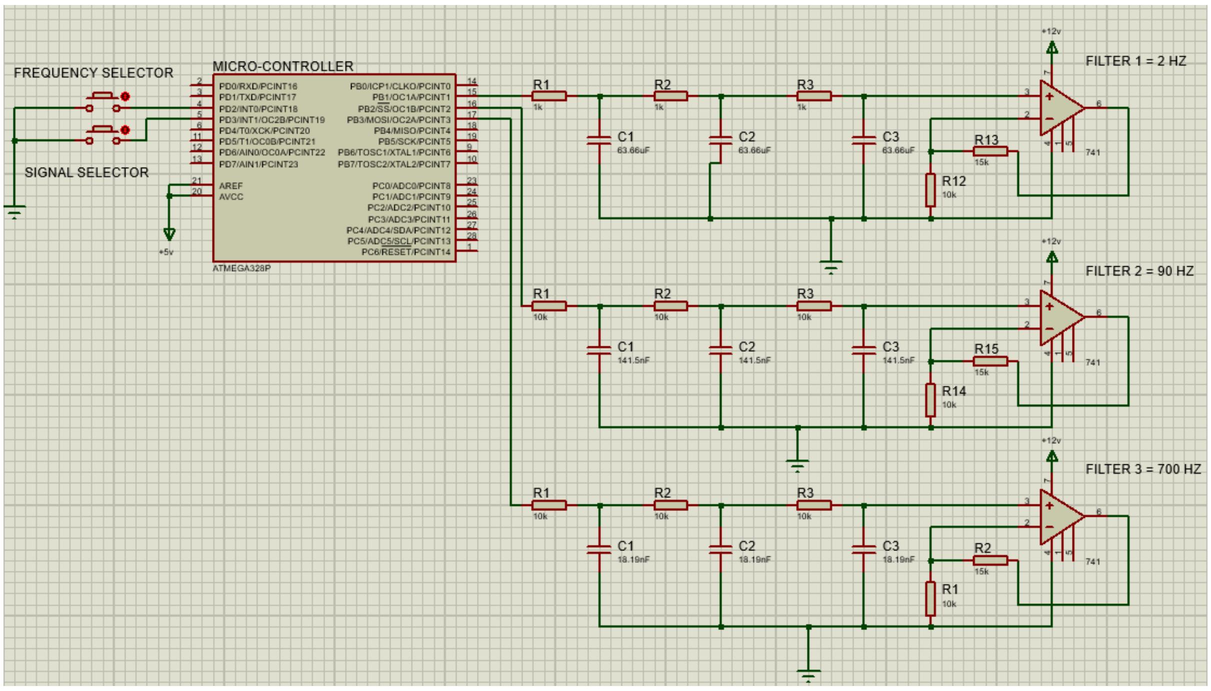


Figure 2: Proteus full signal generator circuit

Link to full circuit:
https://drive.google.com/file/d/1j9MFlEbqhombrxnsv0c70apmBJqqHKj5/view?usp=sharing

4.6 Results 1: Proteus Simulation

Simulation for F1 = 2 Hz:

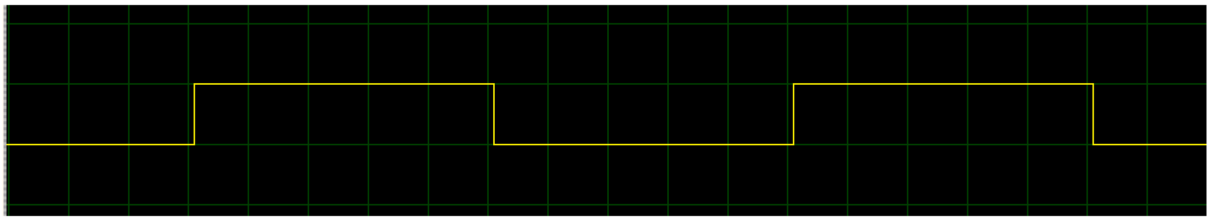


Figure 3: Square wave generated at F1 = 2 Hz

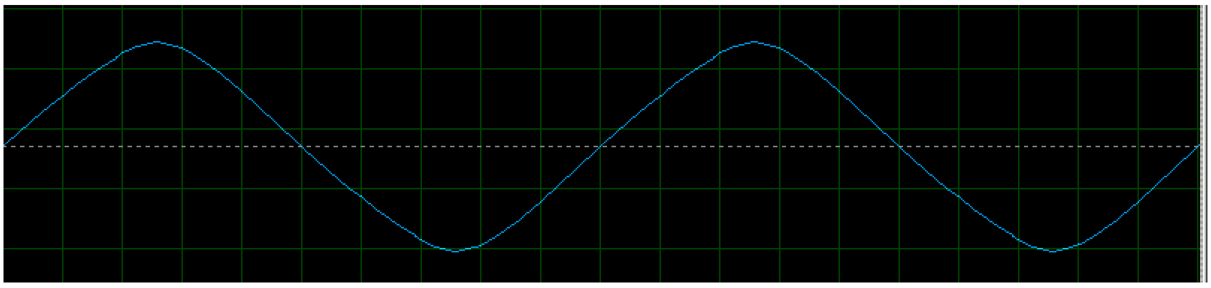


Figure 3: Sine wave generated at F1 = 2 Hz

Simulation for F2 = 90 Hz:

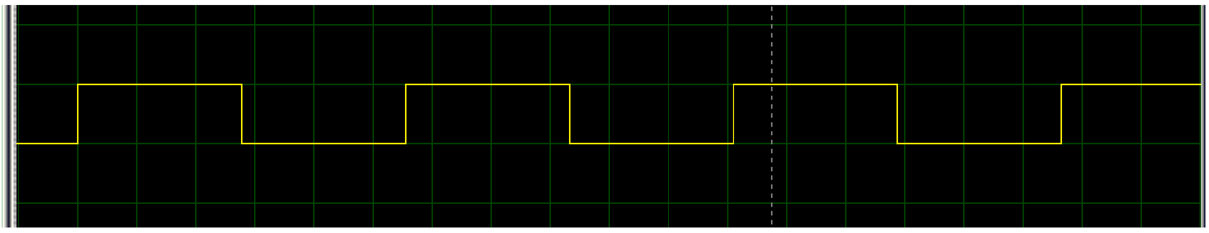


Figure 4: Square wave generated at F2 = 90 Hz

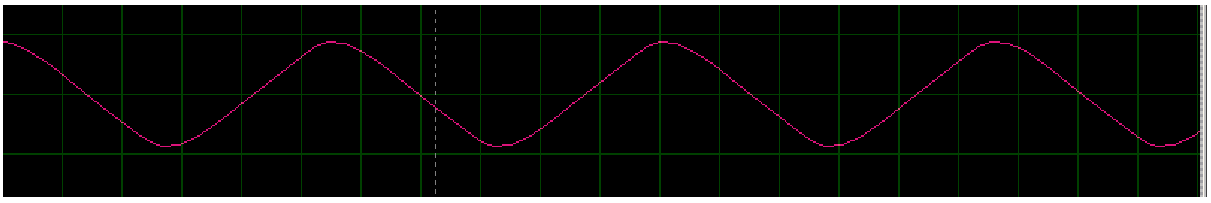


Figure 5: Sine wave generated at F2 = 90 Hz

Simulation for F3 = 700 Hz:

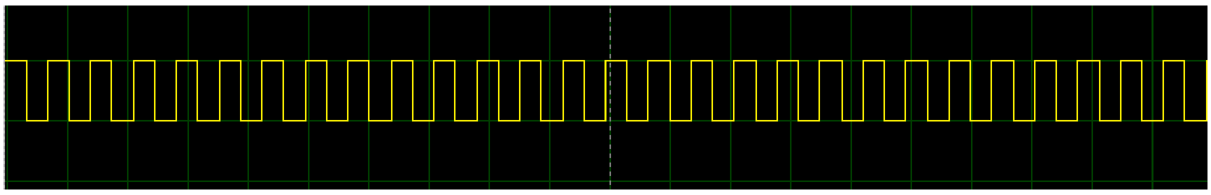


Figure 6: Sine wave generated at F3 = 700 Hz

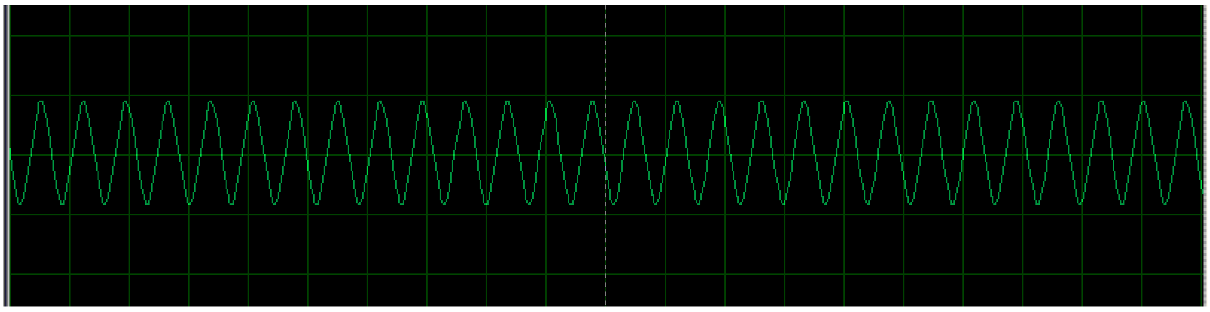


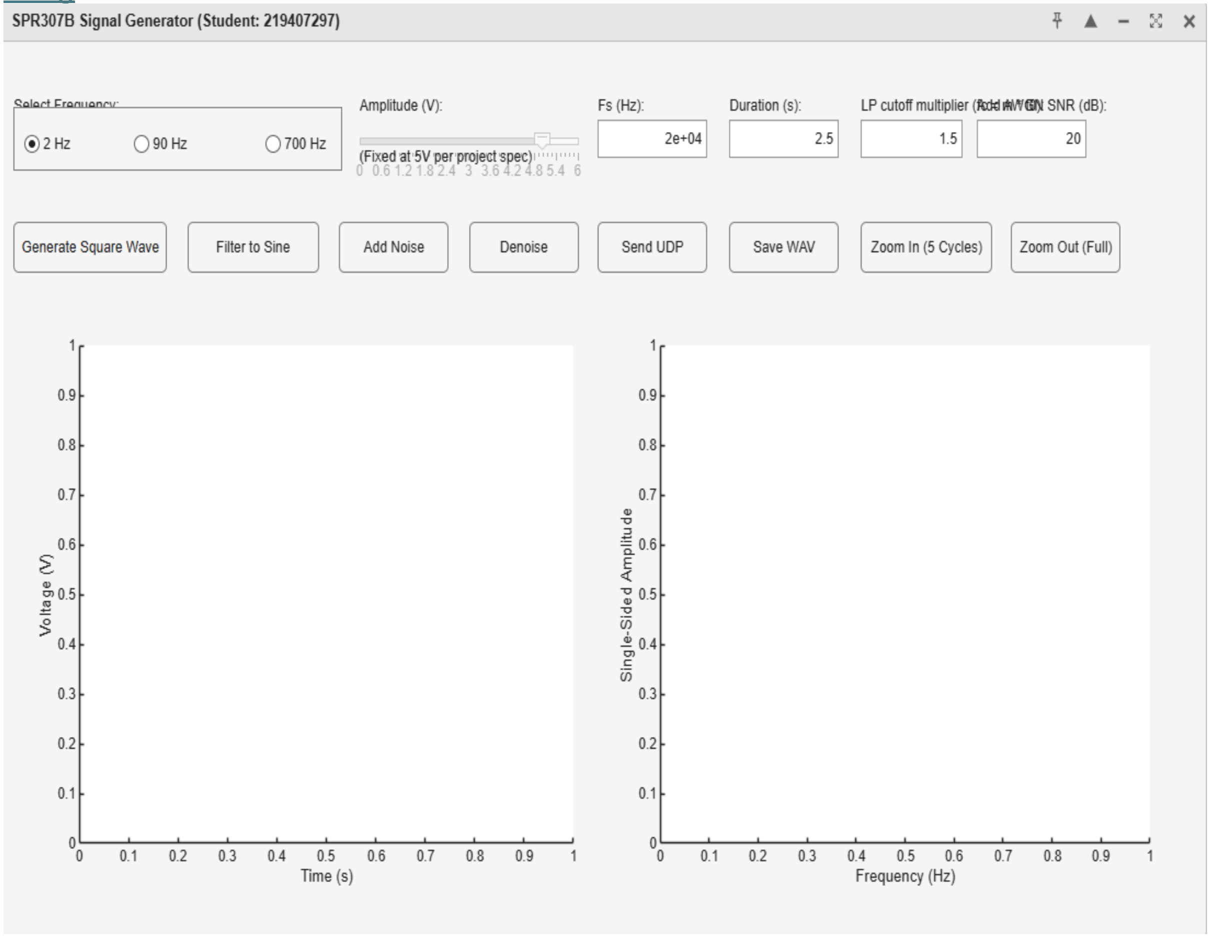
Figure 7: Sine wave generated at F3 = 700 Hz

5. Design 2: MATLAB Modelling

5.1 System Overview

The system was implemented as a MATLAB function signal_generator_app which creates a GUI.

Link to signal_generator_app: https://drive.google.com/file/d/1naUbVjIQ195h9J0LCBaNYjIzBgrIXYMr/view?usp=sharing



SPR307B Signal Generator (Student: 219407297)

Select Frequency: [✓ 2 Hz] 90 Hz 700 Hz

Amplitude (V): (Fixed at 5V per project spec)

Fs (Hz): 2e+04

Duration (s): 2.5

LP cutoff multiplier (fc = k·f0): 1.5

Add AWGN SNR (dB): 20

Figure 8: MATLAB GUI system

5.2. System Parameters and Frequency Selection

The project requirements are defined as global parameters:

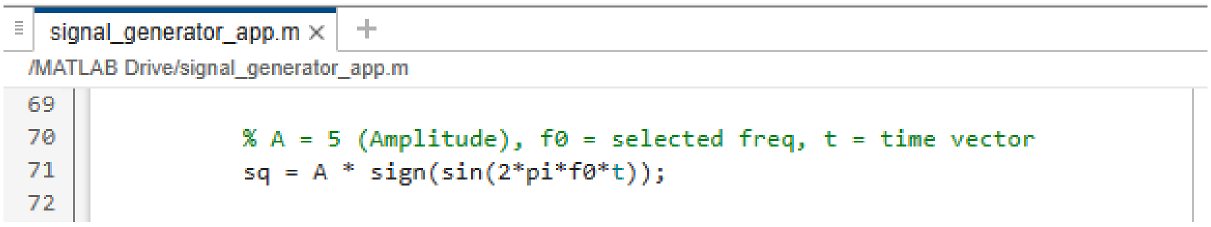
- frequencies = [2, 90, 700], From student number 219407297.
- Fs = 20 000 Hz, sampling frequency.
- duration = 2.5, A default signal duration of 2.5s to capture 5 cycles so that the results are visible to the human eye for interpretation.
- The amplitude is fixed at 5V using a disabled slider.

The user selects the frequency f_0 using a radio button group (Select Frequency) on GUI.

5.2. Signal Generation (onGenerate function)

This function is called when the "Generate" button is pressed. It first checks which signal type is selected.

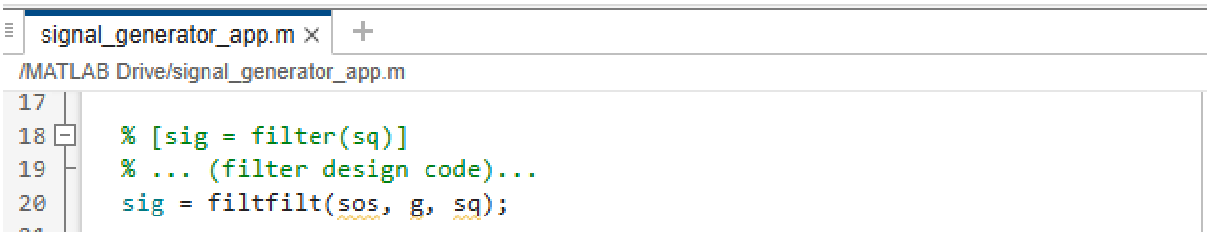
Square Wave Generation: As per the project constraints, no built-in square function is used. Instead, the square wave is generated mathematically from the sin function:



```
        % A = 5 (Amplitude), f0 = selected freq, t = time vector
        sq = A * sign(sin(2*pi*f0*t));
```

This code works by first generating a standard sine wave, then feeding it into the sign() function. sign() returns +1 for all positive inputs and -1 for all negative inputs, converting the smooth sine wave into a perfect square wave.

Sine Wave Generation: If "Sine (filtered)" is selected, the code first generates the square wave as above and *then* applies the 6th-order filter to it.



```
% [sig = filter(sq)]
% ... (filter design code)...
sig = filtfilt(sos, g, sq);
```

The filter design is the critical step, detailed in the next section. We use filtfilt (zero-phase filtering) to apply the filter. This processes the signal forwards and then backwards, which cleverly cancels out any time delay (phase shift) that a normal filter would introduce.

5.3. 6th-Order Filter: Design and Calculations

The project requires a 6th-order Butterworth filter. We use the allowed butter function. The challenge is to provide the correct inputs: n and W_c.

- n = 6 (The order is given)
- W_c is the normalized cutoff frequency, calculated as: $W_c = \frac{F_c}{F_s \div 2}$

We must calculate the specific F_c (cut-off frequency) for each case.

Case 1: F_0 = 90 Hz

- The filter has to pass 90 Hz and block the 3rd harmonic ($n \times f_2 = 3 \times 90Hz = 270Hz$)
- F_s **:** 20 000 Hz (Nyquist = 10 000 Hz)

- Design Choice: Use multiplier m = 1.5 (from the UI).
- F_c Calculation: $F_c = 1.5 \times 90\ Hz = 135\ Hz$.
- W_c Calculation: $W_c = 135 \div 10000 = 0.0135$.
- MATLAB Command: $[sos, g] = butter(6, 0.0135, 'low');$

Case 2: F_0 =700 Hz

- The filter has to pass 700 Hz and block the 3rd harmonic ($n \times f_2 = 3 \times 700Hz = 2\,100Hz$)
- F_s **:** 20 000 Hz (Nyquist = 10 000 Hz)
- Design Choice: Use multiplier m = 1.5 (from the UI).
- F_c Calculation: $F_c = 1.5 \times 700\ Hz = 1050\ Hz$.
- W_c Calculation: $W_c = 1050 \div 10000 = 0.1050$.
- MATLAB Command: $[sos, g] = butter(6, 0.1050, 'low');$

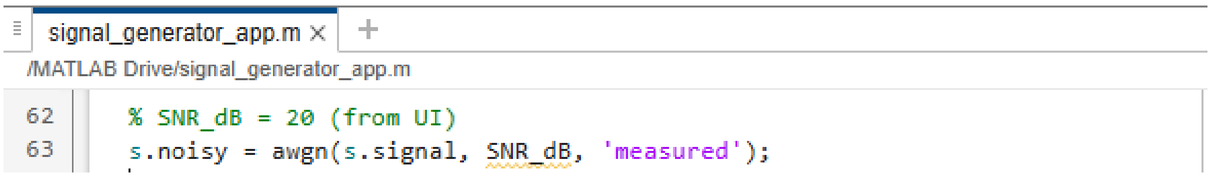
Case 3: F_0 = 2 Hz (Special Stability Case)

- Problem: A W_c calculated at F_s = 20 000 Hz (e.g., W_c = 5 ÷10 000 = 0.0005) is too small and causes numerical instability errors ("Matrix is close to singular").
- Solution: A multi-rate processing approach is used.
  1. Resample Down: The 20 000 Hz signal is resampled to a temporary, lower sampling rate: F_s = 2 000 Hz. $sq\_low = resample(sq, Fs\_low, Fs);$
  2. Design Filter at F_s (low)**:** We now design the filter for this new Nyquist (1 000 Hz).
     - The filter has to pass 2 Hz and block the 3rd harmonic ($n \times f_2 = 3 \times 2Hz = 6Hz$)
     - Design Choice: Use multiplier m = 2.5
     - F_c Calculation: $F_c = 2.5 \times 2\ Hz = 5\ Hz$.
     - W_c Calculation: $W_c = 5 \div 1\,000 = 0.005$.
     - MATLAB Command: $[sos, g] = butter(6, 0.005, 'low');$
  3. Apply Filter: The filter is applied to the low-sample-rate signal. $sig\_low = filtfilt(sos, g, sq\_low);$
  4. Resample Up: The filtered, clean signal is resampled back to the original 20 000 Hz for plotting. $sig = resample(sig\_low, Fs, Fs\_low);$

This design ensures stability while still meeting the 6th-order Butterworth requirement.

5.4. Noise Simulation (onAddNoise function)

This function simply adds AWGN to the currently generated clean signal.



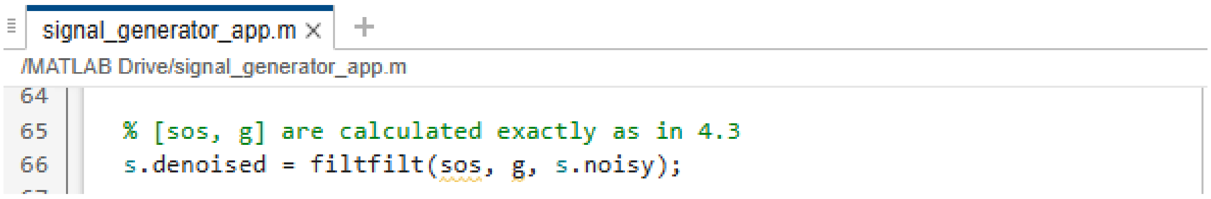
signal_generator_app.m

/MATLAB Drive/signal_generator_app.m

```
% SNR_dB = 20 (from UI)
s.noisy = awgn(s.signal, SNR_dB, 'measured');
```

5.5. Denoise Function (onDenoise function)

The "Denoise" button applies the *exact same filter* that was designed in section 4.3. It runs the same logic: it takes the noisy signal, applies the correct 6th-order filter for the selected frequency (including the special 2 Hz multi-rate fix), and outputs the denoised signal.

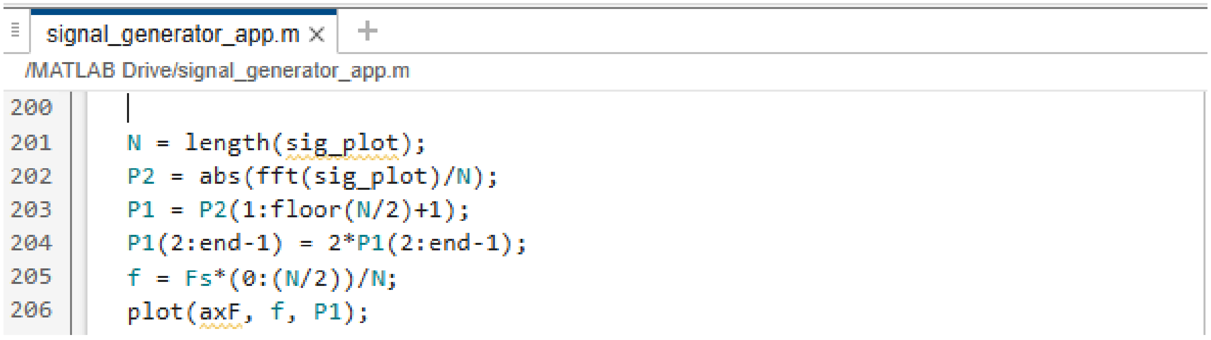


```
% [sos, g] are calculated exactly as in 4.3
s.denoised = filtfilt(sos, g, s.noisy);
```

5.6. Plotting and Analysis (plotAll function)

This function is responsible for creating the graphs.

- Time Domain: It plots the full 2.5-second signal. The "Zoom In" button simply changes the x-axis limits of this plot to [0, $5/F_0$].
- **Frequency Domain (FFT):**



```

N = length(sig_plot);
P2 = abs(fft(sig_plot)/N);
P1 = P2(1:floor(N/2)+1);
P1(2:end-1) = 2*P1(2:end-1);
f = Fs*(0:(N/2))/N;
plot(axF, f, P1);
```

- It implements **dynamic FFT zooming** to show the most relevant information:
- If "Clean": Zooms in on the harmonics (e.g., 0-30 Hz for F_0=2Hz)
- If "Noisy" or "Denoised": Zooms to a specific "action area" to show the noise floor being added and removed.
  1. 2 Hz: $xlim(axF, [0, 20]);$
  2. 90 Hz: $xlim(axF, [0, 450]);$
  3. 700 Hz: $xlim(axF, [0, 3500]);$

5.7. Simulation Results and Discussion

5..7.1. Analysis of 90 Hz Signal

- **Clean Square Wave:**

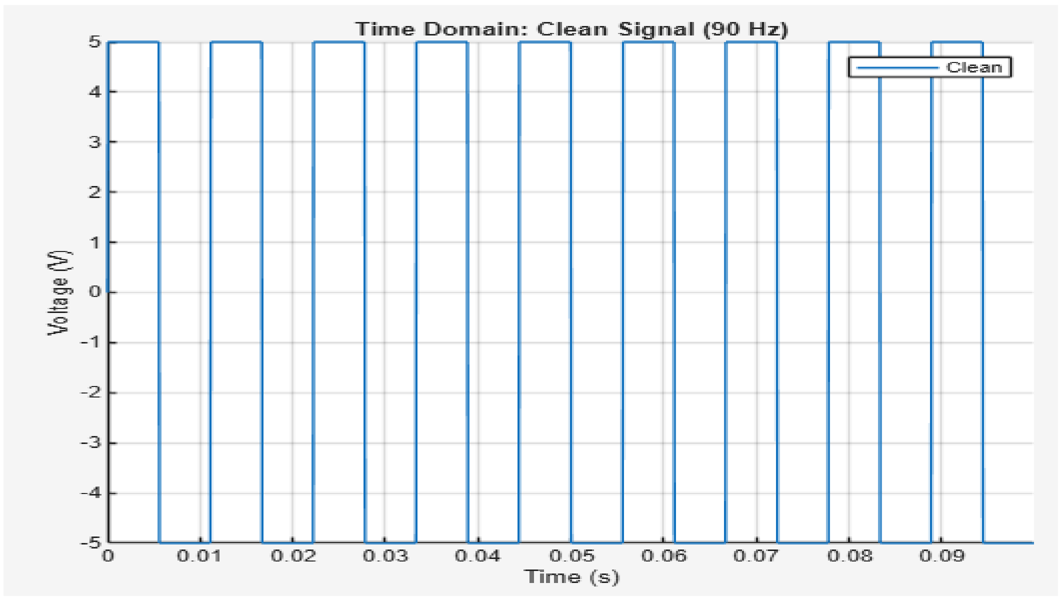


Figure 9: Generated square wave at F = 90Hz

Time Domain: A perfect 5V square wave with a period of $T = \frac{1}{90} = 11.1ms = 0.01s$.

FFT:

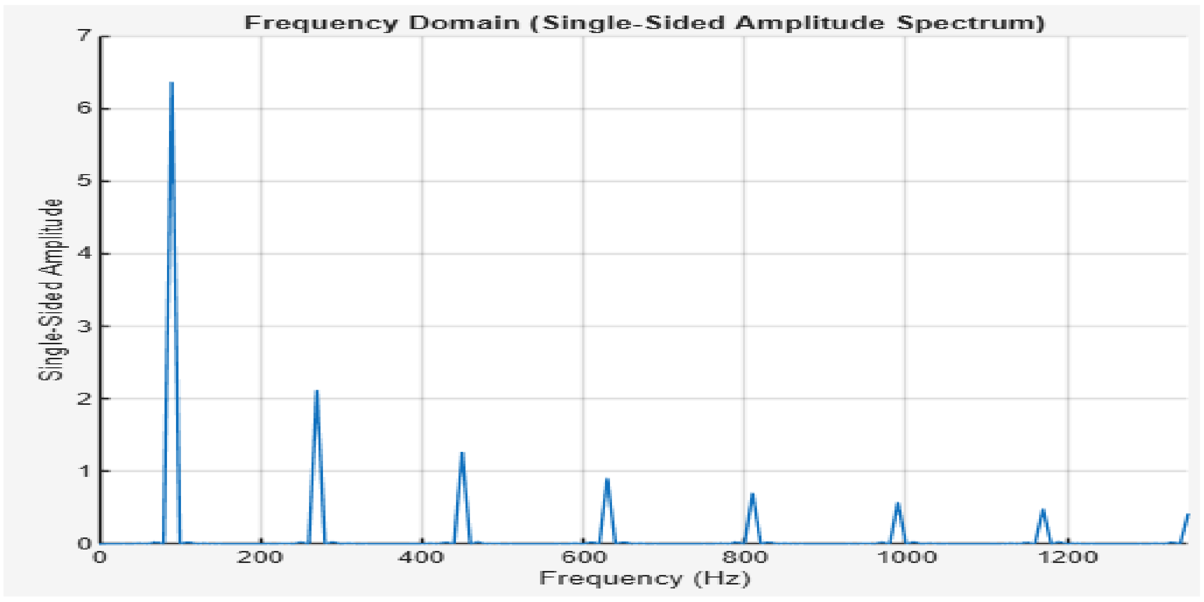


Figure 10: Frequency Domain representation of the square wave

A large spike at the 90 Hz fundamental, with subsequent, smaller spikes at the odd harmonics: 270 Hz, 450 Hz, etc. This perfectly matches the Fourier series theory.

- **Clean Sine Wave:**

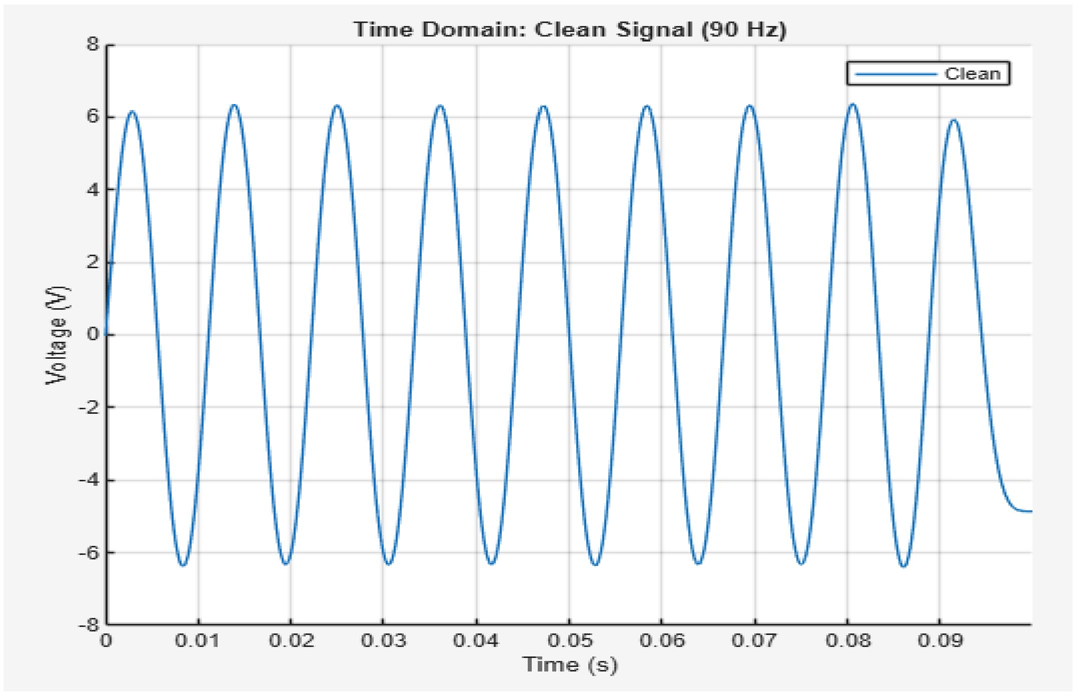


Figure 11: Filtered sine wave at F = 90Hz

Time Domain: A smooth filtered sine wave.

FFT:

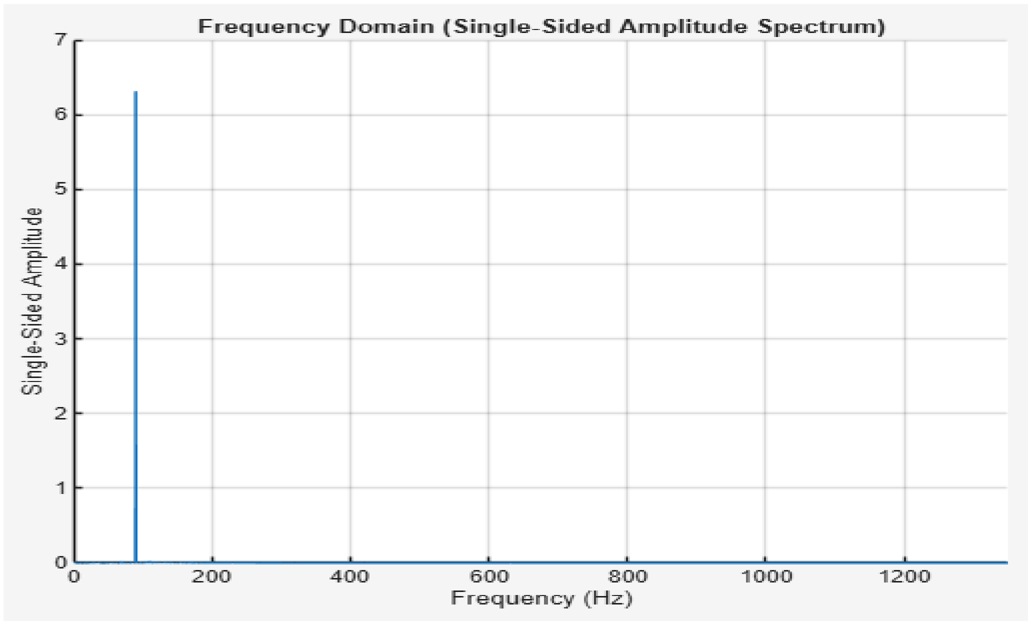


Figure 12: Frequency Domain representation of the filtered sine wave

A single, sharp spike at 90 Hz. The harmonics at 270 Hz, 450 Hz, etc… are gone. This proves the 6th-order filter (with F_c = 135 Hz) successfully removed the harmonics.

- **Noisy Sine Wave:**

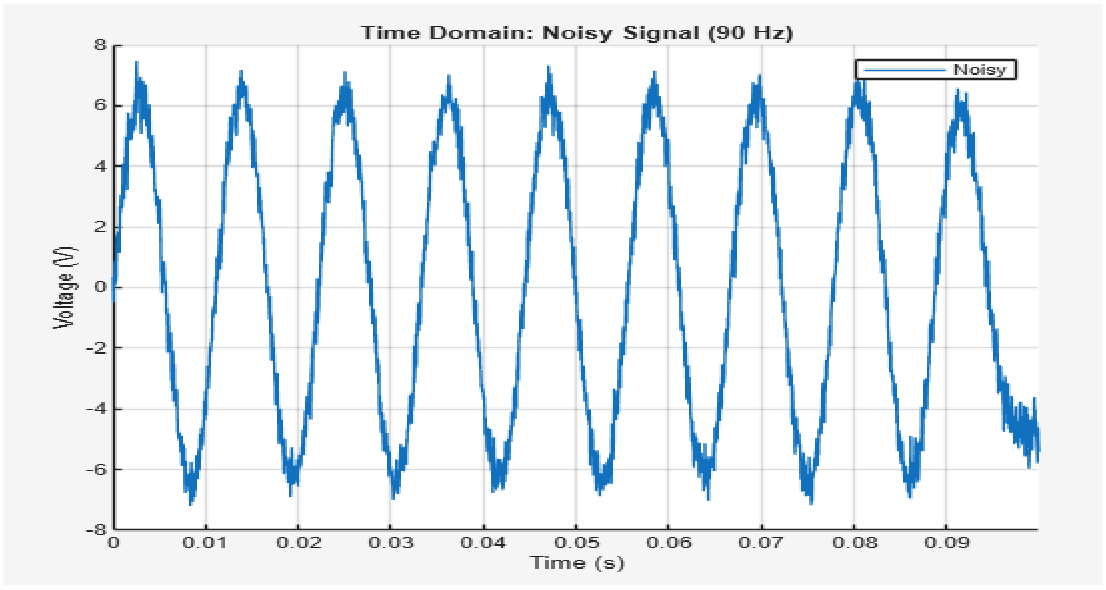


Figure 13: Noisy sine wave at F = 90Hz

Time Domain: The sine wave appears "fuzzy" and "hairy".

FFT:

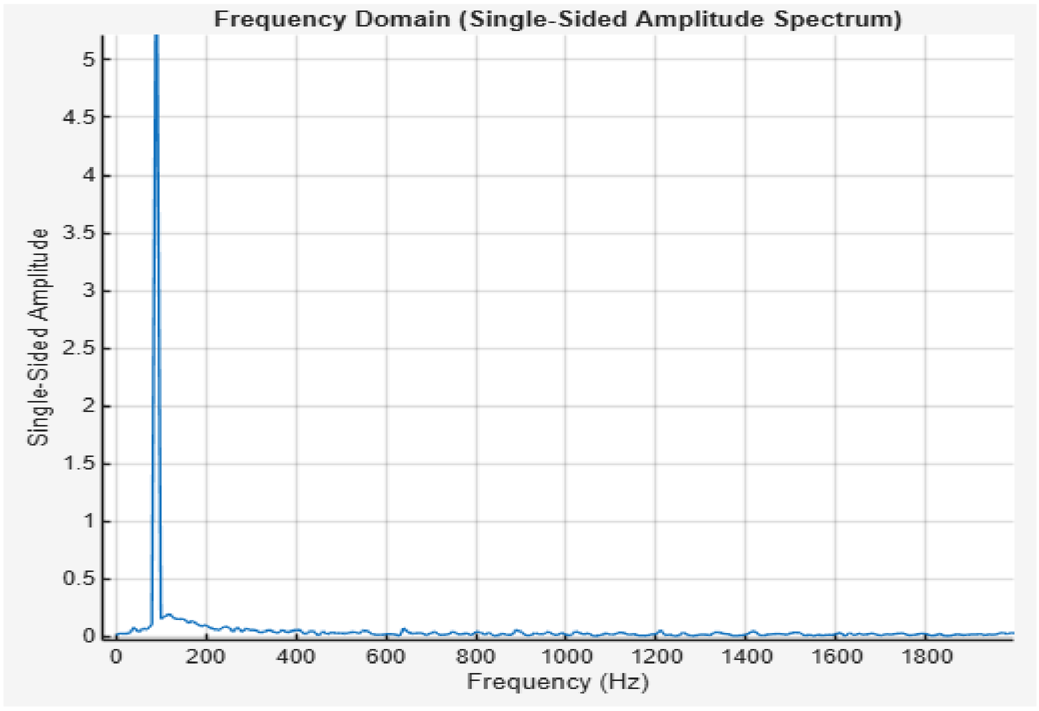


Figure 14: Frequency Domain representation of the Noisy sine wave

The 90 Hz spike is still visible, but the entire baseline (noise floor) in the 0-450 Hz view is visibly raised, showing broadband noise.

- **Denoised Sine Wave:**

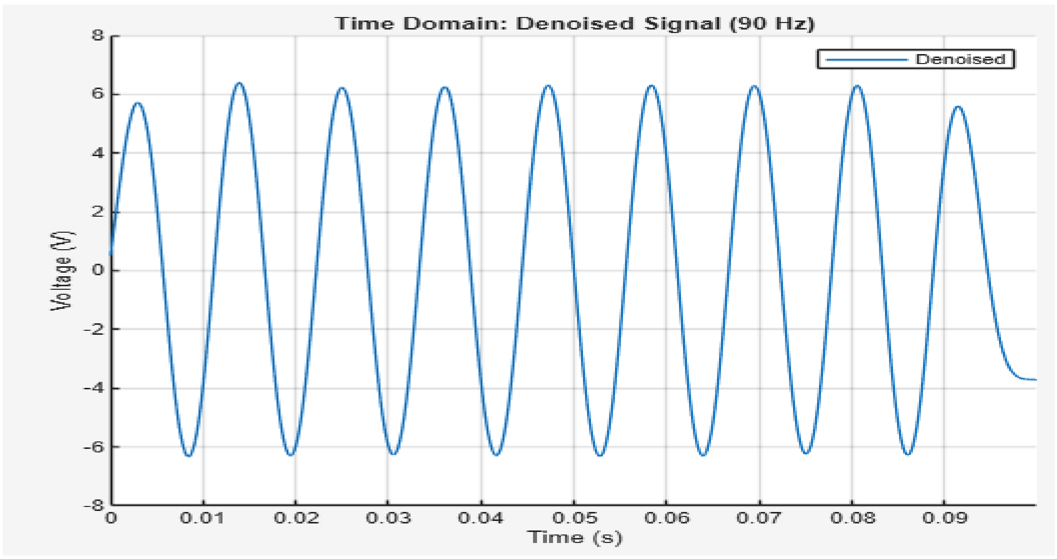


Figure 15: De-Noised sine wave at F = 90Hz

Time Domain: The "fuzz" is gone, and the wave is smooth again.

FFT:

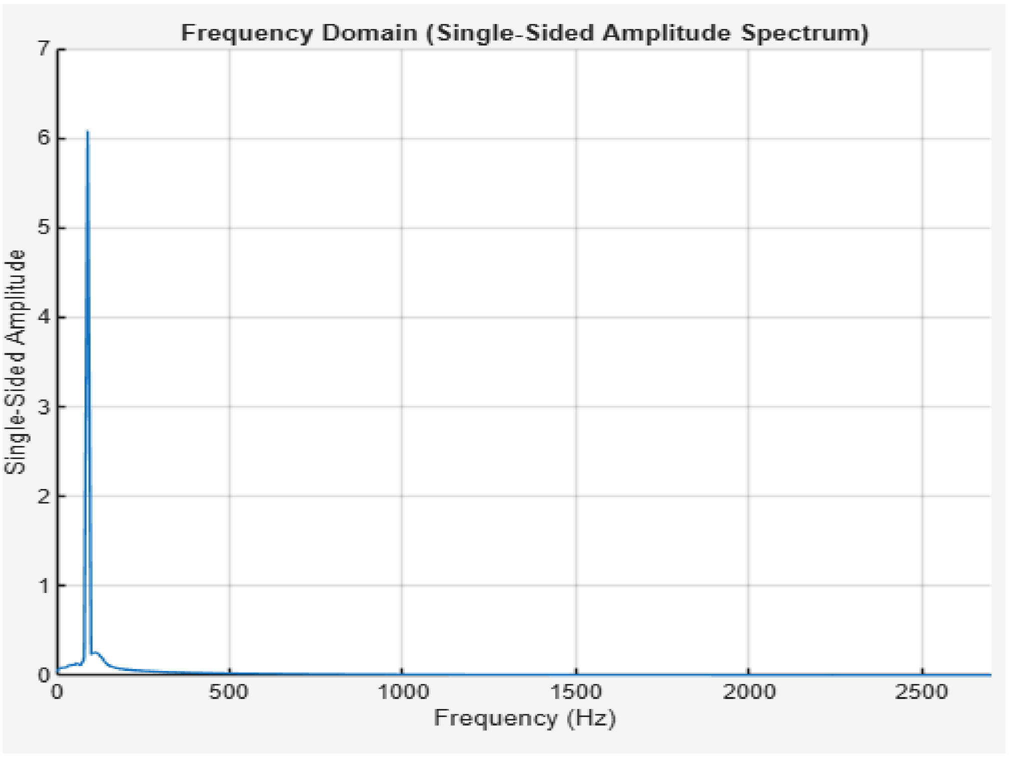


Figure 16: Frequency Domain representation of the De-Noised sine wave

The noise floor drops back to zero, proving the filter successfully removed the high-frequency noise while preserving the 90 Hz signal.

5.7.2. Analysis of 700 Hz Signal

- **Clean Square Wave:**

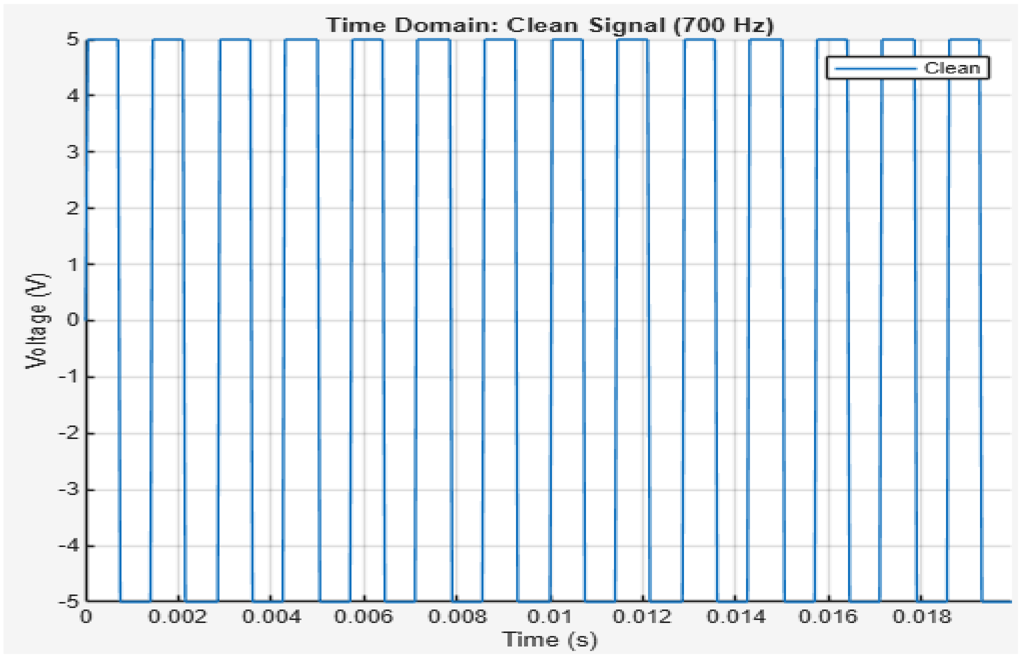


Figure 17: Generated square wave at F = 700Hz

Time Domain: A very fast 5V square wave ($T = 1.43\ ms$). Appears as a solid block when zoomed out.

FFT:

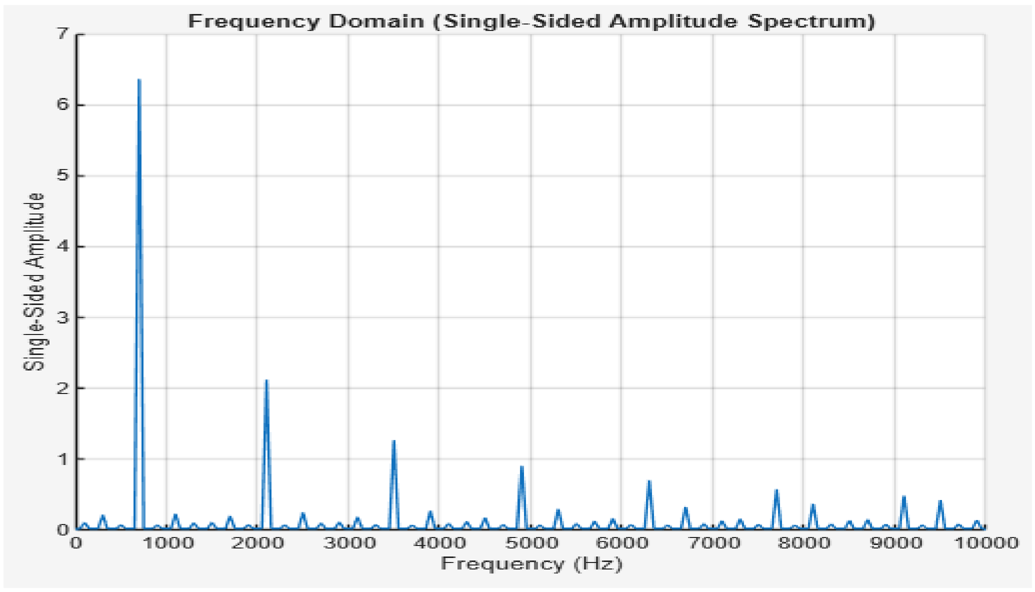


Figure 18: Frequency Domain representation of the square wave

Spikes are seen at 700 Hz, 2100 Hz, and 3500 Hz etc.…

- **Clean Sine Wave:**

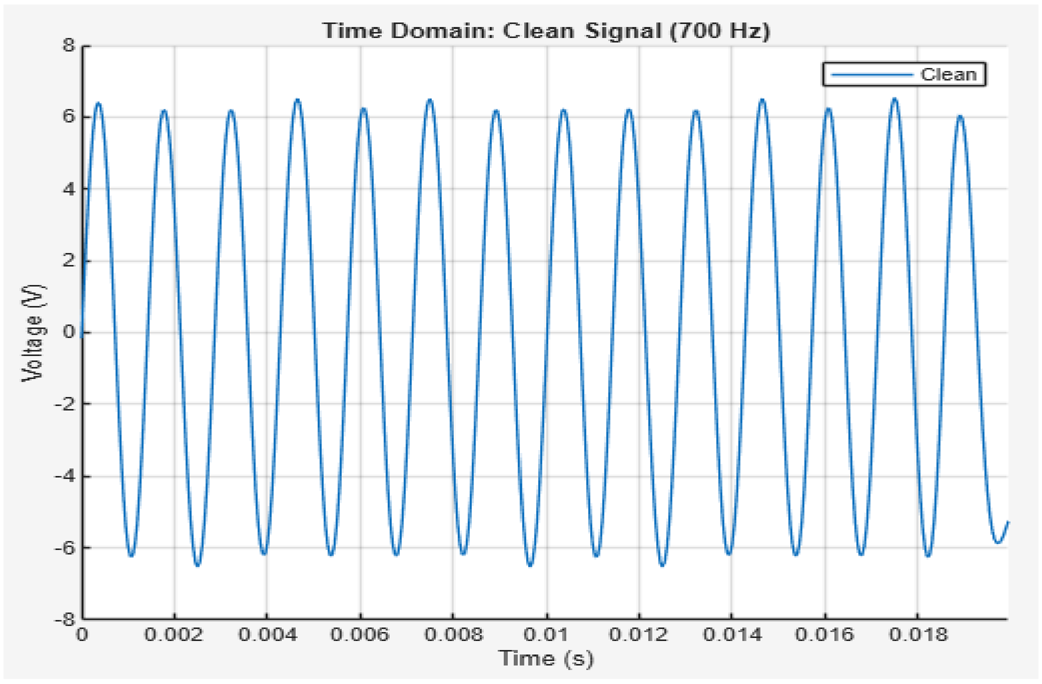


Figure 19: Filtered sine wave at F = 700Hz

Time Domain: A smooth 700 Hz sine wave.

FFT:

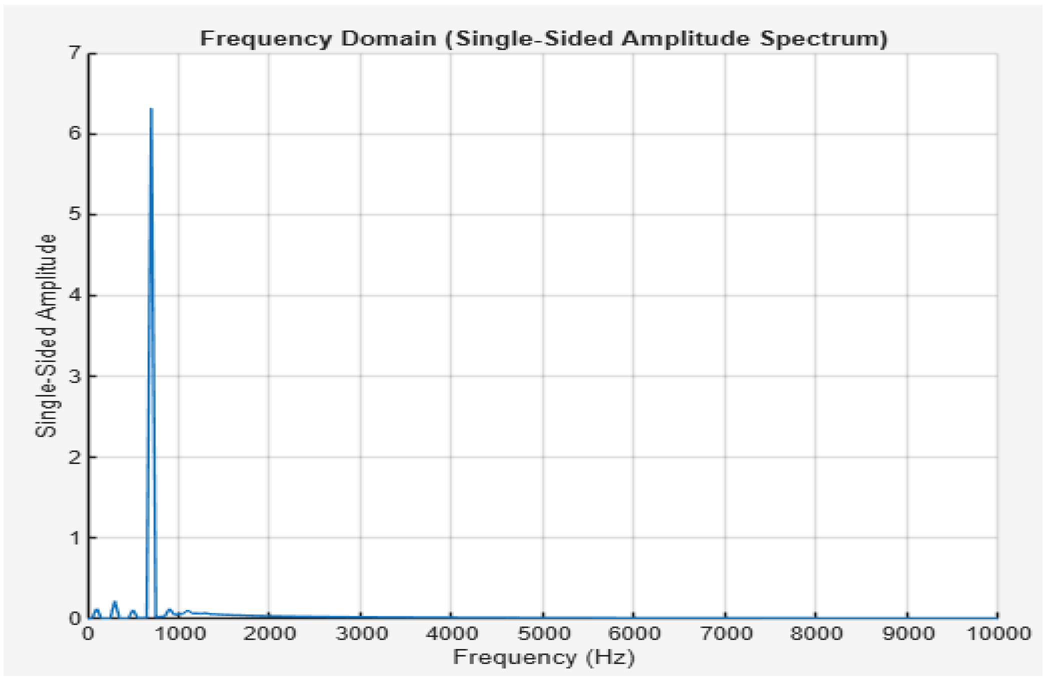


Figure 20: Frequency Domain representation of the filtered sine wave

A single spike at 700 Hz. The filter (with $f\_c = 1050\ Hz$) successfully removed the 2100 Hz harmonic.

- **Noisy / Denoised:**

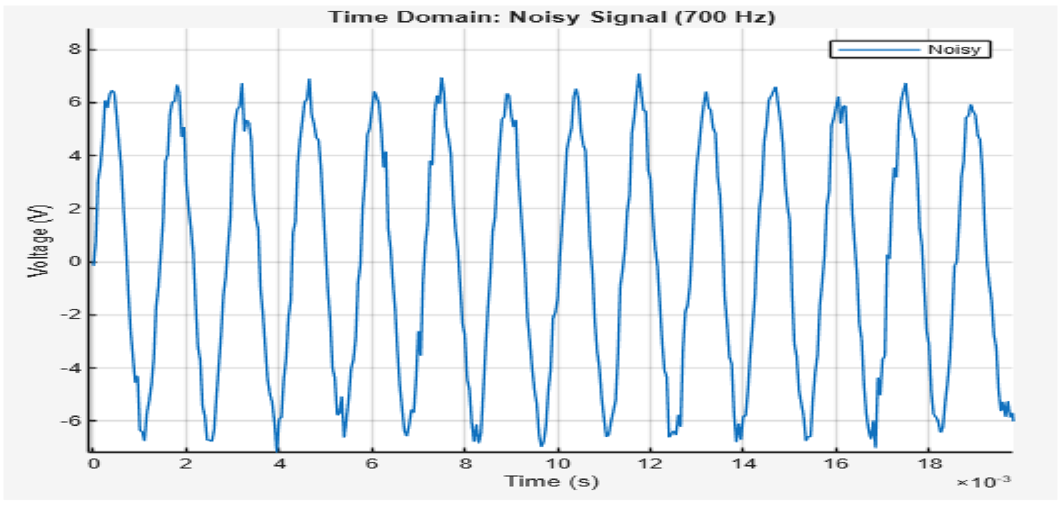


Figure 21: Noisy sine wave at F = 700Hz

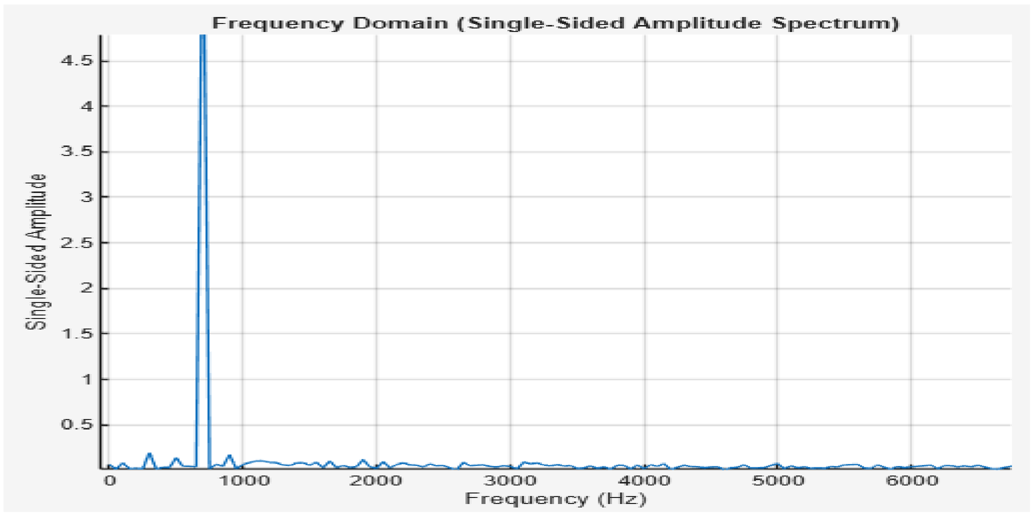


Figure 22: Frequency Domain representation of the Noisy sine wave

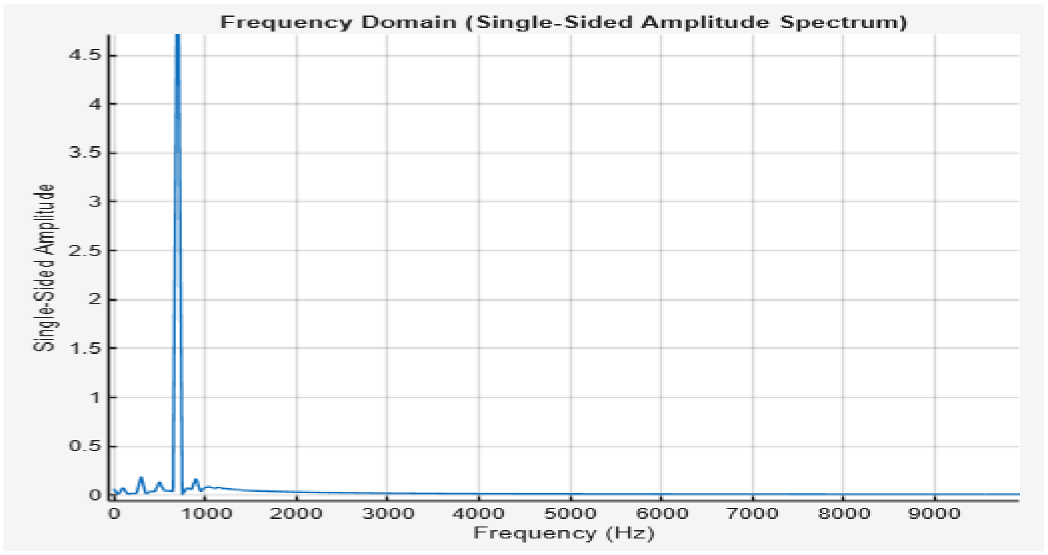


Figure 23: Frequency Domain representation of the De-noised sine wave

As with the 90 Hz signal, the 0-3500 Hz FFT view clearly shows the noise floor appearing when noise is added and disappearing when the filter is re-applied.

5.7.3. Analysis of 2 Hz Signal

- **Clean Square Wave:**

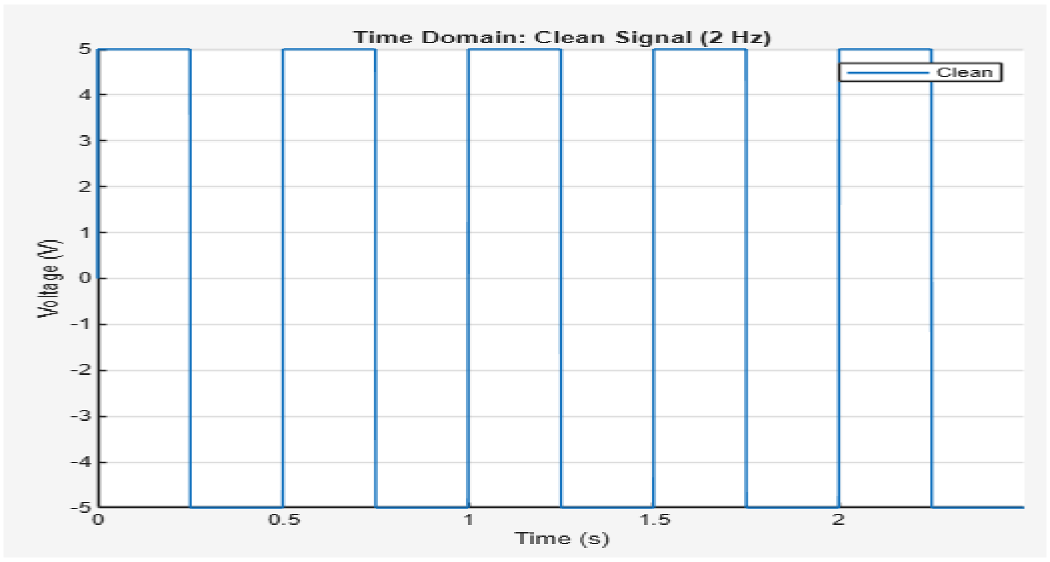


Figure 24: Generated square wave at F = 2 Hz

Time Domain: A slow 5V square wave with a 0.5s period.

FFT:

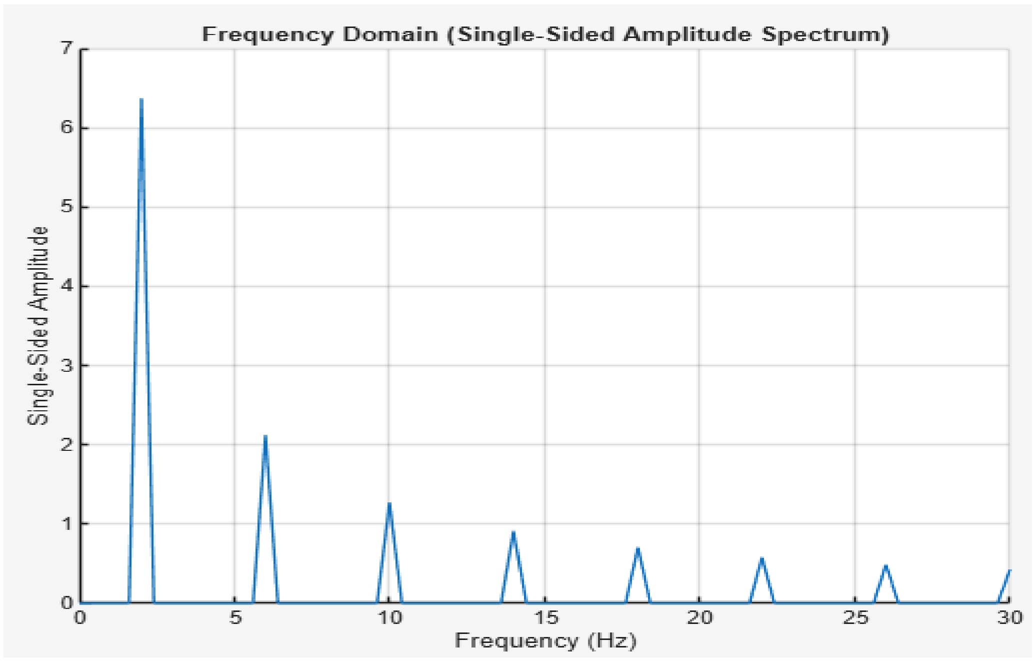


Figure 25: Frequency Domain representation of the square wave

Spikes are seen at 2 Hz, 6 Hz, 10 Hz, and 14 Hz.

- **Clean Sine Wave:**

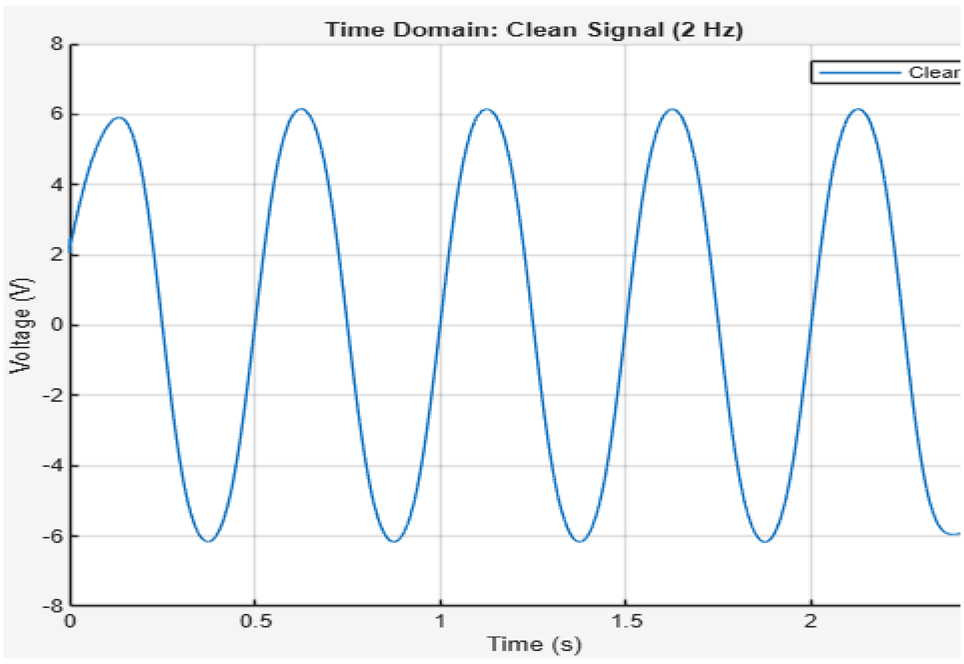


Figure 26: Filtered sine wave at F = 2 Hz

FFT:

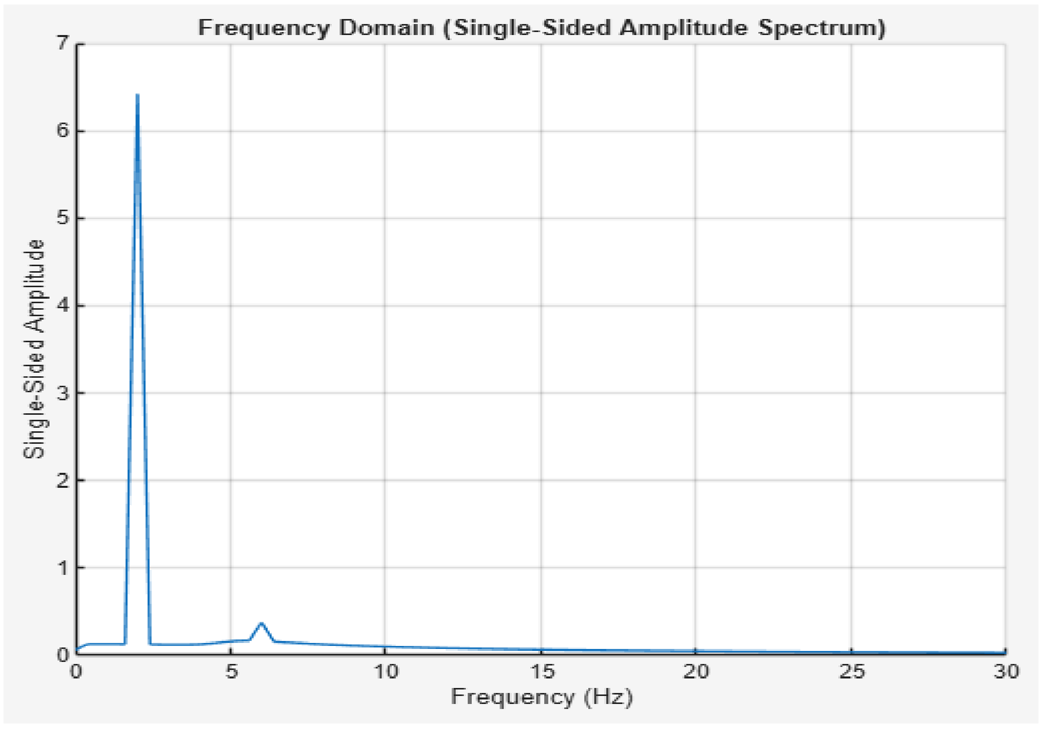


Figure 27: Frequency Domain representation of the filtered sine wave

A single spike at 2 Hz. The multi-rate filter (with F_c = 5 Hz) successfully removed the 6 Hz harmonic.

- **Noisy / Denoised:**

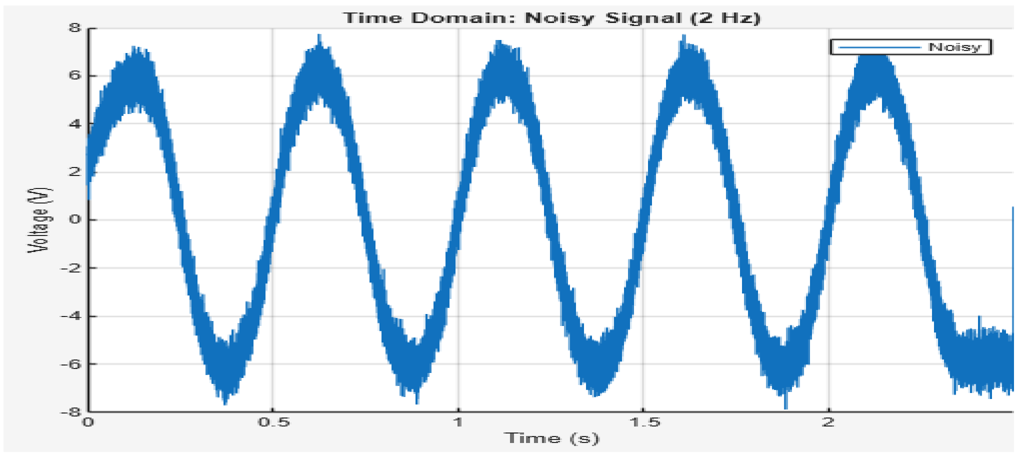


Figure 28: Noisy sine wave at F = 2 Hz

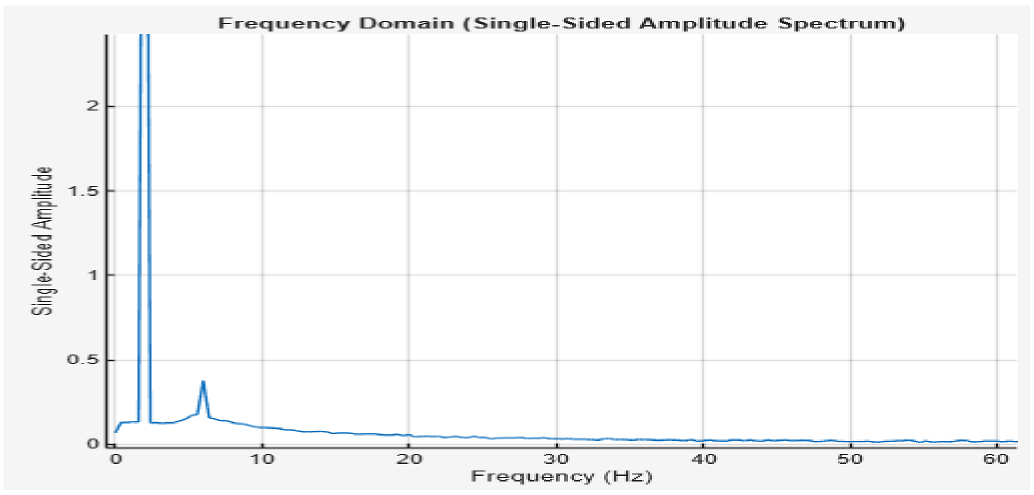
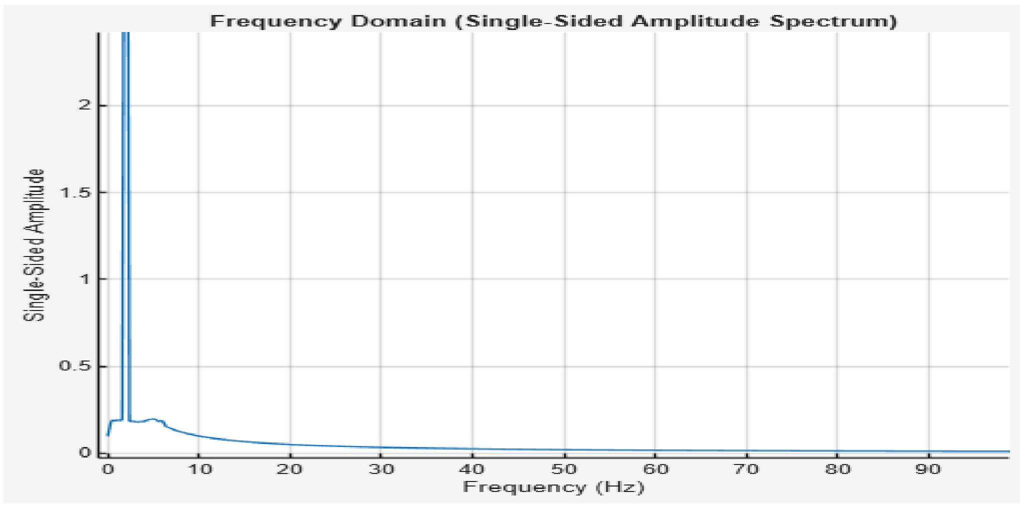


Figure 29: Frequency Domain representation of the Noisy sine wave



The 0-20 Hz FFT view provides a clear analysis. The noise floor rises from 0-20 Hz, and the denoise function clearly removes all energy above the 5 Hz cut-off, leaving the 2 Hz signal clean.

5.7.4. Key Observation: Denoising the Square Wave

A critical observation is made when the Denoise function is applied to a **noisy square wave**. The result is a **clean sine wave**, not a clean square wave.

This is the correct and expected behaviour. The "Denoise" function is a low-pass filter. It cannot distinguish between "bad" high frequencies (noise) and "good" high frequencies (the square wave's harmonics). It simply removes *all* high-frequency content.

By removing the noise, it *also* removes the harmonics that define the "squareness" of the wave, leaving only the fundamental sine wave. This powerfully demonstrates that the filter is working exactly as designed.

5.8 Noise Analysis and Elimination

The AWGN adds unwanted power across the entire frequency spectrum. This is visible on an FFT as a raised "noise floor."

Eliminating the Noise: At the receiver, this noise can be effectively eliminated. Since our desired signal is now a single, known frequency (e.g., 90 Hz), we no longer need a low-pass filter. The optimal solution would be to use a sharp **Band-Pass Filter** (BPF) centred precisely at 90 Hz. This filter would pass only the 90 Hz signal and reject the noise at all other frequencies, significantly improving the signal-to-noise ratio (SNR) and recovering the original clean sine wave.

6. Challenges and Conclusion

Challenges:

1. **Numerical Instability:** The primary challenge was the "Matrix is close to singular" error when designing the 6th-order filter for the 2 Hz signal. The ratio of Fs (20 000 Hz) to fc (5 Hz) was too large. This was solved by implementing a **multi-rate processing** technique, where the signal was resampled to a stable 2000 Hz, filtered, and then resampled back to 20 000 Hz.
2. **FFT Visualization:** The default FFT plot made it difficult to see the effects of noise. This was solved by implementing a **dynamic x-axis zoom** that automatically shows the most relevant frequency range (harmonics for clean signals, noise floor for noisy signals) for each of the three frequencies.

Conclusion:

This project successfully demonstrated the design and simulation of a signal generator for 2 Hz, 90 Hz, and 700 Hz. The core engineering principle of generating a sine wave by filtering a square wave was validated on two platforms.

In Proteus, a hardware-accurate model using an ATmega328P and three dedicated 3rd-order active filters was built and tested. The necessary calculations for the timer-based PWM and the active filters were performed, and the circuit was proven to function as designed, providing high-purity sine waves for all three frequencies.

In MATLAB, a mathematical model using a 6th-order filter confirmed the theoretical purity of the filtered signal via FFT analysis. The effects of channel noise were also simulated, and a practical engineering solution (band-pass filtering) was proposed. Both methods successfully meet all project requirements.

7. Appendix\ Links to resources:

1. Link to Proteus Circuit:
https://drive.google.com/file/d/1j9MFlEbqhombrxnsv0c70apmBJqqHKj5/view?usp=sharing

2. Link to C-code:
https://drive.google.com/file/d/1j0Kn5FceM_u_FpotbGxoAYfbL4byGH8l/view?usp=sharing

3. Link to the MATLAB GUI Signal Generator App:
https://drive.google.com/file/d/1V44qBRsFb3jBh22_OgZmT28Q2Jrek5mP/view?usp=sharing

8. References:

[1] M. A. Mazidi, S. Naimi, and S. Naimi, *The AVR Microcontroller and Embedded Systems: Using Assembly and C*, 1st ed. Upper Saddle River, NJ: Pearson Prentice Hall, 2011, pp. 341-390.

[2] J. G. Proakis and D. K. Manolakis, "Filter Design," in *Digital Signal Processing: Principles, Algorithms, and Applications*, 4th ed. Upper Saddle River, NJ: Pearson Prentice Hall, 2007, pp. 525-600.

[3] 3Blue1Brown, "But what is the Fourier Transform? A visual introduction." [Video]. *YouTube*, Feb. 26, 2017. Available: https://www.youtube.com/watch?v=spUNpyF58BY

[4] All About Electronics, "Active Low Pass Filter (Sallen-Key) Design Tutorial." [Video]. *YouTube*, Aug. 1, 2017. Available: https://www.youtube.com/watch?v=00_nQN1a0vY