DAC Scaling Circuit for Control Voltage Output

Jenna Bunescu

1. Design Requirements

5X amplification, non-inverting output with 1-cent accuracy over 8 octaves. Optional adjustment to set DC offset. Input: 0-2 VDC. Output: 0-10 VDC with 1 k output load impedance. Intended usage: The scaling circuit amplifies the 0-2V output of a DAC by 5X enabling a 3.3V microcontroller to output control voltages in the range of 0-10V. The output of the circuit connects to the control voltage input of a VCO where each volt corresponds to an octave of pitch. Suggested experiment: Connect the output of the circuit to the control voltage input of a VCO and listen to the output of the VCO. (83.33 mV = a semitone/adjacent key on a piano.)

2. Design Approach

2.1 Filter Design

When choosing a suitable filter to transform the PWM signal into a constant analog signal, I experimented with a voltage follower and a two-stage Butterworth filter. While the Butterworth filter would have provided higher accuracy filtering, and the voltage follower could have potentially reduced noise, seeing that both of these required an additional op amp, and that would mean an additional voltage source to power the op amp, I decided to keep the circuit simple. However, moving on, another op-amp could still be implemented as the final circuit included the NE5532P op-amp, which is a dual op-amp.

Ultimately, I decided to use a two-stage low-pass RC filter. The R and C values were chosen to provide a suitable cutoff frequency, using

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

I planned to use a PWM with a frequency of 62.5 kHz. I needed to choose a cutoff frequency that was much lower than the frequency of the PWM. Choosing $C = 0.47\ \mu$F and $R = 10\ \text{k}\Omega$ yielded a cutoff frequency of

$$f_c = \frac{1}{2\pi RC} = \frac{1}{2\pi \cdot 10{,}000 \cdot 0.47 \times 10^{-6}} \approx 33.86 \text{ Hz}.$$

The cutoff frequency is low enough so that most of the PWM signal is attenuated, and the time constant is 0.0047 s, which is low enough for real-time applications.

While this section of the circuit would be useful in acting as a small-scale DAC, it wouldn't affect the functionality of the circuit when the voltage source is a DC signal, therefore making it versatile.

2.2 Amplifier Design

To achieve an amplification factor of 5, a voltage gain of 5 was required in the op-amp circuit. Since a non-inverting op-amp configuration was used, the gain of the circuit was determined by the following equation:

$$G = 1 + \frac{R_f}{R_i}$$

where R_f is the feedback resistor (connected from the output to the inverting input), and R_i is the resistor connected from the inverting input to ground. Rearranging this equation for a gain of 5 gives:

$$5 = 1 + \frac{R_f}{R_i} \Rightarrow \frac{R_f}{R_i} = 4$$

This means the feedback resistor must be four times the value of the input resistor. To satisfy this ratio and use standard resistor values, I chose

$$R_f = 40\ \text{k}\Omega \quad \text{and} \quad R_i = 10\ \text{k}\Omega.$$

This resistor selection provides the desired gain while keeping the resistor values within a practical and readily available range.

An NE5532P op-amp was used as it has a very low input noise, low distortion, and high slew rate (allowing the op-amp to respond quickly to changing signals). These attributes made it a great choice for an audio circuit. The op-amp is not rail-to-rail, so I needed to provide it with $\pm 12\,\text{V}$, instead of grounding the negative V_{cc} pin.

To achieve 1 cent accuracy over 8 octaves, I used highly accurate components. Therefore, the resistors and capacitors had a tolerance of 1%. During testing, I observed some noise in the output voltage. I ultimately determined that most of the noise originated from the Virtual Bench itself. Given the limitations of the testing equipment, I thought that this noise could be disregarded, assuming that a more stable input signal would yield a high accuracy in the final implementation.

2.3 Components

Seven through-hole 10k Ω resistors with 1% tolerance were used in the circuit, two of which corresponded to the filter portion, and five of which belonged to the amplifier portion. As I couldn't find any 40k Ω resistors in the lab, I used four 10k Ω resistors for

R_f. Two aluminum 0.47 μF 20% 50 V Radial capacitors were used in the filter portion. Additionally, I used a NE5532P op-amp. All the components used were taken from the lab's inventory.

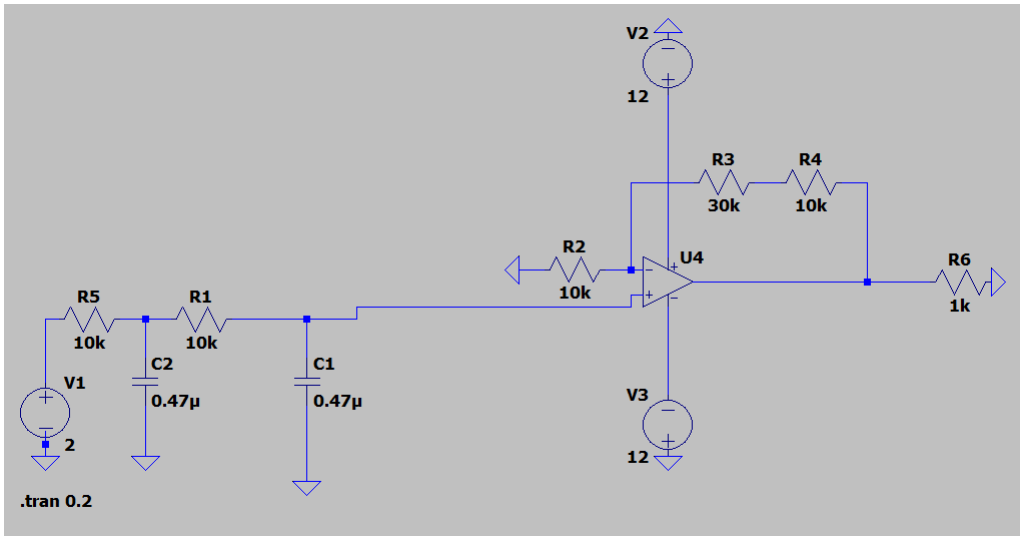
One 10k Ω resistor with 1% tolerance costs about 0.10$ a piece from DigiKey.

One ALUM 0.47UF 20% 50V RADIAL capacitor on Digikey costs 0.11$

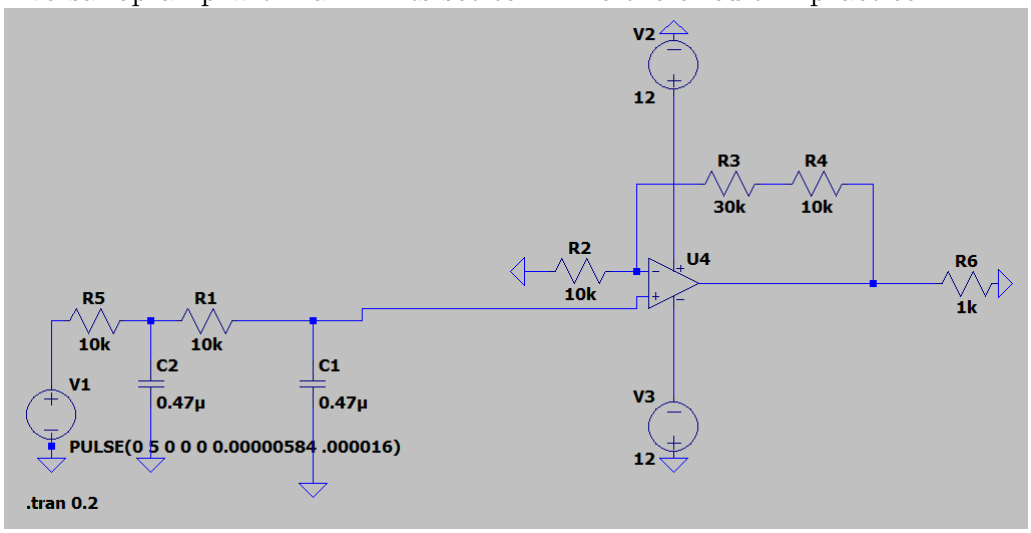
The NE5532P op-amp on Digikey costs 0.57$

The Arduino Uno R3 ATMEGA328 microcontroller on Digikey costs $27.60

3. Demonstration

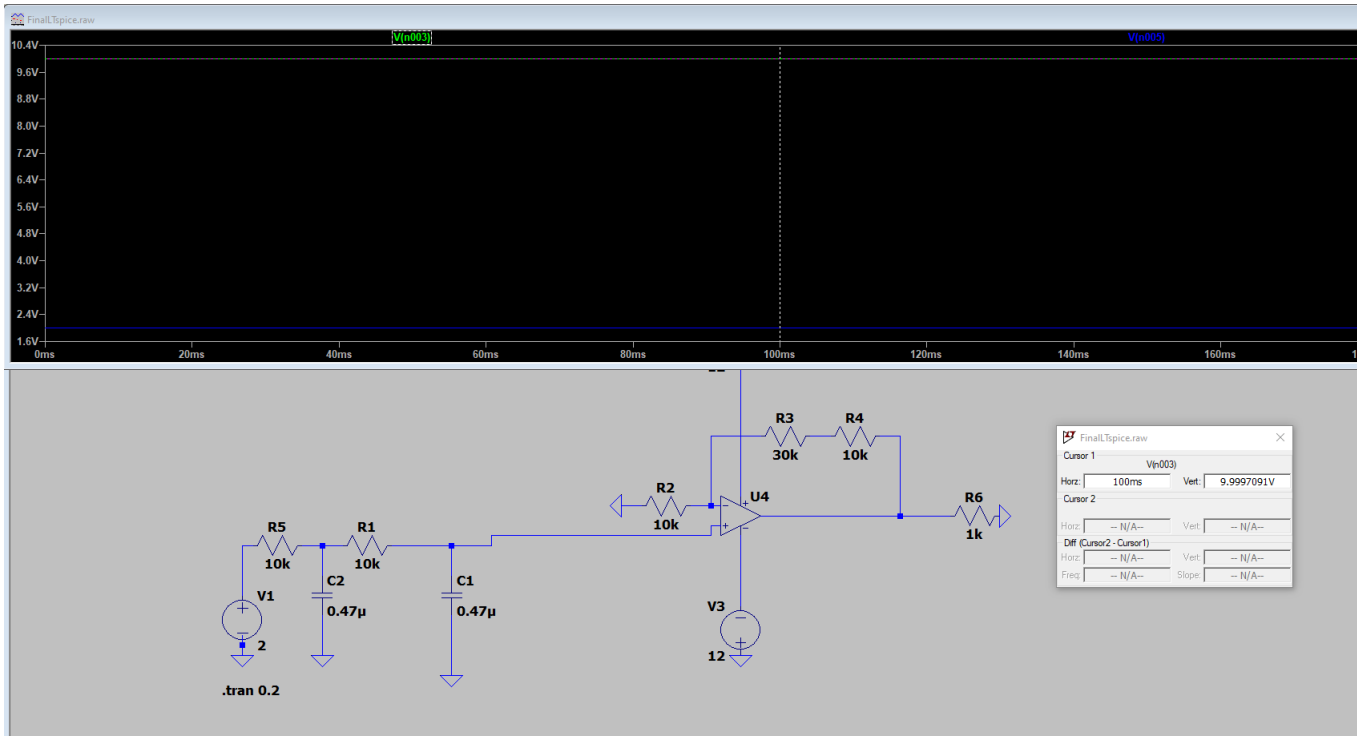


The simulation under LTspice yields ideal conditions for each component, which does not take into account several factors such as resistor and capacitor tolerance and operating temperature. The NE5532P op amp is not present within LTspice's library, so I utilized a universal op amp with rail limits set to mimic the circuit in practice.

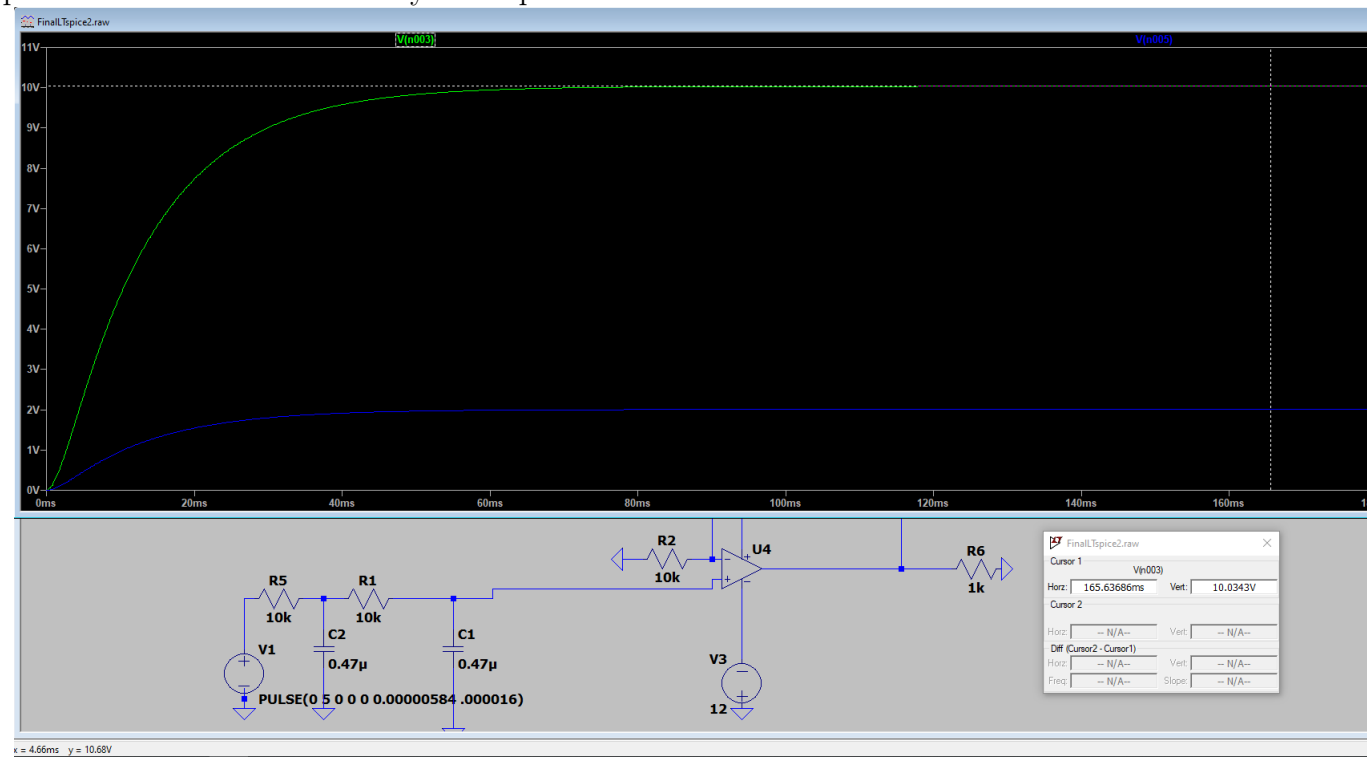


LTspice also allowed to configure an ideal PWM input by utilizing the PULSE function for the input voltage supply. For the simulation, the pulse was configured to mimic

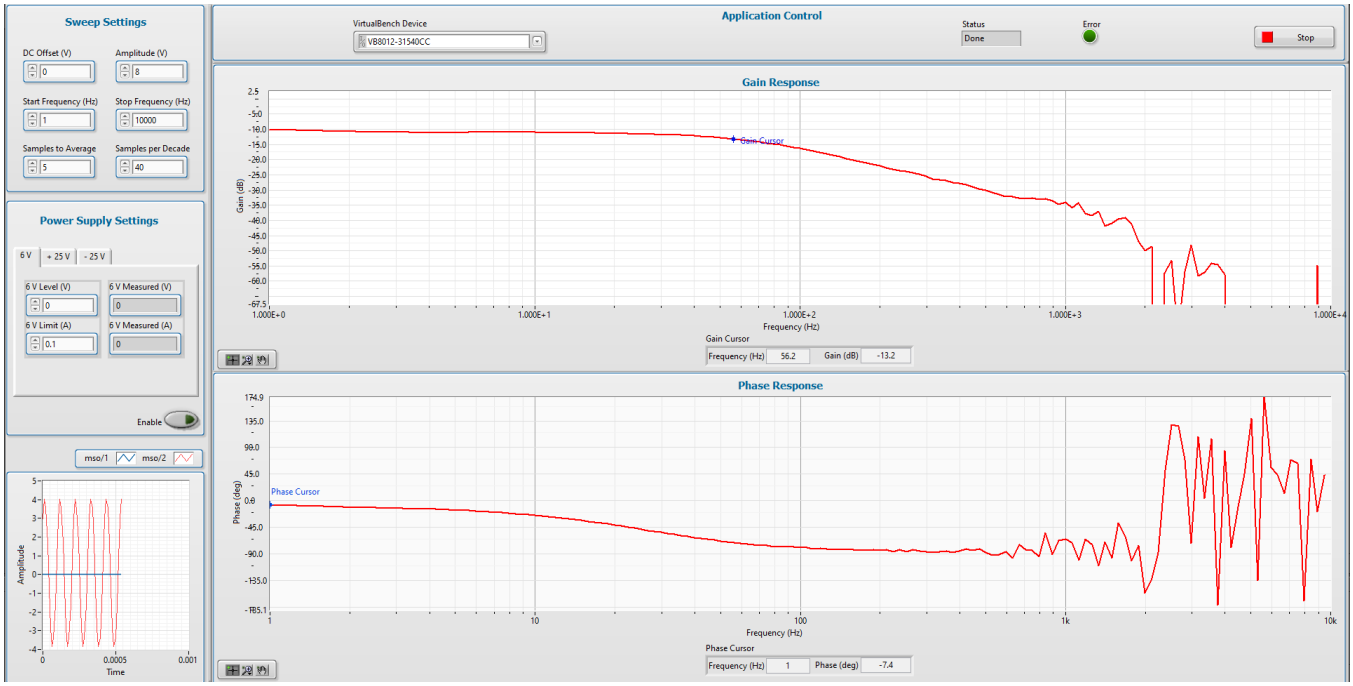
the PWM output of the Arduino Uno, which has a high voltage of 5V, with a period of $1.5 \cdot 10^{-6}$ s^{-1}, based on the Arduino's set frequency to 62.5 kHz. This particular example shows a duty cycle setting of 93/255 (36.47%).



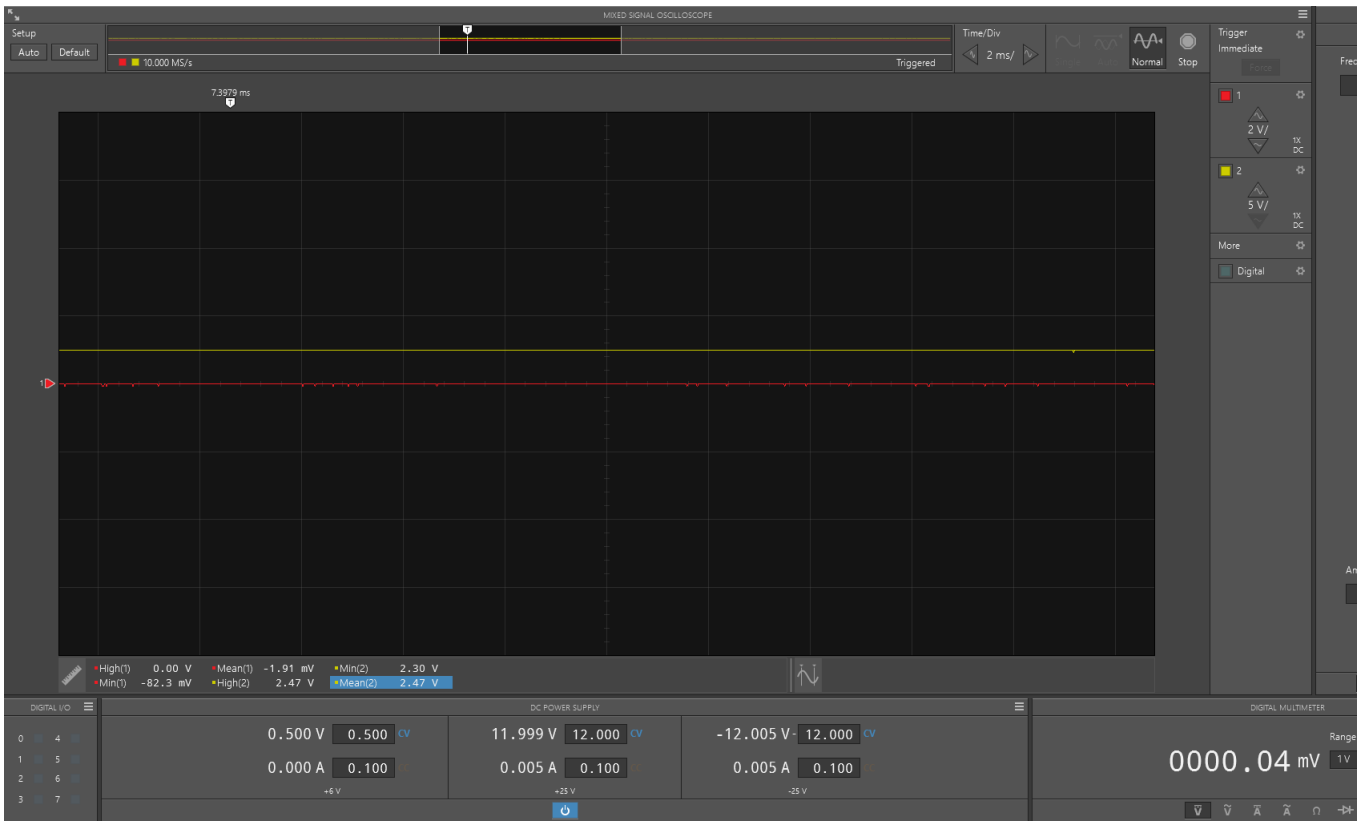
Running a DC input voltage of 2V resulted in an output of 9.9997 V from the op-amp. With a desired gain of 5, this results in an error of $3 \cdot 10^{-5}$ V (0.03%), which is within the requirements of a 1 cent accuracy in output.



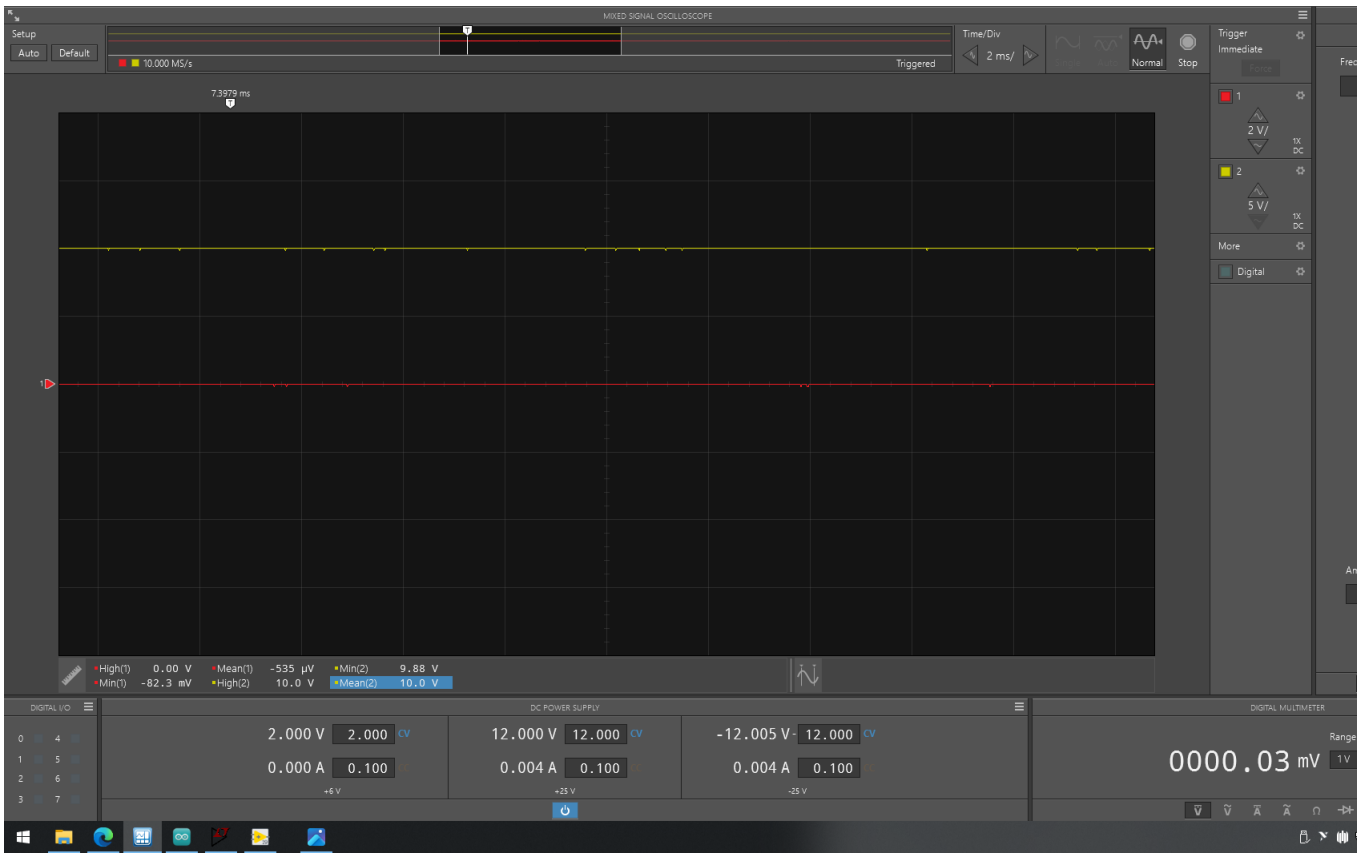
Running an ideal PWM output with a high of 5 volts, duty cycle of 36.47%, and a frequency of 62.5 kHz, simulating the Arduino Uno output. The final result has an error of 0.034 V (0.34%), which is within the 1 cent accuracy requirement, but requires a rise time to reach. This is a potential trade-off with the utilization of a smaller capacitor, where smaller capacitors have a decreased rise time with an increased bandpass width. I chose 0.47μF capacitors because the cutoff frequency is low (about 33.86 Hz), while having a small rise time of only about 80 milliseconds.



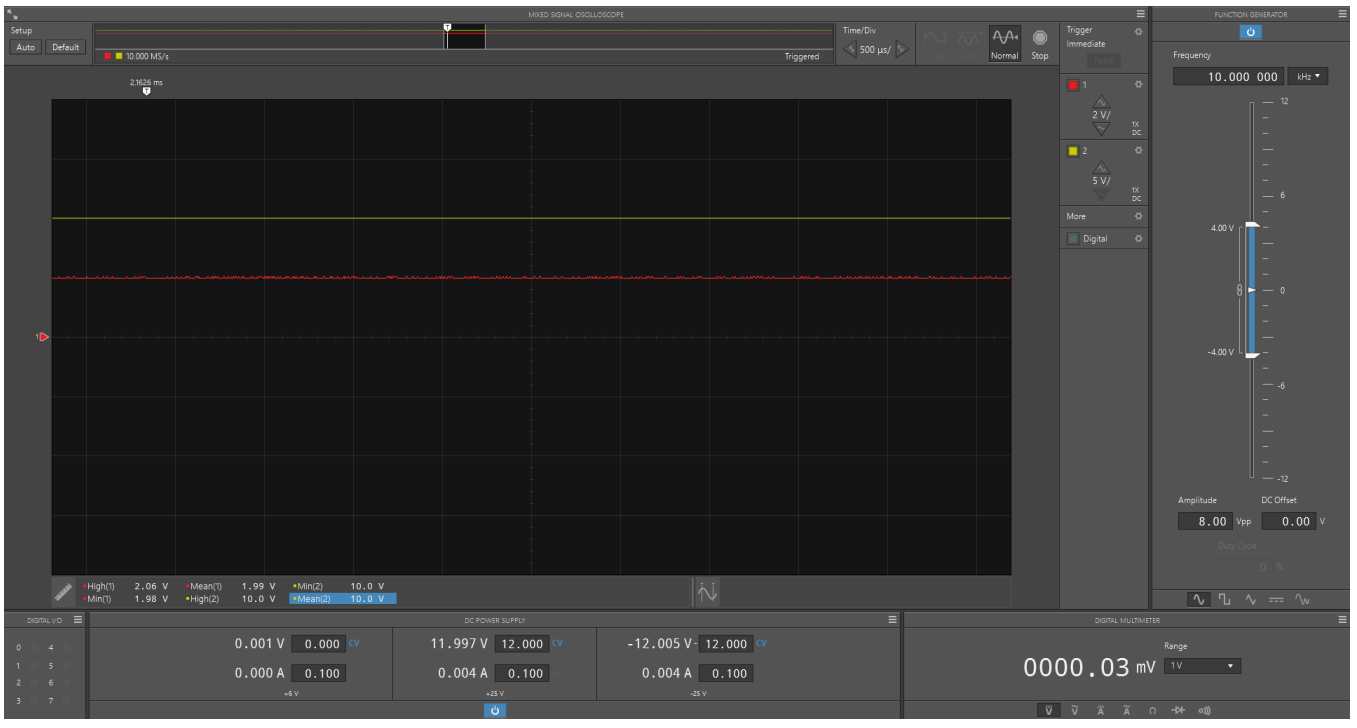
The bode plot somewhat matched the predictions for the cutoff frequency, having the cutoff frequency at roughly 56.2 Hz. Gain at 1 Hz was found to be -10.2 dB. The difference is likely due to the tolerances of the resistors and capacitors. The slope of the gain past the cutoff frequency is roughly -27 dB per decade, with a phase of -7.4 deg, shifting down to around -90 deg. Significant variations in phase arose around 1 kHz.



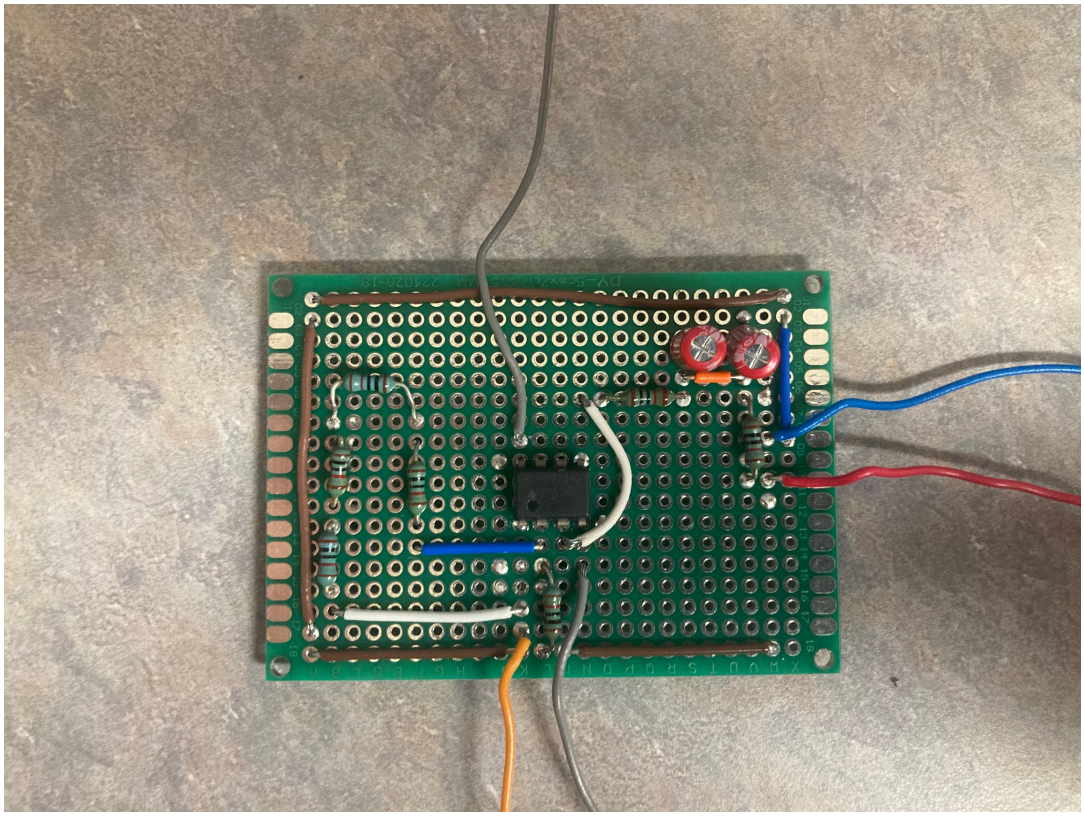
On the Virtual Bench, Channel 1 measured the output of the system, while Channel 2 measured the input from the +6V DC Power Supply. When inputting 0.5 V, the Virtual Bench outputs a mean of 499 mV, with a total noise range from 576 mV to 412 mV. The measured output was found to have a mean of 2.47 V, with a noise range from 2.47 to 2.30 V. The output noise range was found to be 0.17 V, with the mean having a 1.2 % error. The input had a noise range of 0.164 V, and its mean had a 0.2 % error. The noise range of the output was consistent with the noise range of the virtual bench, however, the output retained an error over 1 %.



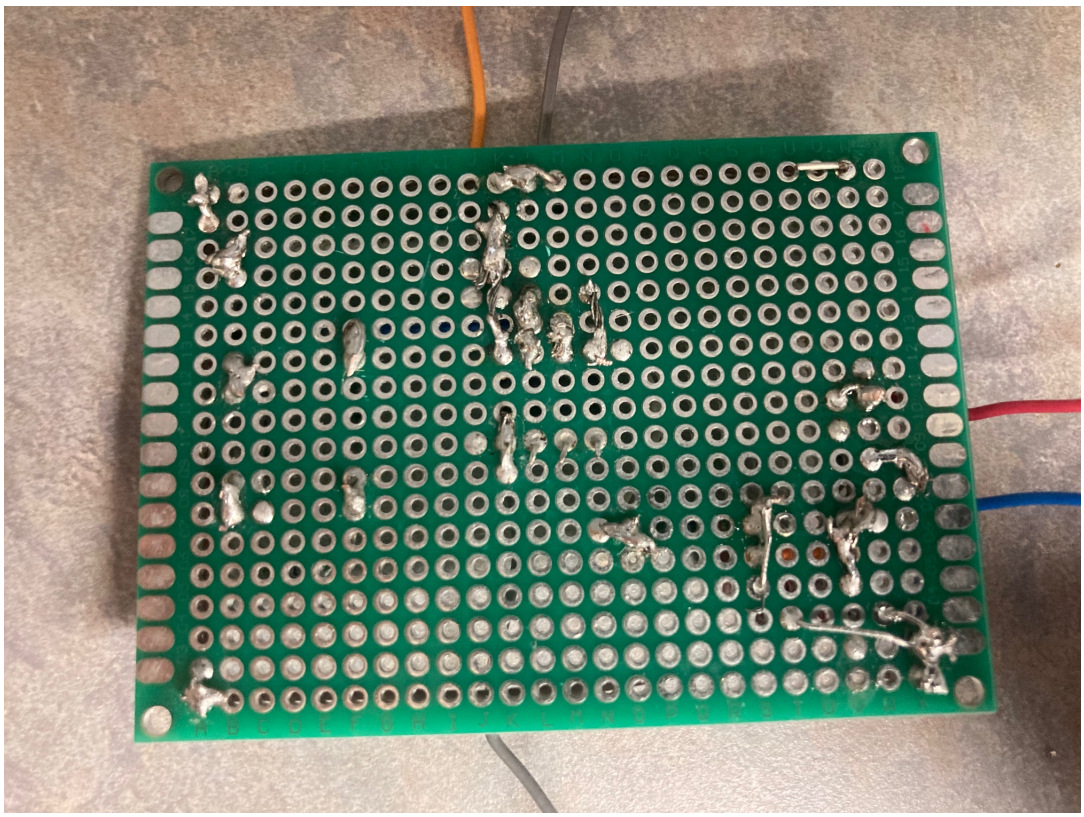
Having a 2V input (representing the maximum) had an output mean of 10.0 V, with an error range of 0.12 V (from 10.0 V to 9.88 V). The input had a mean of 2.01 V, with an error range of 0.08 V. The output mean had an error too small to measure, and the input had a 0.5 % error. It has been found that the output is consistently slightly lower than the expected value, this could be due to the output resistor tolerances being higher than the ideal value of 10 kΩ.



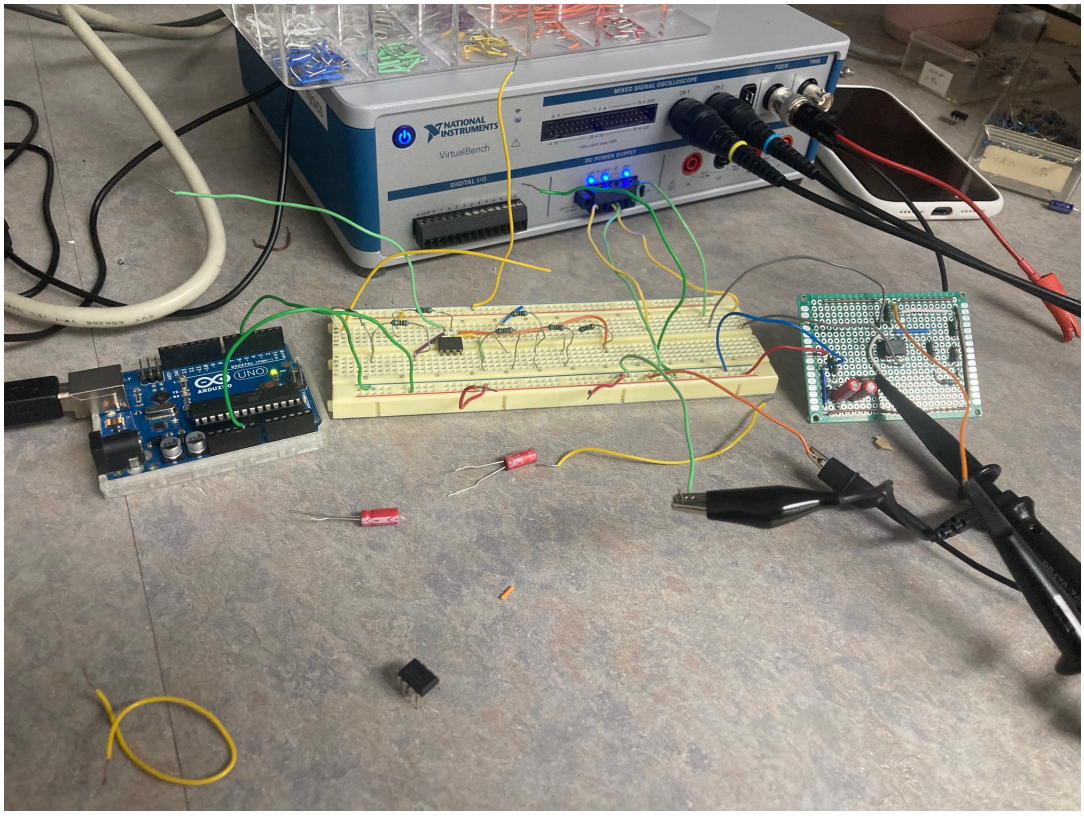
Another test was conducted utilizing the Arduino Uno's PWM signal to act as the input to the circuit. The Uno was set to a frequency of 62.5 kHz, with a duty cycle of 39.61 % (int input of 101 out of a 255 range) relative to a high of 5 V. This duty cycle was set to the desired input voltage of 2 V. Channel 1 was reconnected to the output of the second order RC lowpass filter, due to the high voltage of the PWM output. The output had a mean of 10.0 V, with a noise range not being readable. The input had a mean of 1.99 V, with a noise range of 0.08 V. The output error was too small to be measured, and the input error was 0. 01%. The PWM output achieved a much more accurate output than the DC outputs, with a similar noise range for input. However, the output was not always consistent, with slight variations usually ranging between 0.08 V - 0.12 V for the minimum range.



A top down view of the completed circuit soldered onto a through-hole board. Note that input is on the right side, with red for positive input and blue for ground. The two gray wires are for the op-amp's rail to rail voltage, and the orange wire represents the output of the circuit.



The soldering connections within the through-hole board. Positions are relative to the previous image, flipped along the image's y-axis.



A visualization of how testing was connected, in this example utilizing the PWM output of the Uno. Since the through-board has 5 connections, a breadboard was utilized to allow connections to both the Uno and the VirtualBench. Ground and input voltage

were run along the breadboard's rail, while the rail voltages of the op-amp were supplied through the breadboard's interior lines. Channel 1 was connected to the second resistor in the circuit's second order filter, while Channel 2 was directly connected to the output. Note that the other elements on the breadboard are not utilized, they are parts of the older version of the circuit.

Uno_Final_Sketch_Pic2.PNG.ino

```
// Arduino Uno Code Final
// Author: Nicholas Horner
// Last Edited: 5/8/2025

// Digital pin 5 is targetted for the output (frequencies vary for PWM signal depending on pin)
const int pwm = 5;

// Additional options being looked into was using a float to int unit converter, allowing for more precise input for the Duty Cycle
// For our project this wasn't necessary

void setup() {

  // PWM set to output
  pinMode(pwm, OUTPUT);

  // Frequency address call to the arduino Uno, setting B00000001 sets the frequency output to 62.5 kHz
  // The Arduino Uno has 3 bit settings, the last three digits change the frequency to set increments
  // Default frequency on digital pin 5 is 976.56 Hz
  // Source: https://www.etechnophiles.com/change-frequency-pwm-pins-arduino-uno/
  TCCR0B = TCCR0B & B11111000 | B00000001;

}

void loop() {

  // PWM output with a duty cycle (as an int) set out of 255
  // (Input / 255) Represents the duty cycle percentage, here the duty cycle is set to 19.61%
  analogWrite(pwm, 50);

}

```

This is the sketch used to test the Virtual Bench output on the Arduino Uno.

4. Lessons Learned

Through this project, I learned that an Arduino does not provide true analog voltage output; instead, it uses PWM (Pulse Width Modulation) signals on its pins. I learned how to convert a PWM signal into a clean analog voltage using filters and experimented with various types of filters. This process taught me how to select appropriate cutoff frequencies and understand the trade-off between cutoff frequency and the time constant for a given application.

I also gained hands-on experience with op-amps, learning that different models have unique pin configurations and capabilities. I got the chance to solidify my understanding of rail-to-rail outputs and the limits of non-rail-to-rail op-amps, as well as how to deal with those constraints.

In LTspice, I learned how to simulate PWM signals, which helped me test and visualize circuit behavior before building. Additionally, I became more confident in using LabVIEW to generate Bode plots.