ME 507

MECHANICAL CONTROL SYSTEM DESIGN

Term Project

Charlie Refvem

Department of Mechanical Engineering

California Polytechnic State University

Joseph Penrose

Jared Sinasohn

June 14th, 2024

Table of Contents

Abstract

A strobe tuner was built using an STM32 MCU on custom fabricated printed circuit boards. The project demonstrates real-time programming, including closed-loop motor control, on bare-metal. An extensive design, including a very in-depth analog circuit incorporating complex audio filtering, is presented. After facing a great number of troubleshooting challenges, a proof-of-concept demonstration showed that the function of the strobe tuner was a complete success.

Project Overview

Description

The object of the project is a strobe tuner. A strobe tuner is an electromechanical device used to tune musical instruments. Strobe tuners are still used in music studios as the most precise way to tune an instrument, even when digital chromatic tuners (which use digital signal processing to determine the frequency of the input) have become abundant.

The project requires both analog and digital components. On the analog side, we need to measure the output of an instrument, either from a direct electric input (as in an electric guitar through a cord) or from a microphone. The signal must be carefully conditioned and filtered to remove overtones and harmonics that obfuscate the most important frequency – the fundamental of the note being played. That conditioned electrical signal is used to drive a comparator which creates a logic-level square wave at the same frequency as the input, which can be read by a microcontroller and used to drive a strobe light through a MOSFET.

On the digital side, we primarily need a microcontroller to handle user inputs that select pitch. We also use the microcontroller for closed-loop motor control which helps ensure the most precise speed control of the strobe disk as possible. We also planned to use the microcontroller to read the signal comparator output so that the microcontroller can automatically select the closest musical note to the pitch being handled through the analog circuitry.

A user will be able to adjust the input gain, the level of low-pass filtering applied to the input, and the brightness of the strobe light. Directly inserting a cable for an electric instrument defeats the internal microphone. To control the strobe disk, the user can choose the desired pitch through a rotary encoder or switch to automatic note detection through a toggle switch. The selected note is reported to the user via a 16-segment LED display and two LED's that have sharp and flat symbols stenciled over them.

As a class project, we were also required to implement a "dead-man switch" through an RC controller. Since the RC controller was connected, we also planned to use the steering wheel on the controller as an alternative pitch selection method.

Motivation

The team are both musicians who play instruments that require frequent tuning. Chromatic tuners, albeit common and generally regarded as "good enough," can be finicky and imprecise. They are also fairly small, which is convenient for dropping in an instrument case, but can be annoying to hold near an instrument while needing both hands for playing and tuning. We sought a more benchtop-oriented solution that can be kept full-time in a dedicated practice area.

Electric guitar accessories use a lot of (unofficial, but universal) standards that pervade almost all electronic instruments currently made, rivaled only by the Eurorack synthesizer standards for rack- and panel-mounted modular synthesizers. They use ¼" mono or stereo jacks and plugs for audio, and everything runs on 9V DC power through 2.1mm center-negative barrel jacks. Circuits that require higher voltages still use the same 9V power supplies, but rely on boost converters and voltage inverters. It is also generally understood that audio circuits should always be able to handle high-impedance inputs on the order of 200mV to 1V peak-to-peak AC. By sticking to these standards, we can build a highly accessible product that anyone around the world should be able to use with any guitar pedal power supply they already have.

Real strobe tuners are highly expensive. Only one major company, Peterson, remains manufacturing real strobe tuners for retail. Most of their products are lower-cost and use digital displays that claim to use “strobe tuner technology,” and their actual electromechanical models start at around $800 and stretch upwards of $5000.

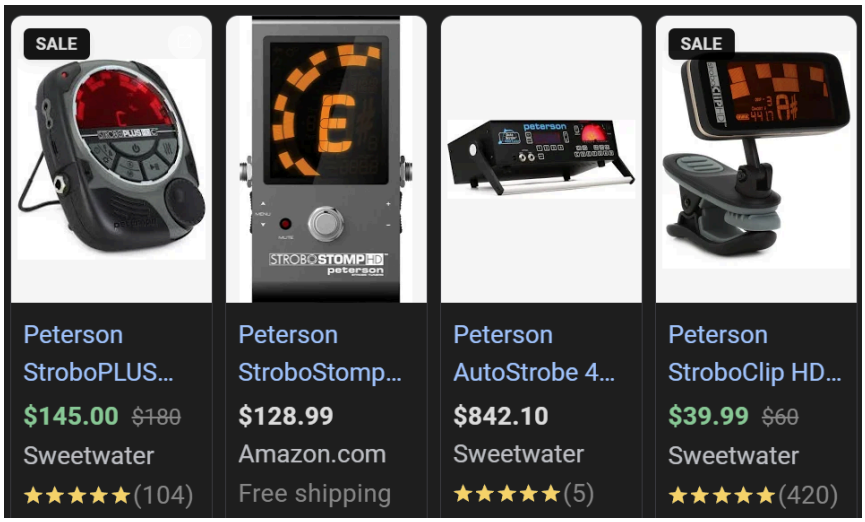


Figure 1. Commercially available strobe tuners.

They are also bulky for reasons that the team is not able to immediately understand. We were convinced that we could deliver the same performance out of a single MCU, a small gearmotor, readily available LEDs, and a reasonable amount of analog circuitry, all at a significantly lower cost.

Background

A motor spins a disk with slots cut in it arranged in concentric rings. The first ring has four slots (two open, two closed), and each subsequent ring carries twice as many slots in it. This is known as a strobe disk.

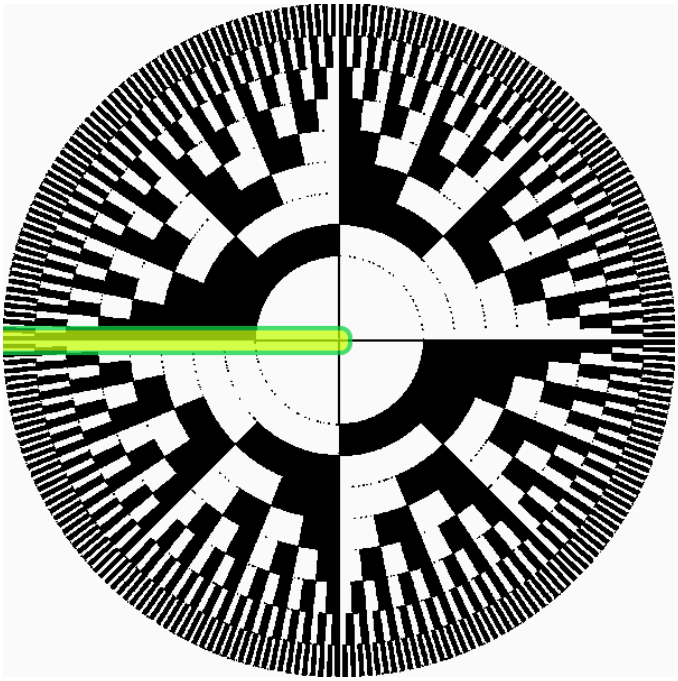


Figure 2. A strobe tuner disk.

Looking at one fixed line along the radius (highlighted and outlined in green in Figure 2) and visualizing how the disk rotates behind that line, you would see that the slots in the disk pass by the line and change state from open to closed at multiples of two of the same frequency of disk rotation. For example, call the frequency of the disk's rotation *f*. The first ring has two open slots and two closed slots. It changes state four times in one disk rotation, at equal intervals, so its state changes represent a 2*f* frequency at 50% duty cycle. The second ring has twice as many slots, which changes state eight times per rotation, and represents a 4*f* frequency. The pattern repeats for 8*f*, 16*f*, and so on.

Now, spin the disk at a known frequency and place a bright light behind it. If the light is left on continuously, the image of the spinning disk would appear to an observer as a blur. However, if the light flashes at the same frequency as one of the disk rings, then the resulting image would appear to be static. This is because the location of the slots would be the same every time the light flashes. The viewer would see the corresponding slot ring standing in place. If the frequency were to be off by a small amount, then the closest equivalent ring of slots would appear to rotate in a direction corresponding to whether the light is strobing too fast or too slow.

This is the concept behind a strobe tuner. The disk is spun at a target frequency representing a musical note, and the vibration, or sound, created by an instrument is used to drive the strobe light behind the disk. When the image appears static, then the instrument is playing the same frequency as the disk is rotating (or one of its exact multiples of two).

In Western music, the twelve-tone equal temperament chromatic scale has twelve pitches represented by the letters A-G, with six semitones placed between the letters (known as "sharps," #, and "flats," ♭) that

repeat at the end of the scale. That is, after G#, the last note in the scale, the next note is A again. The frequency of this A is exactly double that of the previous A; this is known as an "octave." When discussing wide ranges of notes, there are often numbers appended to the note names to represent exactly which note on a piano it is referring to (the "octave number"). Those notes on the piano are:

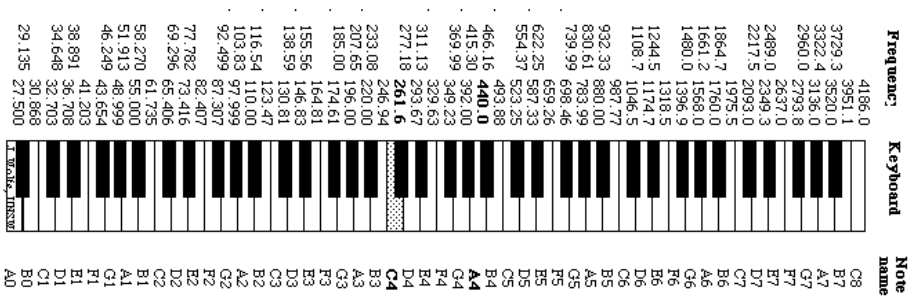


Figure 3. Piano notes, names, and their corresponding frequencies in Hz. The sharps and flats have been omitted from the note name column.

In order to represent every available note on the keyboard, we must drive the motor such that the slowest disk represents the lowest frequency of that particular note on the piano. Then, the subsequent slot rings represent the next octaves up the keyboard. With enough rings, only twelve discrete motor speeds are required to represent the full range of musical notes.

Requirements

As a term project for ME 507, we were given a number of requirements that the project needed to fulfill. The requirements include…

"...a custom PCB designed around an STM32F411 MCU (or similar) programmed in either C, C++ (or Rust, with permission)."

We chose the STM32L476RGT6 because it is the same MCU used in the Nucleo boards provided in ME 405 and 507, and the familiarity made porting existing code that we had developed in those two classes easier. The number of available timers and communication protocols suited our needs, and we determined that the 80 MHz clock frequency would be more than adequate to run our loop. We programmed our project in C using the STM32 CubeIDE.

"...two or more actuators, such as motors, driving the machine, actuated by suitable electronics, such as motor drivers."

Our two actuators are the strobe disk motor and the strobe light. We also consider the LED display as somewhat of an actuator.

"...two or more unique sensors."

Our sensors include the magnetic encoder in the motor, the mechanical encoder in the pitch select knob, and auto-select switch, and the microphone and audio input jack.

"...some sort of closed-loop control loop or similarly complex algorithm."

The speed of the strobe disk must be highly precise with little variation for proper function, so closed-loop speed control was a must. We experimented with PI control as well as feed-forward control.

"...a wireless controller allowing you to command the bot hands-free or to be used as a wireless e-stop. The controller and receiver will be provided to students for use during ME 507."

We planned to use the provided RC controller's trigger as a motor dead-man switch. The motor would only turn so long as the remote trigger was held. We also planned to use the steering wheel as an alternative pitch selection method. We designed an interface between the MCU and the RC receiver to handle the power and level shifting and provided a connector for it on the circuit board, but time constraints did not allow us to finish the implementation in software.

"...[an acceptable level of safety] for builders, operators and bystanders, and conform to the ME 507 lab safety rules."

Our project, when fully assembled, has no accessible moving parts, high voltages, or extreme temperatures. It operates on commercially available power supplies using standard connections. The case is plastic and thus does not need to be grounded.

Our manufacturing was also restricted to printed circuit boards, 3D printing, and laser/waterjet cutting. Our project is entirely built using purchased hardware and 3D printed parts with no post-processing required.

Theory & Design

Motor Frequency Calculations

Before a motor could be selected, we needed to know what motor frequencies would actually be required to capture as much of the musical note range as possible. During initial design, a 3D-printed strobe disk was limited to a maximum frequency ring of 128*f* and minimum of 2*f*. The slots in a 256*f* ring would have been too small to be resolved on the 3D printers we have access to, and a 1*f* ring would mean that the first ring was missing half of its material and would be neither structurally sound nor well-balanced. That means that only seven octaves are available across seven rings, and we would have to choose to lose some notes from the extreme low and high end of the range. For this prototype we chose to capture, at minimum, the lowest notes on a piano: A0, B ♭ 0, and B0. If we take these to be the lowest frequencies, then the motor must spin at the frequencies corresponding to half of the frequency of A0, B ♭ 0, B0, etc. We used a spreadsheet to calculate the required motor speed in RPM. The RPM column already handles dividing the note frequency in half, so that the 2*f* ring represents the note's true pitch.

Note	Frequency		Motor Speed (RPM)
A0:	27.50	Hz	825.00
Bb0:	29.14	Hz	874.20
B0:	30.87	Hz	926.10
C1:	32.70	Hz	981.00
C#1:	34.65	Hz	1039.50
D1:	36.71	Hz	1101.30
Eb1:	38.89	Hz	1166.70
E1:	41.20	Hz	1236.00
F1:	43.65	Hz	1309.50
F#1:	46.25	Hz	1387.50
G1:	49.00	Hz	1470.00
G#1:	51.91	Hz	1557.30

Figure 4. Required motor speed for each musical note.

These speeds, along with the 2*f* through 128*f* rings, cover the entire standard 88-key piano except for the four highest keys. We determined this was an acceptable compromise – because on the other hand, a bass guitar is tuned to E1, 41.2 Hz, which is the 8th-lowest key on the piano, and a very common pitch to tune to. A five-string bass with a low B0 string, 30.9 Hz, is the 3rd-lowest key on the piano.

Once we knew what speeds we needed, we were able to search for a gearmotor that had these speeds available on their motor curves. Our chosen motor, a Pololu 15:1 micro gearmotor, runs on 6V and draws little current under PWM drive (which our standard 9V supply can readily provide), and appears to hit the

required speeds in the middle of its operating range. We assumed that our strobe disk, being 3D printed on the order of a few mm thick, would have a low enough inertia to not bog the motor down below the speeds we needed. It was also available with a 12 counts-per-revolution encoder (CPR) that we determined would have enough resolution for the speeds and precision we wanted.

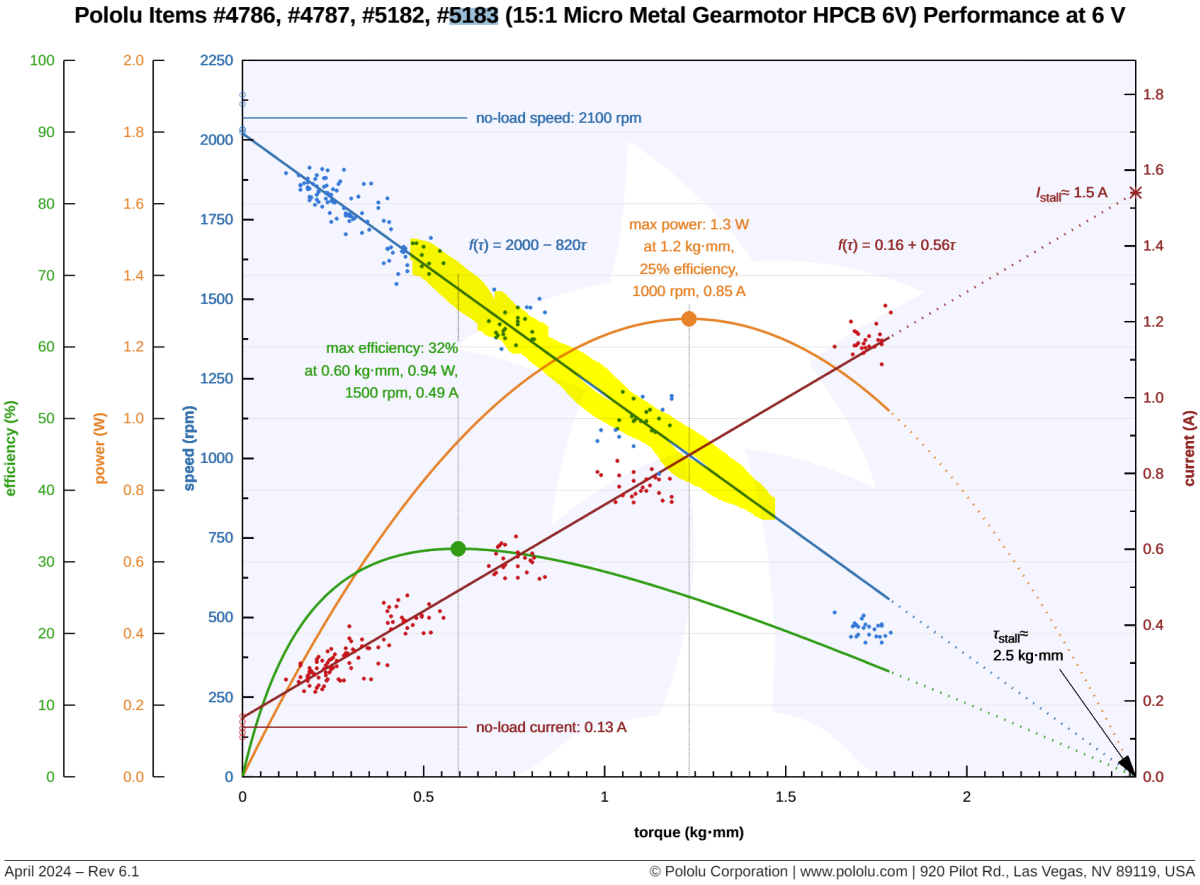


Figure 5. Manufacturer's motor curve for the strobe disk motor. Our approximate operating area is highlighted in yellow.

Having the speeds determined, we needed only to add a column to the speed calculator and determine how many encoder counts per second correspond to the required motor frequencies. The formula is:

$$\frac{counts}{sec} = (RPM)\left(\frac{min}{60\ sec}\right)(gear\ ratio)(CPR)$$

$$\frac{counts}{sec} = \left(\frac{output\ rev}{min}\right)\left(\frac{min}{60\ sec}\right)\left(\frac{motor\ rev}{output\ rev}\right)\left(\frac{counts}{motor\ rev}\right)$$

where the gear ratio is expressed as motor shaft revolutions per output shaft revolution, and the encoder CPR is with respect to the motor shaft revolutions.

Note	Frequency		Motor Speed (RPM)	Motor Speed (Encoder Counts/second)
A0:	27.50	Hz	825.00	2516.25
Bb0:	29.14	Hz	874.20	2666.31
B0:	30.87	Hz	926.10	2824.605
C1:	32.70	Hz	981.00	2992.05
C#1:	34.65	Hz	1039.50	3170.475
D1:	36.71	Hz	1101.30	3358.965
Eb1:	38.89	Hz	1166.70	3558.435
E1:	41.20	Hz	1236.00	3769.8
F1:	43.65	Hz	1309.50	3993.975
F#1:	46.25	Hz	1387.50	4231.875
G1:	49.00	Hz	1470.00	4483.5
G#1:	51.91	Hz	1557.30	4749.765

Figure 6. Updated motor speed calculation with encoder counts per second.

The MCU will use the number of encoder counts per second (normalized to the update speed of the software task) as feedback for the motor effort controller in closed loop.

Audio Processing

When it comes to musical instruments, it is well understood that pianos and trumpets and the human voice are all capable of sounding the same notes, but sound wildly different. This is known as the “timbre” of a sound, or the characteristics of a sound that make it unique from other sounds. The timbre of a sound is created by “overtones,” which are other frequencies that exist around the “fundamental,” or the strongest frequency that is most present to the ear. For example, when a guitar and a trombone play the note F2 together, it is clear to the ear that the pitches are the same because the strongest frequency from each instrument is F2. However, what makes them sound different are the characteristics of the vibration of the strings in the guitar and the brass in the trombone that generate even and odd “harmonics,” or multiples of the fundamental, at different strengths. Fourier showed that any harmonic wave could be represented as a summation of sine waves at even and odd multiple frequencies of the fundamental, and this is where these even and odd harmonics come from. Therefore, the sound being picked up by a microphone is much more complicated than just a sine wave at some musical note frequency.

There are, however, a few things working to our advantage. First, the characteristic that makes the fundamental frequency the fundamental in a musical note is simply that it is the strongest in magnitude. Therefore, even a raw input with a modestly-tuned comparator could generate a halfway-decent signal for the strobe light on its own just by looking at the strongest peaks. Second, the overtones in a musical sound generally present as ripples on top of a carrier wave, where the carrier is the fundamental. Again, the comparator should be drivable on this. Third, even if there are extraneous frequencies present in the strobe light, they will simply obfuscate the proper image slightly without breaking the function. In the

previous background section, it was stated that if the strobe light were left continuously on, then the resulting image would appear as a blur behind the disk to an observer. Similarly, some kind of white noise input to the comparator would also look like a blur. But, if the fundamental frequency is also present at some appreciable magnitude, then the observer should see mostly a blurry image, with the proper static slot image faintly appearing within the noise. It would still work the way it needed to, the image would just not be very clear. The object of filtering the input is to reduce this image noise by isolating the fundamental frequency as much as possible.

Generally, the overtones that make up the timbre of a sound are mostly of higher order than the fundamental. Lower frequency overtones do exist (the "thump" in a bass, for example), but they are usually low in magnitude and should not obfuscate the strobe disk slot image very much on account of being slower than the fundamental. Then, we are generally looking to low-pass filter (LPF) the input. But because the target frequency range (27.5 ~ 3000 Hz) is so broad, it is not enough to simply apply one filter for all pitches. After all, the range includes A0's *hundredth* harmonic!

We chose to split the range into two sections – those above and below A4, 440 Hz. The reason for this is because it is a well-known basis pitch for musicians and universally understood.[1] Labeling a switch on the tuner "> A440" will be immediately useful to the user. Filtering at 440 Hz will provide adequate filtering for all notes below it. Finally, a guitar's highest string is E4, 329.6 Hz. In most use cases, frequencies below 440 Hz should be plenty for most people.

The final filtering circuit, with the LPF switch in the < 440 Hz position, has the following response at full sensitivity:

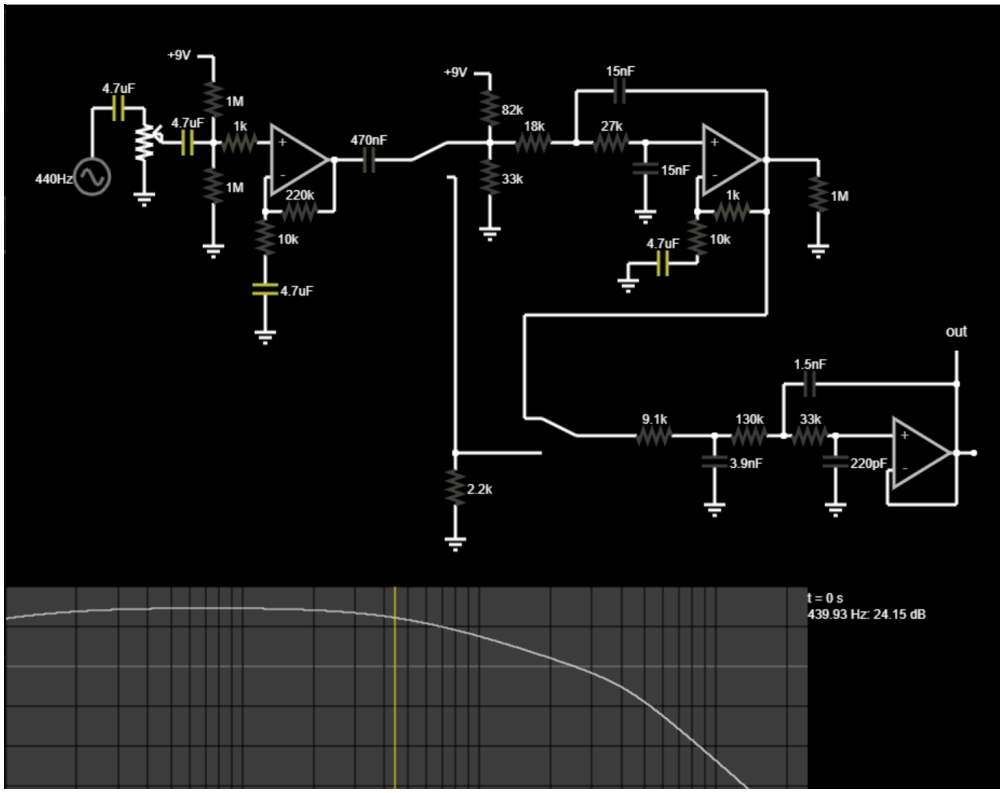


Figure 6. Audio processing circuit, with additional LPF switch engaged.

[1] When you hear an orchestra tune at the start of a classical performance, the note that they are all playing together is A4.

The first op-amp and surrounding components form a non-inverting amplifier with up to 26.8 dB of gain. Combined with the sensitivity potentiometer at the input, the purpose of this stage is to provide a high input impedance buffer to receive either direct feed from an electric instrument or the output from a condenser microphone preamp, and to amplify or reduce that signal as much as needed to drive the comparator at the end of the chain.

The second op-amp block filling the upper-right of the schematic forms the additional LPF for notes < 440 Hz. The circuit is a first-order Sallen-Key LPF set to the cutoff frequency 440 Hz. It forms the knee after the yellow marker in the Bode plot. It has about an 8 dB/decade slope. There is also a 1M load resistor on the op-amp's output, which ties the op-amp in place when the switch is off. In a quad-amp package, it is especially important to keep op-amps tied to DC when they are not in the signal path to avoid crosstalk and noise.

The last op-amp block is the main LPF, which aggressively cuts all frequencies above 4 KHz, which is right above the end of our target frequencies. It is a third-order Sallen-Key active filter with an approximately 30 dB/decade slope when combined with the additional LPF. Its knee appears in the Bode plot around 4KHz and is noticeably sharper than the slope below it.

For pitches > 440 Hz, flipping the additional LPF switch off produces the following response:

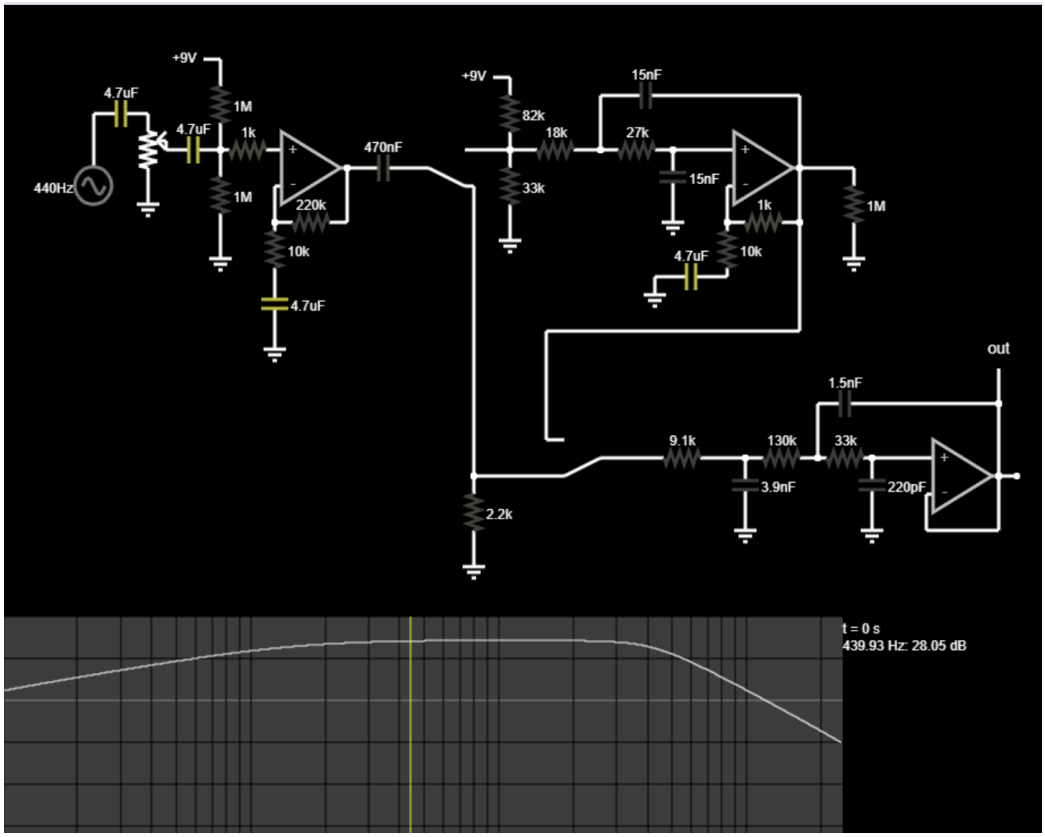


Figure 6. Audio processing circuit, with additional LPF switch disengaged.

The addition of the 2.2k resistor in the bypass path creates a 6 dB/decade, passive first-order high-pass filter that helps tame lower-order harmonics, if present. Without the additional LPF, the slope of the third-order filter in the last block is only about 16 dB/decade – but this is plenty for what it needs to do.

You may notice some biasing networks appearing in the first and second op-amp blocks. Because the project is powered by a single-ended 9V DC supply (as is standard practice), offset biasing is required to maintain the signal between the 0V and 9V rails. It is absent in the third block because it is provided by the second when the LPF switch is engaged, and the rail-to-rail action for frequencies well above 440 Hz was shown to be advantageous in simulation. An example screenshot of the full simulation, with the LPF switch engaged, is shown below:

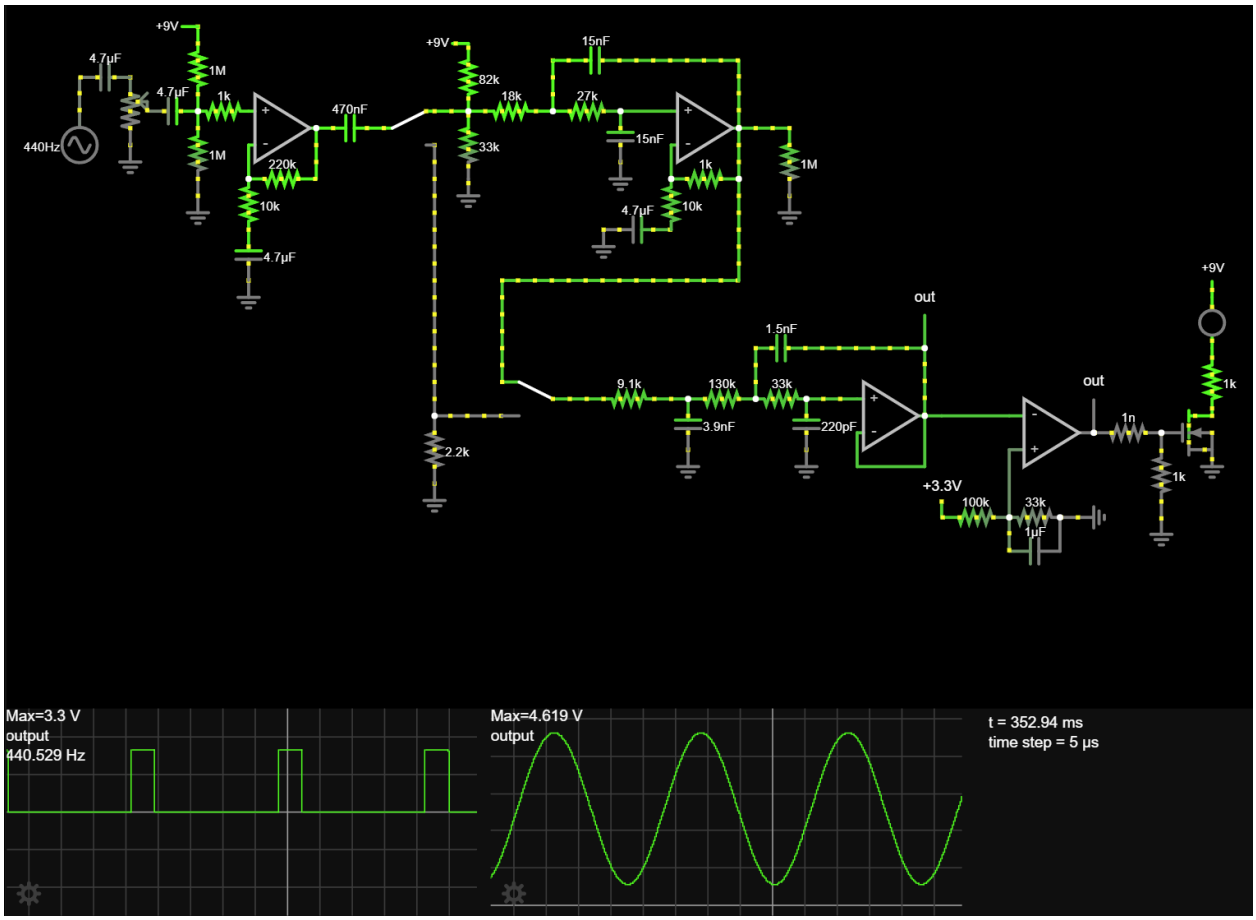


Figure 7. Audio processing simulation with the additional LPF switch engaged.

The simulation contains the filtering circuit from the filter analysis applet with the addition of a comparator and a MOSFET LED driver. The scope on the right is reading the output of the filtering and the scope on the left is reading the output of the comparator. The comparator is peculiar in that the input is attached to the (-) terminal of the op-amp, and the compare level is attached to the (+) terminal. The rationale for this is due to the fact that the audio runs on 9V rail-to-rail and the comparator is constrained to 3.3V due to the STM32's logic level. We would not be able to trigger the comparator at any level above 3.3V; the op-amp would not work or it would be incompatible with the STM32. So, we chose to trigger the output when the input signal dips below a threshold; that is, in the *troughs* of the audio wave. We added a trimpot to the audio partition of the circuit board in place of the simulation's fixed resistors to allow for adjustment of the trigger level to whatever worked best and provided the best output duty cycle. This does require that the audio be driven nearly to the rails, so a rail-to-rail compatible quad op-amp IC is required for the filtering circuits.

When the LPF switch is disengaged, the simulation looks like the following:

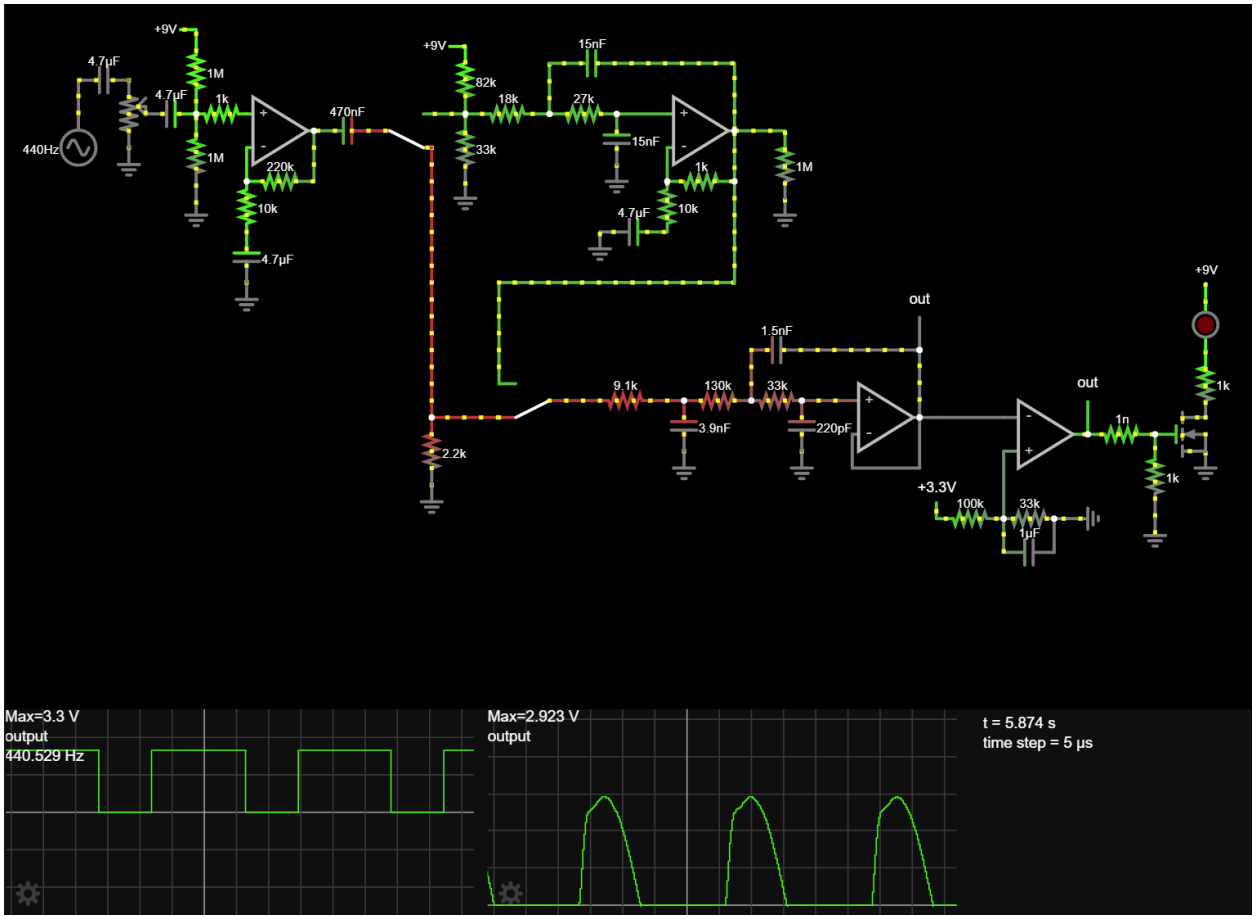


Figure 8. Audio processing simulation with the additional LPF switch disengaged.

The right scope shows that, in the absence of a bias network on the third stage, the op-amp is driving from the 0V rail. This leads to a flipped comparator output and increased duty cycle. One oversight we made at the design stage is that, without a bias network, the strobe lights are switched on continuously in the absence of a signal because the filter output rests at 0V. In the second version, we need to either add a bias network to the bypass path or switch the comparator logic when the additional LPF is disengaged.

The only portion of the audio circuit missing from the simulation is the microphone, microphone buffer, and the switched input jack. They were not necessary in simulation so long as the input was assumed to be high impedance and low magnitude (worst-case). The microphone buffer uses an op-amp IC expressly designed for FET condenser mics, the MAX4465, and the circuit was copied directly from the datasheet with the addition of a trimpot to the feedback network to adjust the gain on the board. The full schematic is attached to the Appendix.

Power Supply & MCU Requirements

We decided early in the project to split the project across two circuit boards – an “interface board” to handle all of the audio processing, and an “MCU board” to handle the MCU and motor driving. That way, we could keep noisy signals away from one another. Better yet, the LPF switch and the LED display

could be directly mounted to the interface board and the faceplate of the tuner case, and the MCU board could be kept in the back of the case for short cable runs to the motor.

Each board has separate power requirements. The interface board needs 9V to drive the display LEDs and the audio circuitry. This is provided directly by a regulated 9V DC supply, commonly known as a “wall wart” or “one-spot.” These are common fare among electric instruments. In terms of current requirements, the biggest consumer was going to be the display LEDs, each limited to 20mA through a control resistor on the display driver. This accounts for 16 × 20 mA = 320 mA when all LEDs are lit. However, the most segments any one note can need is 10 at a time, which makes 200 mA max. The audio ICs are not driving any significant loads and are limited to ~10 mA total.

The interface board also needs a 3.3V regulator for the comparator, the microphone preamp, and the LED display driver, which will pull no more than 40 mA altogether per their datasheets.We determined that an LDO-type regulator was acceptable even in the face of high drop voltage because the current requirements are so low. The total power dissipation across the LDO is (9 - 3.3) V × 40 mA = 228 mW. The maximum power dissipation allowed in the LD1117 regulator is 12 W, and the thermal resistance will create a 25°C temperature rise in the SOT-223 package.

The MCU board doesn’t use 9V for anything, but it needs 6V to drive the motor, motor driver, and RC receiver, and 3.3V for the logic level components and MCU. For the 6V, we chose an L7806 fixed LDO regulator in D2PAK that has a maximum output current of 1500 mA. The motor curve shows a maximum draw of about 700 mA, and the MP6550 driver IC only draws a few mA during operation. The current draw of the RC receiver is not published, but we assumed it to be less than 100 mA, worst case. In terms of heat, the total dissipation in the L7806 should top out at (9 - 6) V × 800 mA = 2.4 W, which corresponds to a 150°C temperature rise in the D2PAK package. While this is worst-case (and *highly* conservative, because a PWM-driven motor should draw a lot less current), we were still concerned about thermal issues in the regulator. We made sure to add as much copper to the 6V pours as possible, with loads of thermal stitching vias, and even included a heatsink:

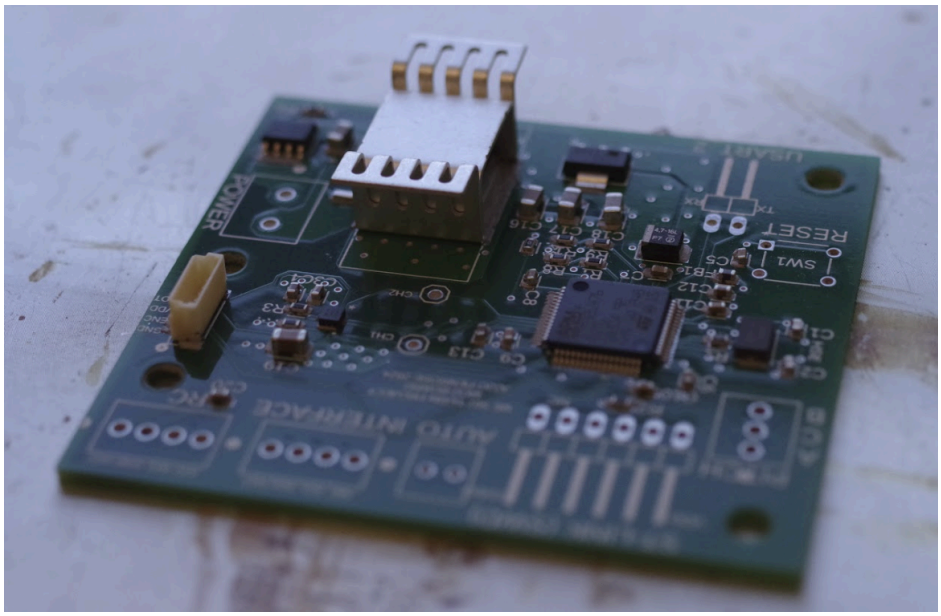


Figure 9. MCU board with heat sink visible.

The heat sink simply gets soldered to the 6V pad and adds thermal mass for dissipation. It isn't designed for passive use; it's oriented horizontally for a fan, but we figured it was enough insurance to keep our chip within safe temperatures. We also made sure to place components that dissipate a lot of heat as far as possible from the MCU crystal.

We used the same LD1117 LDO 3.3V regulator for everything using logic level. According to the STM32L476xx datasheet, the MCU using all peripherals uses, at most, 112 µA/MHz. Using an 80 MHz clock, this leads to 112 × 80 MHz = 8.96 mA max. This is really the only considerable current draw through the 3.3V regulator, so we expect to dissipate no more than (6 - 3.3) V × 10 mA = 27 mW. We are at more risk of damaging the regulator due to heat from the nearby 6V regulator than through the 3.3V rail draw.

Finally, the only other calculation we needed to do was for the high-speed external (HSE) crystal. We followed the STMicroelectronics Application Note 2867, "Guidelines for oscillator design on STM8AF/AL/S and STM32 MCUs/MPUs," to direct our crystal design. We chose the 8 MHz ABM3-8.000MHZ-D2Y-T based on their recommendation and used their formulae to calculate the correct load capacitors.

From the ABM3-8.000MHZ-D2Y-T datasheet:

$$C_L = 18 \text{ pF}$$

$$C_0 = 7 \text{ pF}$$

$$\text{ESR} = 140\ \Omega$$

$$f = 8 \text{ MHz}$$

From the STM32L476xx datasheet, page 150:

$$C_S \approx 10 \text{ pF}$$

From AN 2867, pages 12-13:

$$C_{L1} = C_{L2} = 2 * (C_L - C_S) = 16 \text{ pF}$$

$$g_{mcrit} = 4 * ESR * (2\pi f)^2 * (C_0 + C_L)^2 = 0.00088 \text{ mA/V}$$

From the STM32L476xx datasheet, page 150:

$$G_m = 1.5 \text{ mA/V}$$

From AN 2867, page 14-16:

$$gain_{margin} = \frac{G_m}{g_{mcrit}} = 1696 \text{ (gain margin should be > 5. Good!)}$$

$$R_{ext}(estimate) = \frac{1}{2\pi f C_{L2}} = 1243\ \Omega$$

This is the exact resistor value that filters out 8 MHz; so to avoid that, choose a lower standard value and recheck the gain margin.

Let $R_{ext} = 470\ \Omega$. Then:

$$g_{mcrit} = 4 * (ESR + R_{ext}) * (2\pi f)^2 * (C_0 + C_L)^2 = 0.00102 \text{ mA/V}$$

$$gain_{margin} = \frac{G_m}{g_{mcrit}} = 1473$$

So, we chose load capacitors of 16 pF and an external resistor of 470 Ω.

Materials & Methods

Printed Circuit Board Design

Interface Board

The schematics were drawn in Fusion 360. The schematic sheets are appended to this report. The printed circuit boards were created from these schematics. The interface board contains the audio processing circuits, the strobe light driver, the LED display, and its driver.

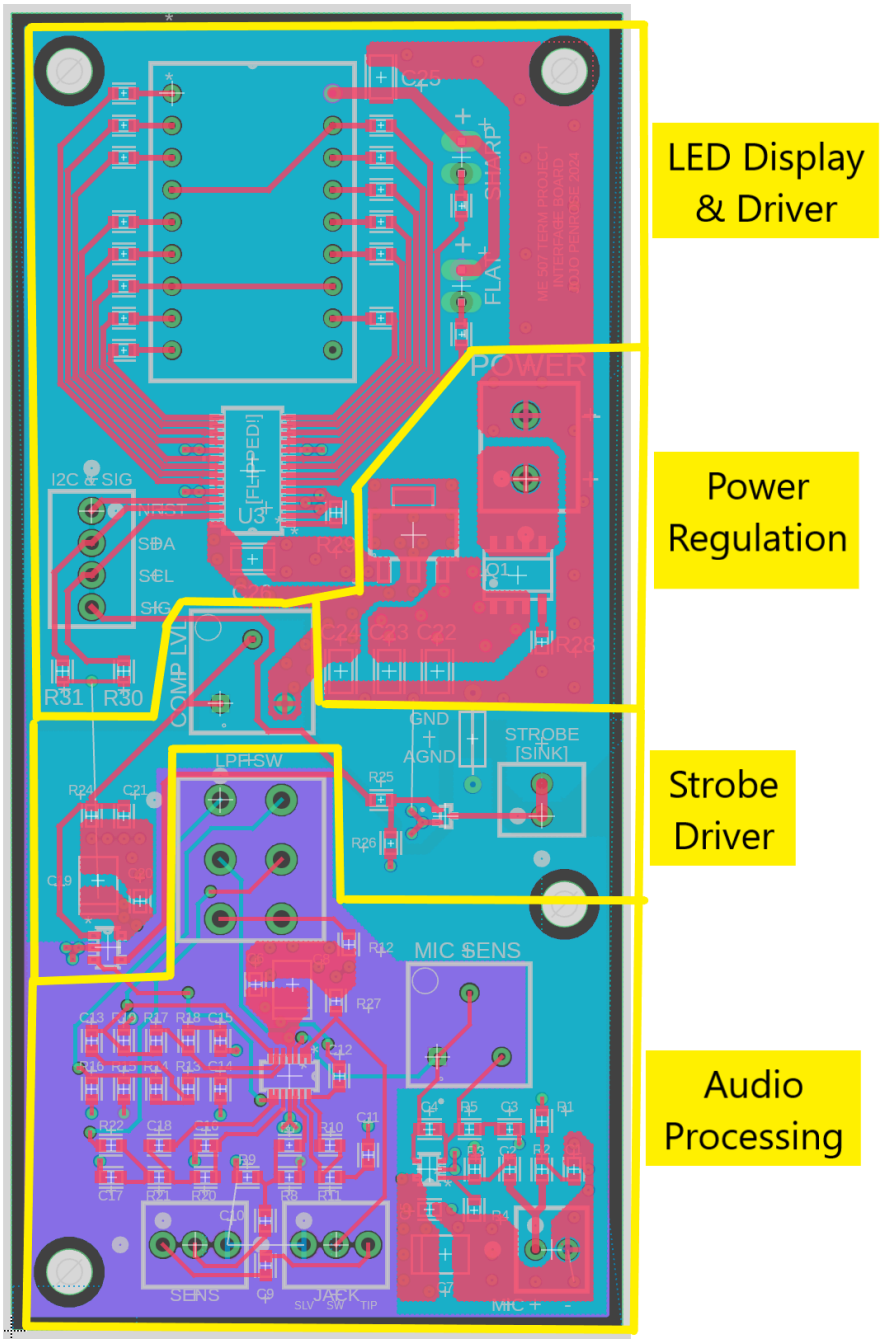


Figure 10. Interface board partitions.

The power regulation partition accepts 9V regulated DC voltage from a one-spot using screw terminals. Reverse polarity protection comes from a common P-MOSFET (FDS4435BZ) arrangement. An LD1117 LDO regulator provides 3.3V for the display driver and the comparator.

In the LED display & driver partition, the interface uses a 16-segment LED alphanumeric display (LTP-587HRLTP) for the note letter, and two LEDs to represent the sharp and flat symbols. The driver IC (TLC59116IPWR) is designed for driving a single 16-segment display. In order to handle the two extra sharp and flat LEDs, we left two of the 16 segments disconnected and used those two connections. Since the letters A-G don't need any of the diagonal segments, we could leave them out of the circuit. There is a screw terminal block for connecting the MCU board's I^2C lines, the reset logic connection for the driver IC, and the comparator square wave signal.

The strobe driver is considered separate from the audio processing partition because the op-amp that forms the comparator is essentially converting the analog signal to digital. A single op-amp IC (MCP6001) was used for the comparator, which will keep the sharp transients on the output separated from the filtering circuits. The op-amp is rail-to-rail compatible, as any comparator must be. A trimpot sets the threshold for the comparator. The output of the comparator drives the gate of an N-MOSFET (SSM3K56FS), which is used as a driver for the strobe LEDs. The strobe LEDs sink current through the MOSFET, and are connected through a screw terminal block.

The audio processing partition is by far the most complex. It contains the quad op-amp IC (TLV9154) that handles the filtering, the microphone preamp op-amp IC (MAX4465), the microphone gain trimpot, the additional LPF switch, and the screw terminal blocks for the microphone, input jack, and sensitivity potentiometer. The microphone preamp is specifically designed for FET condenser microphones like the one used for input (CMA-4544PF-W). The quad op-amp must be rail-to-rail compatible, as the last stage must be driven into near-saturation for the comparator to see the troughs and work correctly.

The interface board is a four-layer board.

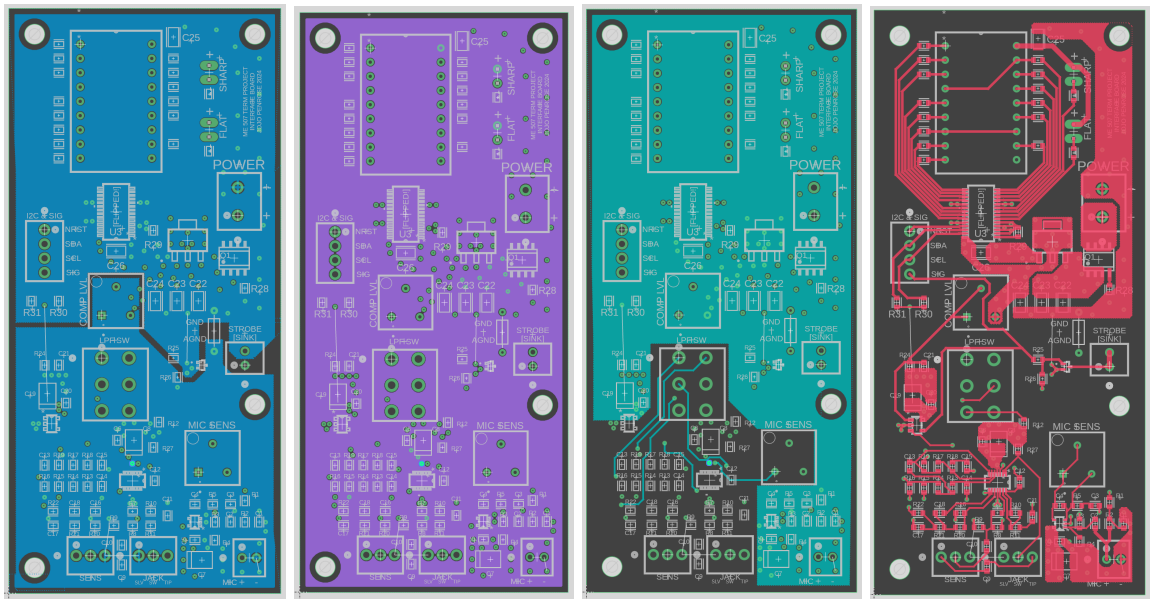


Figure 11. From left to right: bottom layer, third layer, second layer, and top layer.

The bottom layer carries the two ground planes. On the bottom is the analog ground plane, and on the top is the digital ground plane. A set of pads for an axial resistor was provided to jump the two grounds, but the final assembly uses star grounding to connect the two interface board grounds and the one MCU board ground to the barrel jack ground.

The third layer carries a large pour for the 9V plane.

The second layer is used for some extra routing in the audio processing partition and for the 3.3V plane. Note that the pour wraps around the audio partition, because it needs to only power the display components, the comparator, and the microphone preamp.

The top layer is the main routing layer. It also uses polygon pours for power connections where possible. Large traces were used to carry the LED currents.

Most of the components are surface-mount. All of the surface-mount components are on the top layer. For through-hole components, the switch and the LEDs were soldered on top of the board so that they would stick through the faceplate when mounted. The other parts, the screw terminal blocks and the trimpots, were soldered to the back of the board. That way, they would still be accessible after the PCB is attached to the faceplate.

Four clearance holes for M3 screws were added for mounting the board to the tuner.

MCU Board

The MCU board contains the STM32, the motor driver, the connections necessary for programming and debugging, and all of the I/O needed to run the project.

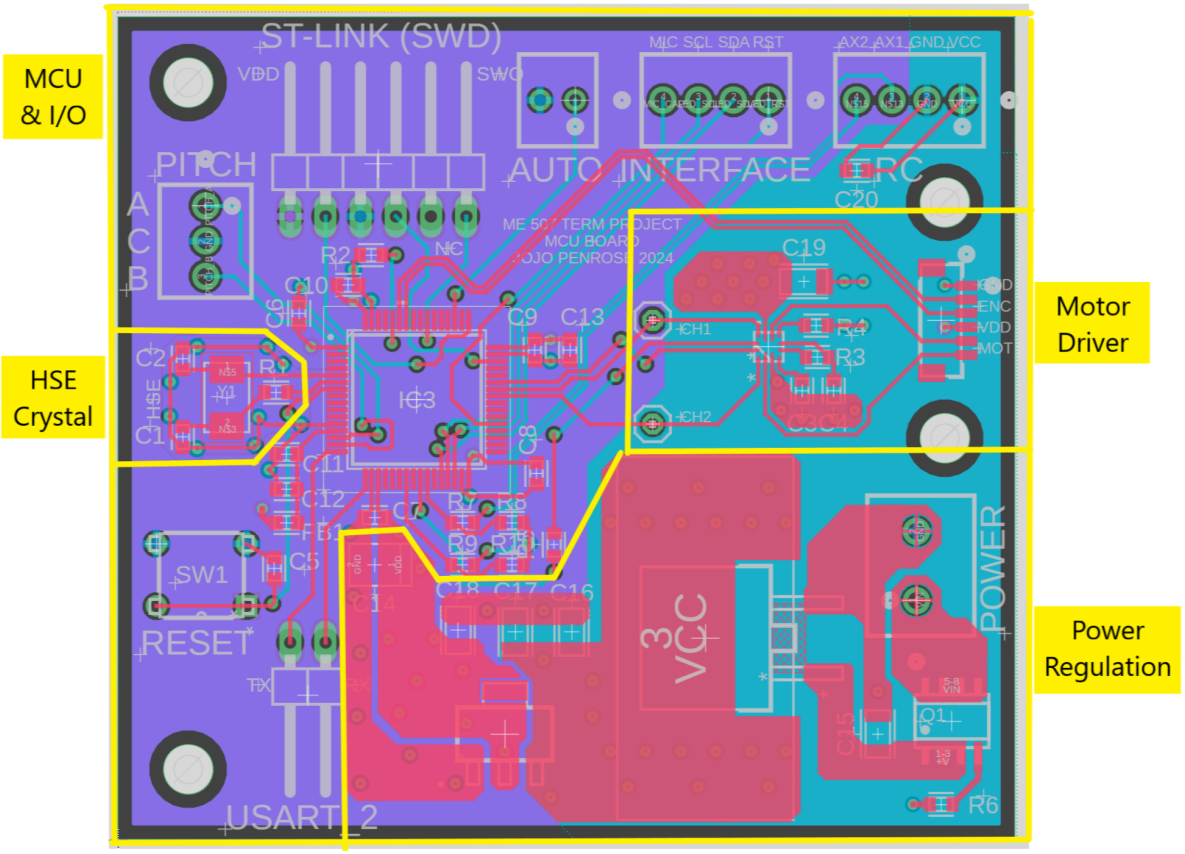


Figure 12. MCU board partitions.

The power regulation partition accepts 9V regulated DC voltage from a one-spot using screw terminals. Reverse polarity protection comes from a common P-MOSFET (FDS4435BZ) arrangement. An L7806 LDO regulator provides 6V for the motor and the RC receiver. The L7806 was given as much copper as possible to help dissipate heat under motor load, as well as pads for the soldered-on heatsink. An LD1117 LDO regulator provides 3.3V for the STM32 and all of the logic-level I/O.

The MCU & I/O partition contains the STM32 and all of its I/O connections. It also contains all of the necessary external components in a minimal MCU design. There are bypass capacitors and power connections on every power pin on the STM32, a pull-down resistor on the BOOT0 pin to select the right program memory, and a debounced reset button. For I/O, we have screw terminals to connect to the mechanical pitch encoder, the automatic pitch detection switch, the interface board, and the RC receiver. Because the RC receiver has to run on 6V (it requires > 5V, so native 3.3V was out of the question), resistor dividers are used to reduce the level to a safe voltage for the MCU. There are also male header pins for the ST-Link programmer and UART connection for debugging. These particular connectors were chosen because they match the ones on the ST-Link, and the connection is only temporary.

The HSE crystal was given its own partition because it is highly sensitive to noise and temperature. It was intentionally placed as far as possible from hot and noisy components; namely, the power regulation and

motor driver partitions. It also uses a special grounding scheme known as a “guard ring,” which is described in the “PCB design guidelines” section 7.1 of the STMicroelectronics Application Note 2867, “Guidelines for oscillator design on STM8AF/AL/S and STM32 MCUs/MPUs.”

The motor driver partition contains the motor driver IC (MP6550) and a six-pin male JST header for connecting the motor. There are also two pads for through-hole single-pin male connectors, used for debugging the motor control PWM signals.

Most of the components are surface-mount. All of the surface-mount components were soldered to the top layer. Through-hole components were all soldered to the top of the board, so that the PCB could be mounted to the back of the tuner case.

Four clearance holes for M3 screws were added for mounting the board to the tuner.

The MCU board is a four-layer board.

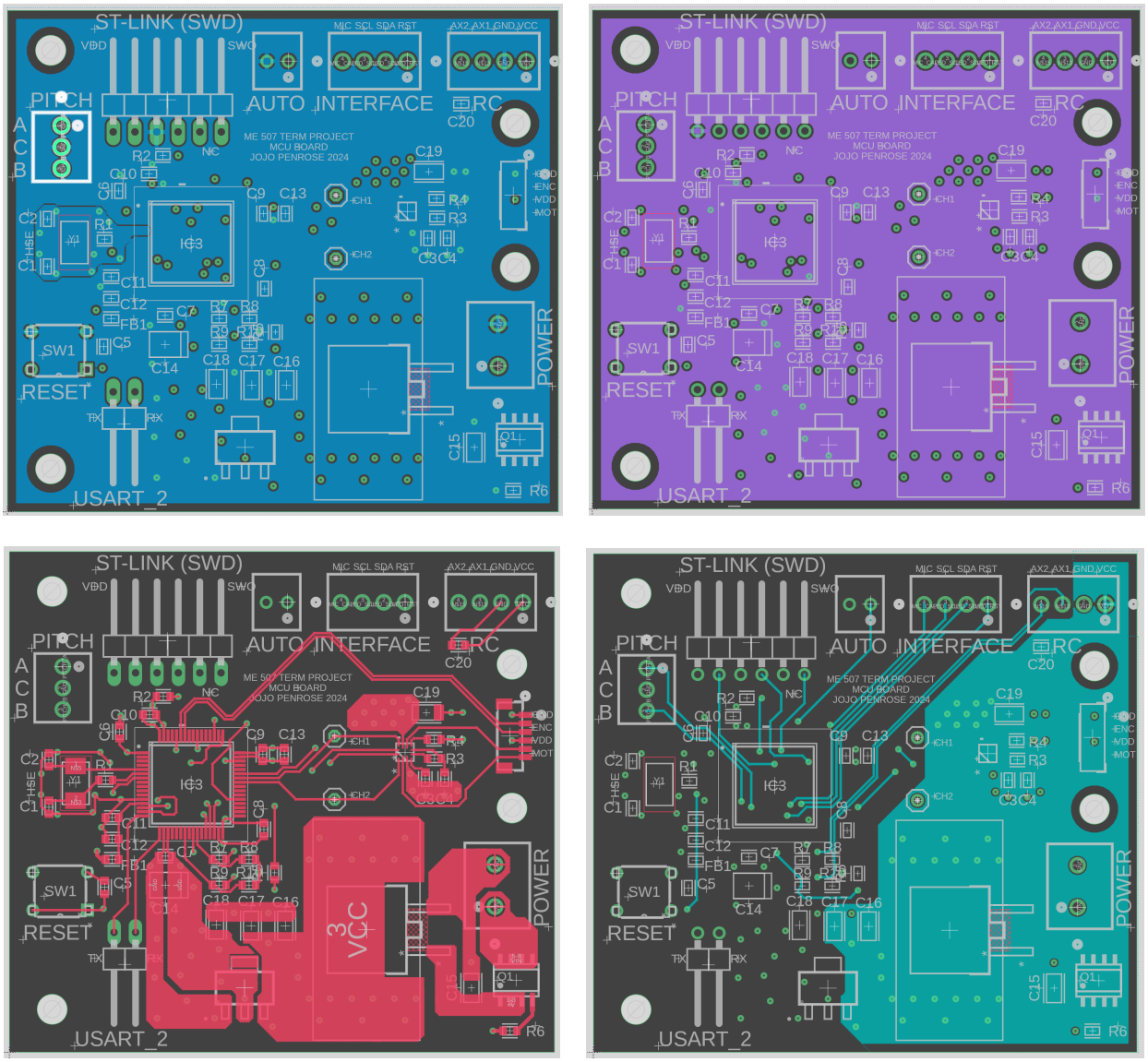


Figure 13. Clockwise from upper left: bottom layer, third layer, second layer, and top layer.

The bottom layer carries the ground plane. It also has a dedicated ground plane for the crystal.

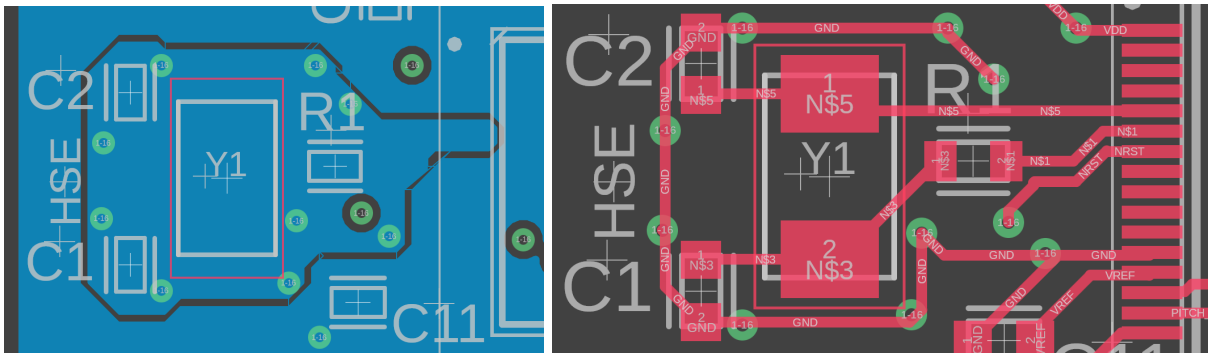


Figure 14. Detail view of the HSE crystal partition, bottom layer (left) and top layer (right).

The ground plane for the crystal is independent from the main ground plane. It is stitched to the guard ring on the top layer through several vias, and the guard ring is directly connected to an MCU ground pin. The connection to the main ground net is made at the pin. STMicroelectronics recommends this arrangement to protect the crystal from noise.

The third layer carries a large pour for the 3.3V plane.

The second layer is used for extra routing for the MCU and I/O traces. It also has a copper pour for the 6V plane. That pour was made as large as possible to help pull heat out of the driver and the regulator.

The top layer is the main routing layer. It also uses polygon pours for power connections where possible.

Both boards were assembled and soldered using a hot plate for the SMD components and using the laboratory soldering stations for the through-hole components.

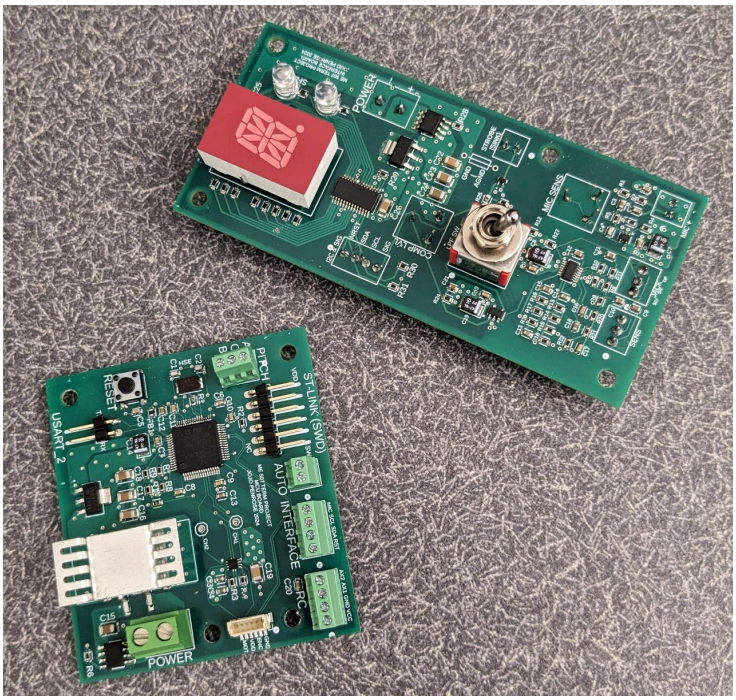


Figure 15. Assembled printed circuit boards.

External Electronics

The strobe tuner has a lot of user-customizable parameters. As such, there are a number of external knobs and switches. Connected to the interface board are the FET condenser microphone (CMA-4544PF-W), the 1MΩ audio-taper sensitivity potentiometer (PDB181-E415K-105A2), and the switched mono ¼" phono jack (Switchcraft 12A). It also connects the current sink for the strobe LEDs.

The strobe LEDs are driven directly off of the 9V external DC supply, and sink their current through the driver MOSFET. Several arrangements of LEDs were considered, and we ended up choosing eight LEDs for even coverage behind the strobe disk.

Figure 16. Strobe LED plate.

We knew that the resistance for the right amount of LED current was ballparked around 50~150Ω, so we purchased several 36Ω 2W resistors so that we could arrange them in series and parallel combinations to achieve whatever resistance we needed. We also included a 50Ω 5W wirewound linear-taper potentiometer (026TB32R500B1A1) so that the brightness could be adjustable.

We knew from the LED (C503B-ACN-CY0Z0341-030) datasheet that we wanted around 20 mA through each LED for low brightness and around 40 mA for high brightness. Going over 40 mA would risk damaging the diodes. If we split the LEDs into two segments, where the two segments are in series and the LEDs within each segment are in parallel, then we end up with this circuit (assuming the MOSFET is a perfect short when it is conducting):

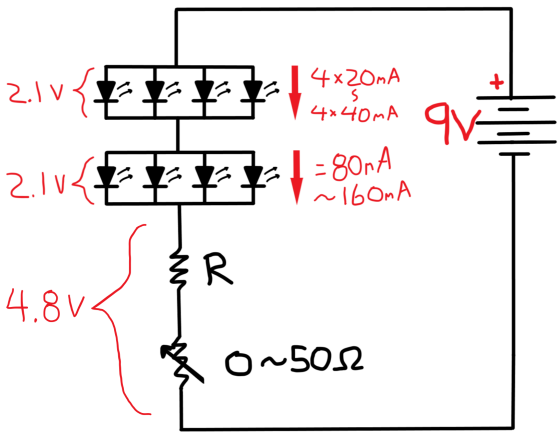


Figure 17. Strobe LED circuit.

By adding the LED forward voltage drops and desired currents, we determined that the current-limiting resistor value we needed was described by:

$$4.8V = 80mA * (R + 50) \Rightarrow R = 10\Omega$$

$$4.8V = 160mA * R \Rightarrow R = 30\Omega$$

We could choose the lower value of R and get 20mA/LED for dim current, but we would get a whopping 120mA/LED when the potentiometer was completely off – a surefire recipe for cooked diodes. Instead, we chose 30Ω for R to achieve 40mA/LED on full brightness and 15mA/LED on low brightness, which offers a good range in brightness without risking burnt or non-lighting LEDs at any setting. To get 30Ω from our 36Ω resistors, we connected two bundles of resistors in series where one bundle had three resistors and the other bundle had two, which results in a resistance calculation of:

$$R = (36/3) + (36/2) = 12 + 18 = 30\Omega$$

Under this arrangement, the worst-case resistor in the two-resistor bundle sees $160/2 = 80$mA through it, which amounts to $0.080^2 * 36 = 230$mW, well below the 2W rating. Likewise, the potentiometer will dissipate no more than $0.080^2 * 50 = 320$mW, which is nothing compared to its 5W rating.

The MCU board has connections for the external RC receiver (provided by ME 507), the auto pitch selection switch (SPST mini toggle), the mechanical pitch encoder (288T232R161A2), and the motor (Pololu #5183, 4765).

The RC receiver must run on > 5V, so it is supplied 6V and ground from the regulator on the MCU board. The first two axes are wired back to the MCU. The first axis, the trigger, is intended for a dead-man switch. The second axis, the steering wheel, is intended to be an alternative pitch selection method. These features would be absent from a final product, but were required under the ME 507 term project parameters.

Tuner Case & Assembly

The tuner case was designed for 3D printing in Fusion 360. By keeping the mechanical design in the same program as the electronics, it made it easy for us to push the PCB to 3D and use it in our design. The mounting hole locations and the locations and dimensions of the display LEDs and LPF switch were referenced directly from the 3D PCB in Fusion.

We started by modeling the strobe disk. For a personal use, benchtop unit, we determined that a strobe disk, 4.5” in diameter, having 0.5” wide strobe rings, was a good starting point for a prototype.

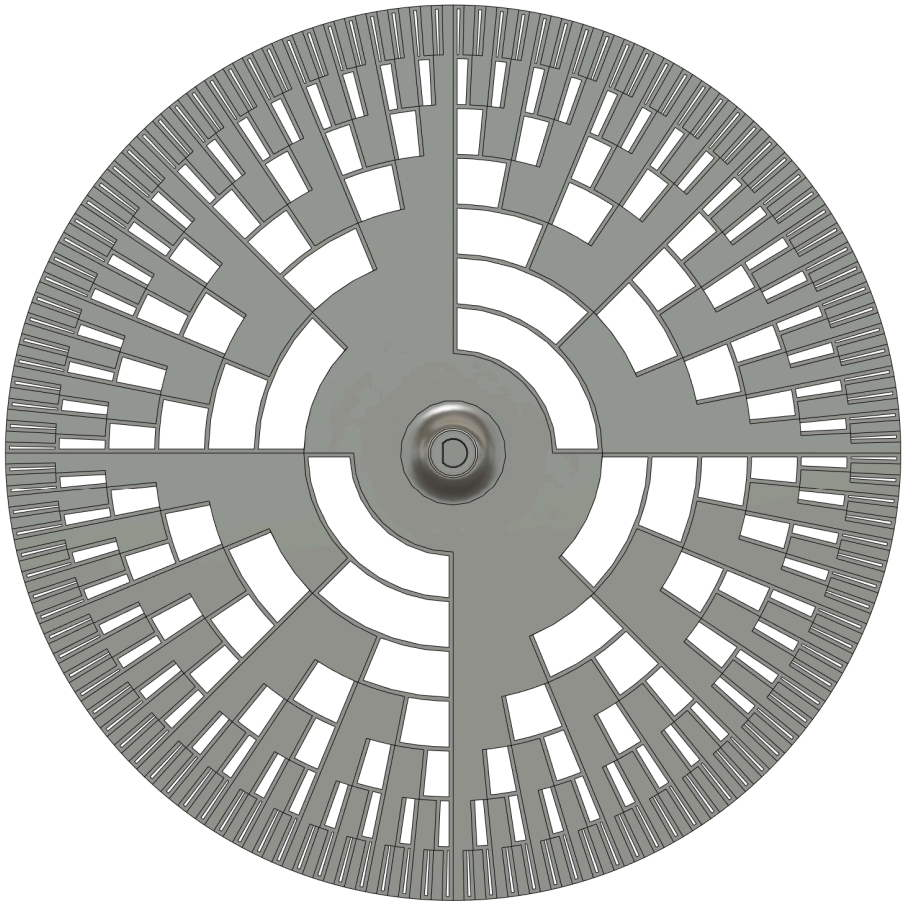


Figure 18. Strobe disk model.

Small wings were needed to connect the empty space between slots for structural integrity, no more than three 3D print layers thick. The closed areas of the disk are only 0.070” thick, in order to cut down on inertia as much as possible and ensure that the tiny motor would get up to speed without issues. There is a 3mm flatted-shaft hole to accept the motor shaft in a friction fit.

We designed the strobe tuner unit to sit on its own as a benchtop tool. It was given a slight backwards tilt for stability and viewing, and the overall profile was designed to be somewhat aesthetically pleasing. The strobe LED plate, shown previously, attaches to the inside of the tuner case via countersunk flat-head M3 screws and 3D printed threads. The inside of the case also has a mounting point for the motor using M1.6 screws, M3 threaded mounting points for the MCU board, cutouts in the side for the power connector, power switch, and strobe brightness pot, as well as some cable management loops against the back wall.

Figure 19. Strobe tuner case model.

The faceplate attaches to the case by the four countersunk flat-head M3 threaded holes in the case. The faceplate has a cutout for the LED display, sharp/flat stencils for the indicator LEDs, a functional and decorative cutout for the microphone, and mounting holes for the LPF switch, auto pitch detection switch, sensitivity knob, pitch encoder, and input jack.

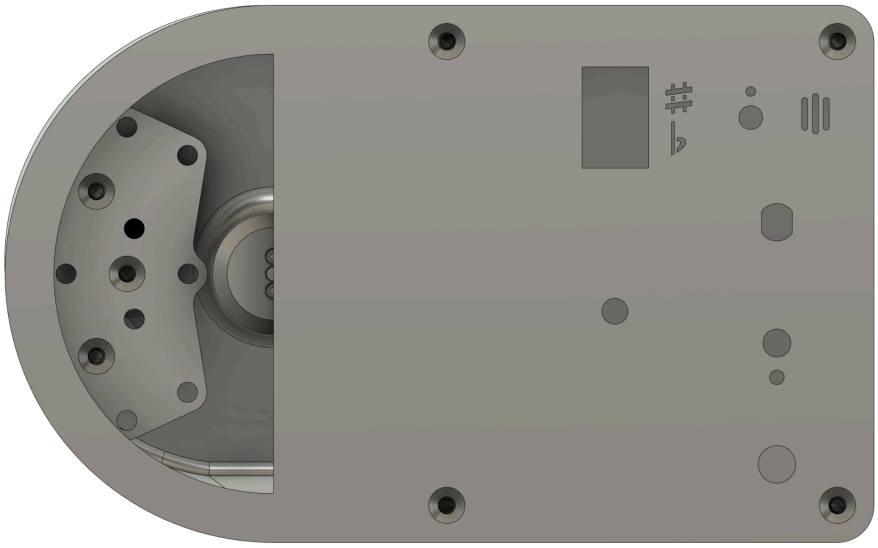


Figure 20. Strobe tuner faceplate model.

The rear of the faceplate has M3 threaded attachment points for the interface board, sleeves that isolate the sharp/flat LEDs from one another, a hole to carry the microphone in, and some cable management loops.

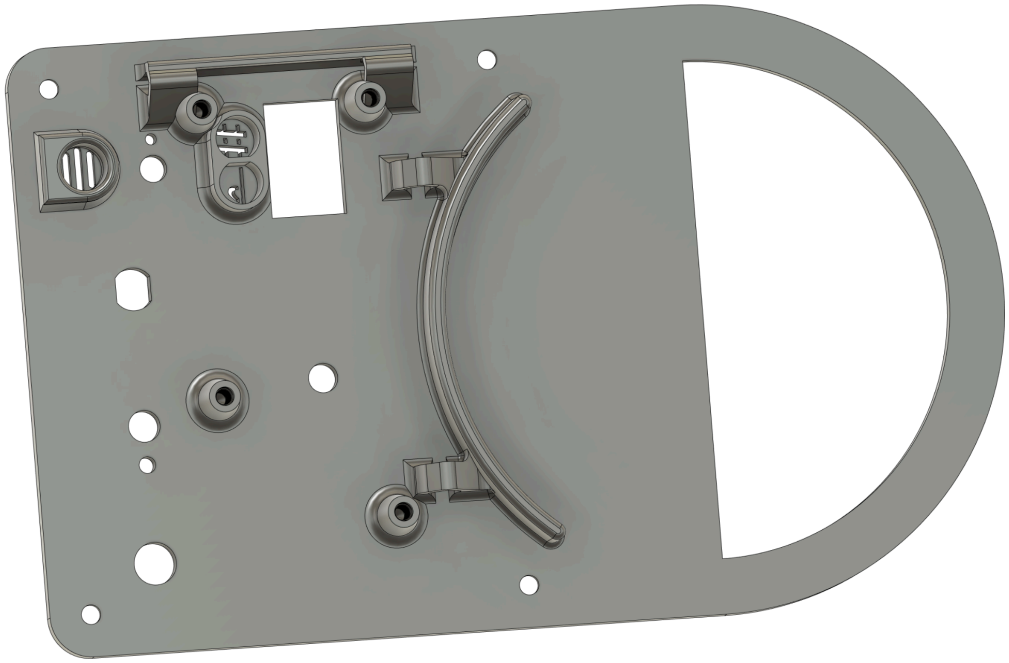


Figure 21. Rear of the strobe tuner faceplate.

The strobe tuner parts were printed and the tuner was assembled and wired using the complete circuit boards. Solid core wire was used for off-board connections because the stiffness kept things in place while working, and after final assembly, should not be expected to move anywhere.

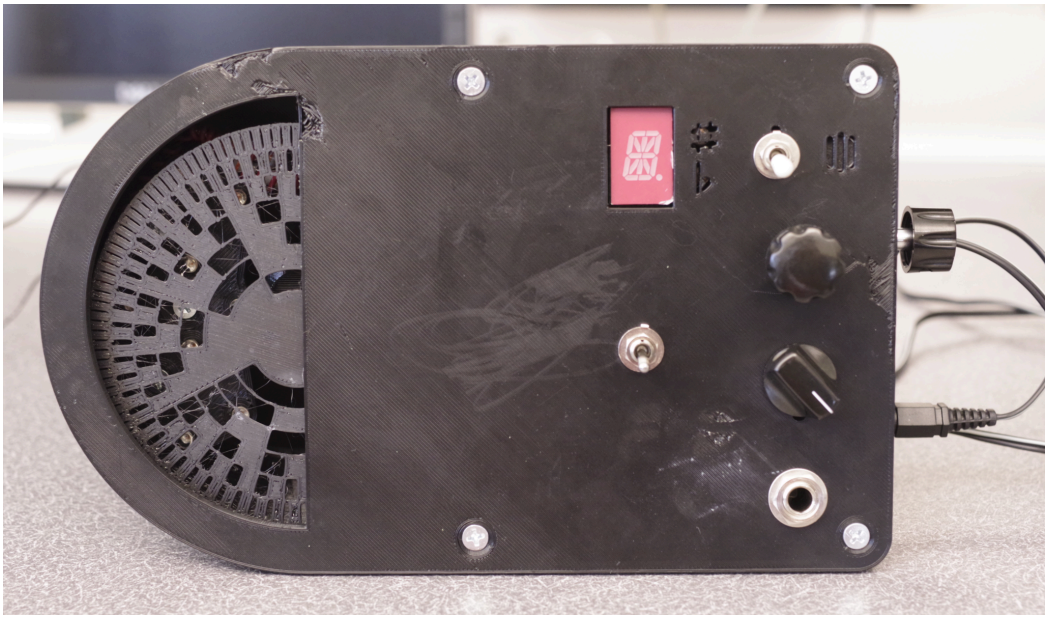


Figure 22. Assembled prototype, front view.

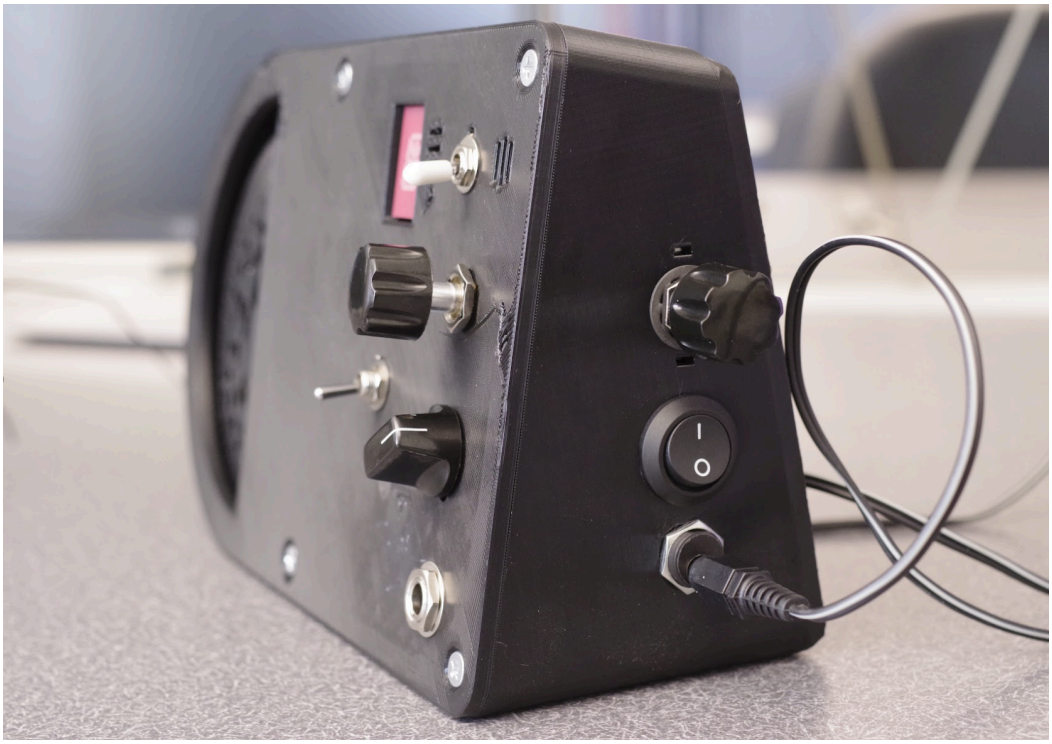


Figure 23. Assembled prototype, showing side components.

When assembled, the cable management loops help keep the wires roughly under control when the faceplate is detached from the case. Certain connections, such as the power and communication lines between the two boards, are allowed to float and fill the space between the two boards when the faceplate is closed.

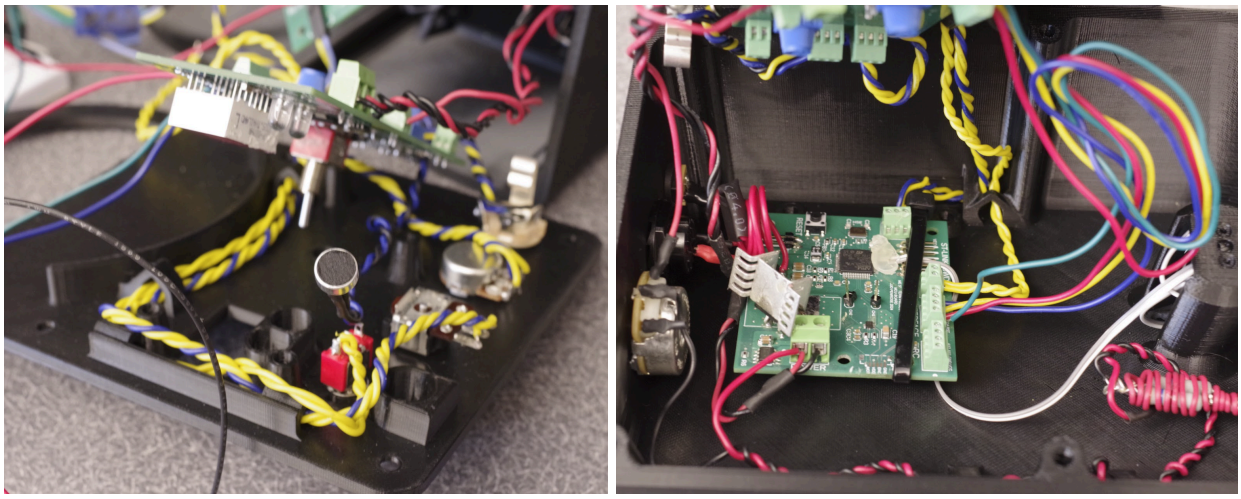


Figure 24. Detail views of the cable management.

The strobe LED resistor bundles were wrapped in wire to keep them stiff and tucked behind the strobe disk, using provided cable management loops to hold it in place. The motor cable, when installed, is short enough that it spans the board to the motor without veering too far from the shortest path, and needs no extra managing.

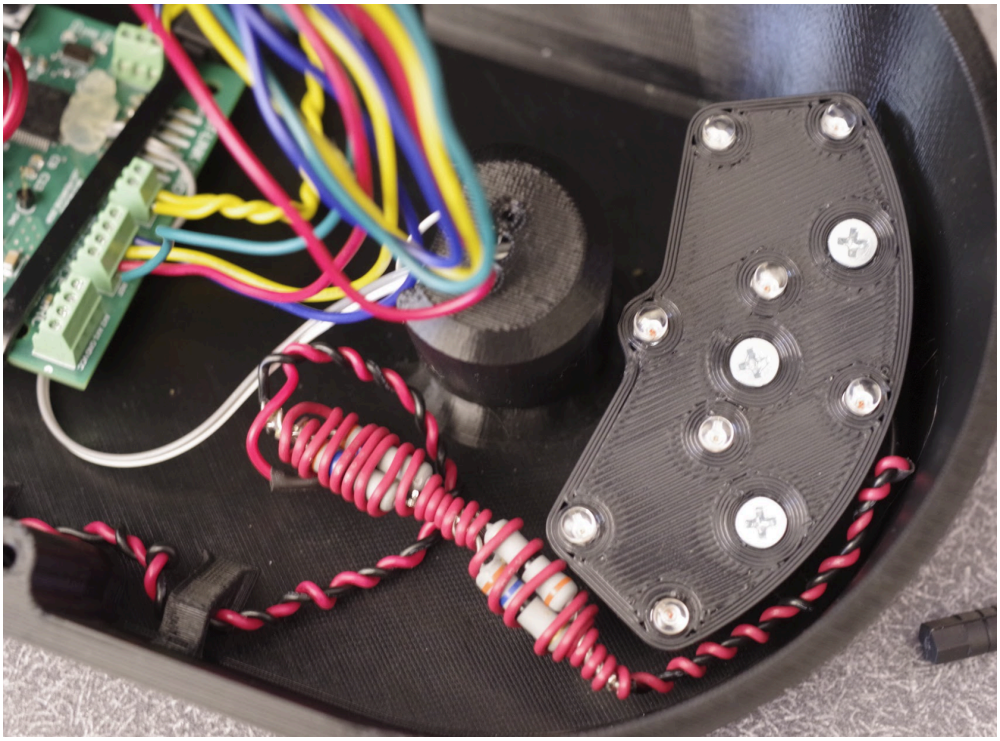


Figure 25. Detail view of the strobe LEDs and resistors.

Firmware & Software

Setup

Our microcontroller was coded in C using the STM32CubeIDE. This allowed for simple configuration of the GPIO pins, timers, and communication protocols. For the original design of this project, we planned to use six timers, three GPIO pins, an I^2C bus, a UART Communication Protocol, and Serial Wire Debugging communication. A table of the configured pins and their functions can be found below in Table 1.

Table 1. Pin and Function Mapping

Category	Pin	Function	Description of Purpose
Serial Wire Debugging	PA13	SWD I/O	Serial wire Debugging via ST Link
	PA14	SWD Clock	
	PB3	SWD SWO	
GPIO	PA10	Output	Motor Enable Pin
	PB2	Output	Display Enable Pin
	PC10	Input	Auto/Manual Switch input
I2C	PB10	I2C2 SCL	I2C2 Bus for running the 16 segment display driver
	PB11	I2C2 SDA	
UART	PA2	UART2 TX	UART for communicating to computer for debugging purposes
	PA3	UART2 RX	
ADC	PC5	ADC1 IN14	ADC for current sense of the motor
Timers	PA8	TIM1 CH2	Motor PWM Channels
	PA9	TIM1 CH1	
	PA7	TIM3 CH1/CH2	Input Capture for the trigger of the RC Controller (Used for E-Stop)
	PB0	TIM3 CH3/CH4	Input Capture for the wheel of the RC Controller
	PB6	TIM4 CH1	Motor Encoder Quadrature Channels
	PB7	TIM4 CH2	
	PA0	TIM5 CH1	Pitch Knob Rotary encoder Quadrature Channels
	PA1	TIM5 CH2	
	N/A	TIM6	Internal timer for speed calculations
	PC6	TIM8 CH1	Input Capture for microphone
Crystal Oscillator	PH0	RCC OSC IN	Crystal Oscillator pins
	PH1	RCC OSC OUT	

These pins were configured in the IOC for the STM32CubeIDE Project. A picture of the IOC can be found below in Figure 26.

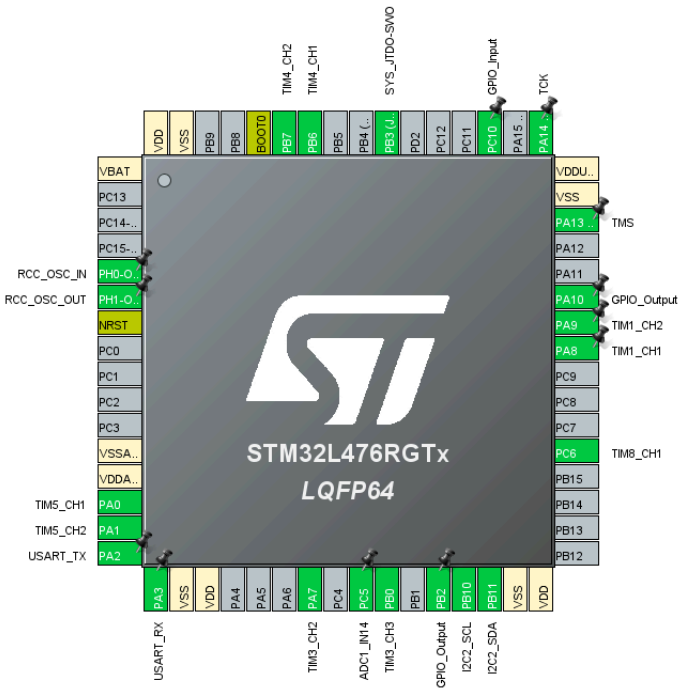


Figure 26. IOC setup for the STM32L476RGT6.

The GPIO, UART, and Crystal Oscillator were left as default settings for their respective functions, but the Timers, I2C, and ADC all had some modifications to their base settings. Timer 1, used for the motor PWM, configured the timer 1 channels 1 and 2 pins for PWM output, with a prescaler (PSC) of zero and an Auto-Reload (ARR) of 999. This runs the motor PWM at 80 kHz (above audible and below the driver maximum of 10 kHz) and allows for the desired pulse width percentage to simply be multiplied by 10 and applied to the timer PWM output for control of the motor. Timer 3 configured all four channels to be in input capture mode, with channels 1 and 4 being indirect capture and 2 and 3 being direct capture. This configuration is able to track both rising and falling edges of the pulse width created by the radio receiver for control with the RC controller. Timer 3 is set up to be able to generate interrupts, which allows for non-blocking calculation and tracking of the RC controller inputs. Channels 1 and 3 are configured to track and generate interrupts on rising edges, and channels 2 and 4 track and interrupt on falling edges. The timer was configured with a PSC of 79 and an ARR of 65535. This allows for the count of the timer to be in units of microseconds, allowing for easy calculation of the RC signal pulse widths. Timer 6 follows a similar setup of PSC/ARR, which allows for simple calculations where the time is in milliseconds.

Continuing with the setup, timers 4 and 5 were both set up in Encoder Mode, which is a built-in function to count up and down pulses on a quadrature encoder. The motor encoder, on the 16-bit Timer 4, has a PSC of 0 and an ARR of 65535 to allow for the most amount of ticks possible before overflowing. The pitch knob encoder uses a 32-bit timer (an unnecessary range, but the pin placement on the MCU was convenient), and has a PSC of 0 and an ARR of 1073741823. This is a quarter of a full maximum 32-bit number, due to overflow issues with numbers closer to the full 32-bit number. Both timers were set up to track pulses on channels 1 and 2, which allows for full quadrature counting. Timer 5 had the additional setting of enabling the internal pull-ups on the pins. The pitch encoder knob does not have power wires

going to it, so internal pull-ups to feed the encoder power for the quadrature mode are required. For the final timer, Timer 8 was set up for input capture in a similar manner to Timer 3, but this time on only one channel with interrupts generated on rising edges only. Using the comparator, a wave's period can be simply tracked by measuring the time between the last two rising edges. This, when inverted, can simply be used to find the frequency of the current signal.

For the final bits of setup, the ADC (used to measure the motor current, for safety) uses single-ended ADC, since the current sense only requires a reference to ground. And finally, the I^2C pins set the clock frequency to 20 kHz for powering the display. This was done because the pull-ups used on the data and clock lines were too large, and were causing issues that will be discussed further later. Reducing the frequency of the clock helped a bit to remedy the issues, but did not fully solve all of the problems.

Structures

Since we had already been coding in C and had not used C++ before the project, the team chose to continue coding in C. This allowed for an easier ramp up of coding, but became a challenging obstacle later on. Our team comes from an object-oriented programming (OOP) background, and so we wanted to continue to follow an object-oriented framework for its many advantages (data organization, code understandability, etc.), but C does not have objects, which made OOP more challenging. Our team made do by approximating classes with structures.

That being said, our team implemented the following "classes": a Closed-Loop Controller class, a base Encoder class, a Motor class, a Display Class, and a Pitch Encoder Class.

The CLController class implements a full proportional-integral-derivative-feedforward (PIDF) controller. It takes in kp, ki, kd, and kf values (multiplied by 10^6 for rounding purposes), a setpoint, an output effort, a current measured value, the error, the accumulated error, and time information for derivative calculations. The structure has two methods: run(CLController* con, int32_t measured) and reset_controller(CLController* con). The former calculates the required effort with a PIDF controller. Proportional, integral, and feedforward are calculated in a standard way for a discrete-time controller. Derivative implements a very simple impulse filter in an attempt to reduce the effect noise has on the derivative control. It does this by storing the last n number of items in a list (n is chosen by the user), and the slope between the first and last item in the list is calculated with respect to the time in milliseconds (from HAL_GetTick()). This does not actually filter impulse per-se, since if there is an impulse on the first or last value in the list, it still accentuates this impulse, but future work would include calculating the slopes between each of the values in the list, and take an average. But, since our team did not even end up using derivative control for our motor, no further action was taken to improve the derivative control. The function returns the effort calculated by the controller (or just the value calculated by the controller in whatever units the user sees fit). The other function reset_controller() does as it says, it resets the controller to an initial, zero state.

The CLController class is used for speed control in our project, and takes values from the Encoder instance for our motor. The Encoder class implements the reading of a quadrature encoder. The encoder class takes in the timer the encoder runs on, a timer specifically to be used for precise calculations of speed, the previous timer count and time read by the class, the current timer count and time read by the class, the position of the encoder, the speed of the encoder, the change of position of the encoder, and the change in time of the the encoder between reads. The Encoder class has four methods: encoder_read_curr_state(Encoder* encoder), encoder_calc_speed(Encoder* encoder,int32_t dx, int32_t

dt), zero(Encoder* encoder), and delta(TIM_HandleTypeDef* timer, uint32_t initial, uint32_t final). The encoder_read_curr_state() reads the encoder and calculates the current position and speed of the encoder. The encoder_calc_speed() function is only used inside the encoder class but can calculate the speed of the encoder based on a dx and dt value. The zero() function resets all numeric values of the encoder to zero, and sets the counting timers to zero as well. Finally, delta() calculates the change between a final and initial value of a timer, and accounts and corrects for overflow. All in all, these functions work together to create a full encoder class that allows us to read the motor encoder and have accurate speed control.

Once the encoder value has been fed into the closed loop controller, the output can be used to drive a motor, implemented with the Motor class. The Motor Class was implemented in Lab 2 and is used to drive a motor driver with a standard H-bridge setup with PWM. The class takes in the timer for the PWM channels, which set of channels to use (1 and 2 or 3 and 4), the duty cycle to send out, and an enable flag. The class has two methods: motor_set_duty_cycle(Motor* motor, int32_t doot), and motor_enable_disable(Motor* motor, uint8_t enable). motor_set_duty_cycle() sets the duty cycle of the motor based on an input doot. The doot can be any integer, and numbers greater/less than +/-100 will be saturated to +/-100. motor_enable_disable() enables or disables the motor via software via a boolean enable, where 1 is enabled and 0 is disabled.

Moving to the pitch side of things, a PitchEncoder class was created to implement the input pitch knob. The PitchEncoder class (if we were using OOP) inherits the Encoder class. It takes in a pitch from 0-11, an encoder (to act as inheritance), and a delta. The PitchEncoder class has one method: get_pitch(PitchEncoder* p_enc). get_pitch() returns the current pitch the speed should be set to, where the pitch is a value from 0-11 mapping chromatically to notes A-G#.

The final class that was implemented for this project was the display class. The Display class has an I^2C bus as an input, as well as a current note, and (temporarily) a UART pointer. The Display class has one method: display_note(Display*disp, uint8_t note). display_note() displays the currently desired note on the I^2C controlled display driver using the 16 segment display based on the given note (a number from 0-11 as before). Because we had issues with the display driver IC, the method currently just prints the current note to a serial console via UART, which is why the display driver requires a UART pointer right now.

Cooperative Multitasking

To run the peripherals all at once, a cooperative multitasking system was created. We decided to use a round-robin style system with two tasks: one that handles the motor and one that handles the display and microphone (since they are so closely intertwined). The finite state machines for both tasks can be found below in Figures 27 and 28.

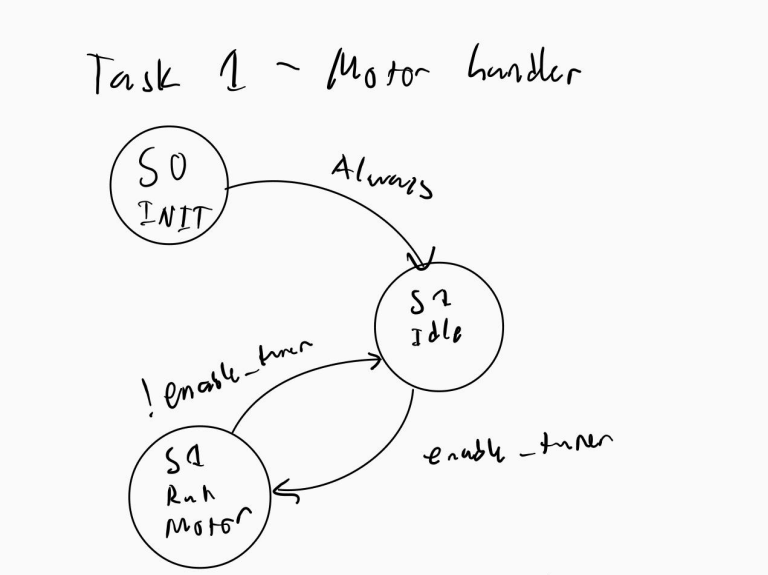


Figure 27. Finite State Machine for the motor task.

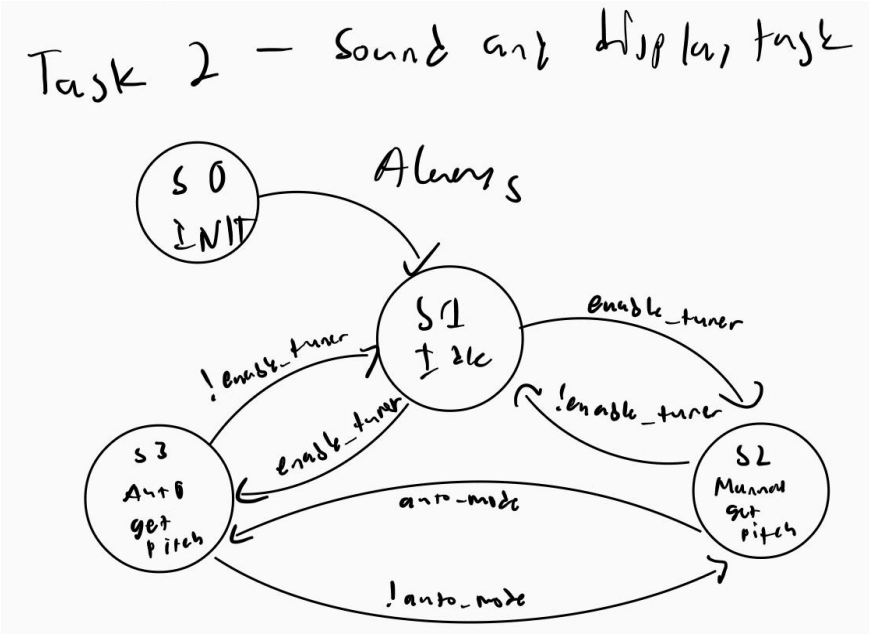


Figure 28. Finite State Machine for the sound and display.

While more complicated cooperative multitasking processes such as a priority scheduler were considered, they were deemed unnecessary due to the simplicity of the final project's tasks. Similarly, due to the accuracy required of the speed control and the microphone frequencies, a real-time operating system was considered. However, we decided that using an internal timer to calculate speeds and using interrupts to calculate the frequency of the pitch would provide enough accuracy to be able to keep the multitasking system simple.

Discussions & Conclusion

Initial Testing & Immediate PCB Issues

Once the boards were built, the first tests were to probe the boards under power and make sure everything was getting the correct voltages. Further into testing, we continued to discover issues and tried to remedy them as we went.

When probing the MCU board, we found that 3.3V was not getting out of the regulator. Upon checking the schematic against datasheets, we found that the pinout for the 6V regulator was incorrect on our board. Partway through the design, the 6V regulator was switched out for a different model using the same package. Unfortunately, the pinouts between the two ICs are different, but the symbol was not updated. To remedy this, we desoldered the L7806, resoldered it upright where it hit the correct pads with its legs, and used solder wick (the heaviest gauge stranded wire in the room at the time) to connect the tab to the ground plane. We also directly soldered the heatsink to the chip, which inadvertently put it in the best orientation for passive heat transfer – a welcome bonus.

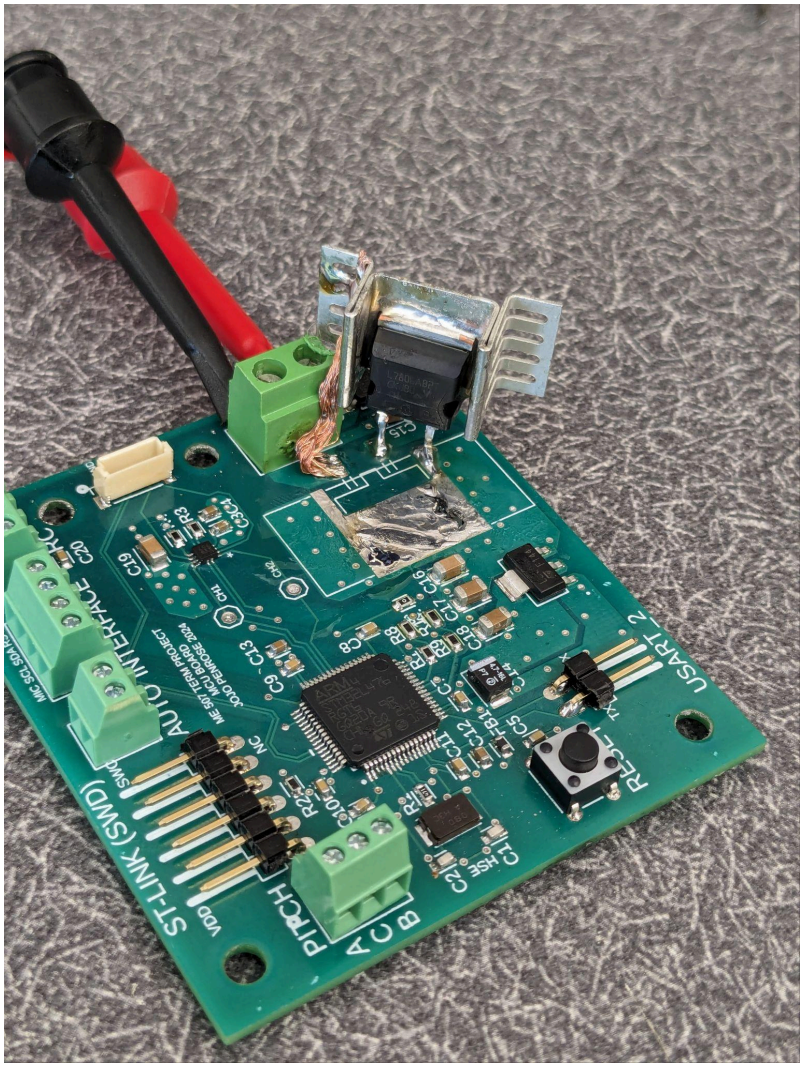


Figure 29. Relocated L7806 to the correct pinout.

While we were trying to connect the interface board LED display driver to the MCU over I2C, we discovered that the SDA and SCL screw terminals were labeled incorrectly on the interface board. We also believe that a lot of our problems with unreliable communication partly stem from our choice of 10kΩ I^2C pull-up resistors, which are far too high for the application. The addition to the RC time constant causes the I^2C rising edges to be curved, which makes for poor and unreliable communication.

When testing the audio circuitry on the interface board, we had issues with the comparator op-amp IC and the N-MOSFET not soldering flat to the board, which left certain pins floating. Once we were able to find and fix those connections, the audio circuitry worked immediately. However, we discovered that, in the absence of a bias network in the LPF switch bypass path of the 3rd-order LPF at the end of the filtering circuit, the strobe LEDs would latch on in the absence of a signal when the additional LPF switch was off. The strobes still work as intended, but leaving the LEDs on when no signal is detected was an unfortunate oversight that must be corrected in future iterations. With the additional LPF switch on, the audio circuit worked precisely as intended.

The motor driver was a major pain point for the entire project. The driver IC was chosen because it is the same IC that Pololu uses for their proprietary motor driver boards that are intended for the motor we purchased. We overlooked just how difficult the QFN package can be to solder, and we struggled with troubleshooting a non-functioning driver. We were able to probe and show that the supply pins, PWM input, and the control logic (enable pins) were at the correct levels. Then again, we could not be sure that we were probing the *actual* pins of the IC and not just solder that was sitting *around* the IC pins. We believe there was possibly an open circuit on one of the motor outputs, because we would measure both motor terminals at the same voltage when varying the motor effort over PWM. It could also have been damaged by the heat gun and iron during all of the soldering and rework, though, and we were not able to figure out why, exactly, it did not work as intended.

We made a mistake when making the symbol for the six-pin motor cable connector and got the pinout backwards. We realized later that official Pololu boards use a special reversible footprint to account for this sort of thing:

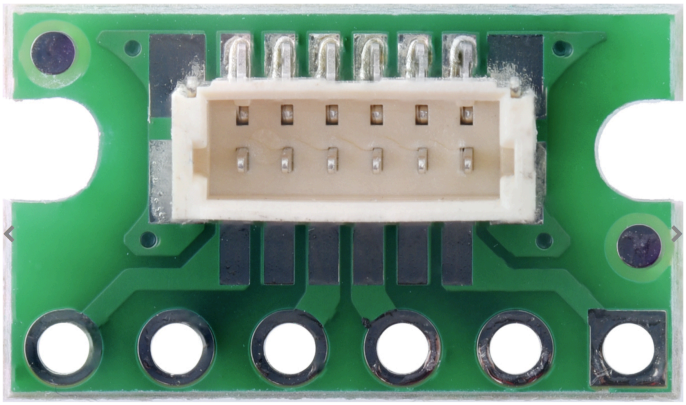


Figure 30. Pololu's six-pin cable breakout board, showing the reversible footprint.

In the absence of a reversible footprint, we were forced to flip the signal pins, which meant we no longer had access to solder pads by which the anchor tabs could be attached to. That meant that our connector was loose, and would bend by the small terminals. We used superglue to hold the connector down, which worked for a while, but during testing on the last day before the demonstration, an attempt to disconnect the motor cable caused the connector to be ripped up with the traces attached.

In an attempt to recover the project before the demonstration, we soldered fine-gauge wire directly to the STM32's pins where the motor encoder signals were read, a 3.3V power wire to the LDO tab for encoder power, and attached jumper wires to the debug pins on the PWM lines. Hot glue and a zip-tie were used to keep the fragile encoder wires in place. Those signals could be sent to and from an independent motor driver so that another spare motor could be used to demonstrate the strobe disk.

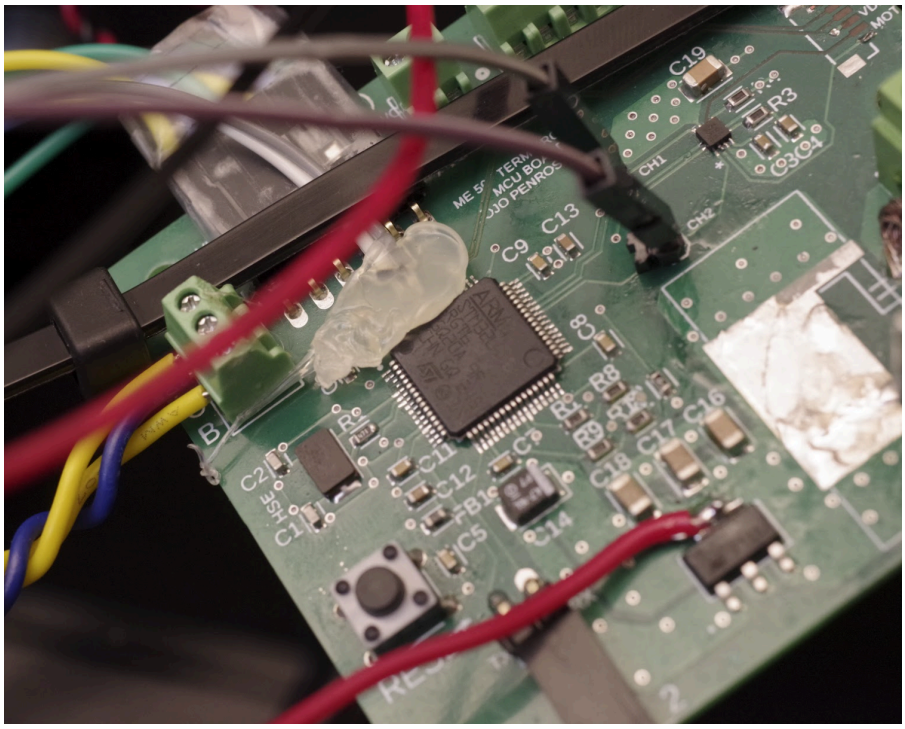


Figure 31. MCU board with motor control rescue wires visible.

Further Testing & Troubleshooting

Apart from the motor driver (in which a solution was forced via the mechanical board failure), the biggest remaining things to troubleshoot were the pitch encoder and the display driver.

The pitch encoder had two main causes for issues. Initially, pulses were not being generated, and when they were being generated, the timer counter would not function correctly. The first issue was a simple fix – adding internal pull-up resistors to the encoder signals. The encoder itself is not powered, only grounded. This was, ultimately, a communication failure between teammates. So, internal pull-ups on the encoder pins are added to pull the pulse lines high, and the encoder physically worked as intended. This was when we encountered an absolutely astonishing issue. On the particular timer we had selected, whenever the encoder overflowed or underflowed, the timer counter (not the counter in the encoder class, the actual counter register on the MCU) would stop counting. This was puzzling and concerning, as we needed the pitch knob to be able to roll over and, more importantly, to not stop counting when it did. The fix here was to cheat. We set the initial value of the encoder count register to 2^{30}, with an auto-reload of 2^{31} -1. This allows for about one billion pulses of the encoder before the encoder underflows or overflows. We don't anticipate a user turning the knob one billion times, so we don't expect the knob to overflow or underflow. Following this change, the pitch knob was able to select pitches perfectly as intended.

The other largest problem we ran into was communication with the display driver over I2C. Firstly, the display would not even turn on. When running some code we found on the internet to recursively scan for

I²C devices on the bus, we found nothing. This was strange, until we probed the pull-up resistors for the SCL and SDA lines. One of them was pulling up fine, the other was not. Investigating further with a microscope revealed a small bead of solder was lifting up one of the resistors, causing it to lose contact with the pad on the board, creating an open circuit. Fixing this, both the SCL and SDA lines worked, but this did not fix all of our issues.

The datasheet for the display driver clearly states how to turn off and on different ports of the driver via I²C. There are four registers, each driving four lights, with two bits for turning on and off the segments on the display. The important command is that sending 0b01010101 to all four registers should turn on all 16 segments (or in this case, 14 plus the two sharp/flat LED's) to full power. This is not what happened. Half of the lights turned on – specifically, always the two that corresponded to the middle two sets of two bit numbers in the register. An image of this can be seen below in Figure 30.

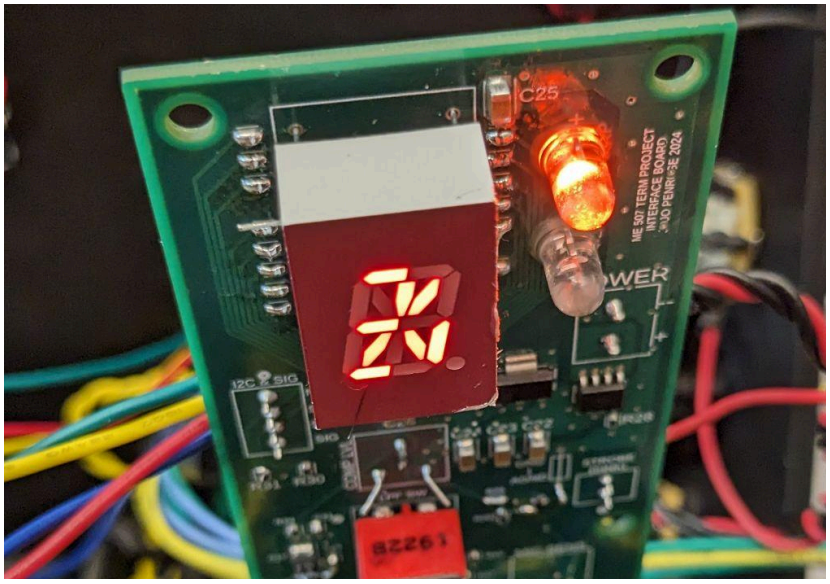


Figure 32. LED 16-segment display showing incorrect light pattern when sending command to turn all lights on.

We tried everything. We tried inverting the bits, we tried sending 0b00010101 (which actually set the correct lights), we tried lowering the I²C frequency, and we tried controlling each individual light with their individual PWM brightness registers. None of these attempted solutions worked. Electronically, everything was set up correctly as well. When we scoped the SCL/SDA lines using the lab oscilloscopes built-in I²C mode, we found that the command send by the MCU did not appear to match what we wrote in code. So, it is unclear where the source of our troubles was. Our best guess was some data corruption due to too large of pull-up resistors on the I²C lines, but this was just a guess. We were unable to get the display working, but were able to send what note should be displayed over UARTto the console for demo, so it is reasonable to assume if we were able to diagnose the issue with our I²C communication, the display would have functioned properly.

Planned Improvements: Hardware

The PCB mistakes listed previously must be remedied in future versions. We intend to choose a different motor driver that is easier to use.

In designing the strobe disk, we went for the lowest possible inertia so that the motor would not struggle. In hindsight, adding inertia would actually make for better performance because the extra inertia, acting as a flywheel, would resist changes in angular velocity. Therefore, once the disk gets up to speed, it will be more *stable* – which is positively *critical* in this application.

Screw terminals worked great in the prototype, but further versions will use gendered and crimped connectors, now that we have proven that the wiring works the way it is designed. They will be easier to assemble and work with.

The 3rd-order LPF at the end of the audio circuit needs a bias network in the bypass path of the additional LPF switch to remedy the strobe LEDs being latched on continuously.

The case still needs work. The cable management system has some improvements that need to be made, as well as shifting the MCU board up the wall a small distance so that the pitch encoder terminals are more accessible.

The strobe LEDs need to be diffused. During testing, the image of the eight LEDs behind the disk was highly noticeable. For the strobe effect to be strongest, the light should be more evenly diffused behind the disk. We are ideating on how to best do this now, and discussing solutions like using more LEDs of smaller size, or adding an intermediary obfuscating material such as fabric or paper.

Figure 33. Tuner showing the non-diffused LEDs behind the strobe disk.

The audio processing circuit was designed under the same guidelines as electric guitar effect pedals, which is something the team has extensive experience with. However, neither of us had built a microphone circuit before, and the performance based on the design was frankly a mystery to us until we were able to test it. We were relieved to see that, with the microphone gain and filter sensitivity dimed, the microphone worked well when the audio source was within a few inches of it. However, we really wanted the microphone to be able to pick up an instrument at least a few feet away with the sensitivity maxed out, and expected the user to turn it down when they were closer. We will have to either change the gain network of the microphone preamp (depending on whether or not the op-amp would be stable at that new gain, we must consult the datasheet) or add more gain stages to the circuit.

Planned Improvements: Software

Due to time constraints from both team members, a lot of the things we had hoped to complete with this project were unable to be finished by the time of the demo. The biggest thing we wanted to complete but were unable to was the automatic pitch selection using input capture. The theory behind how to implement it is simple – interrupt on rising edges on the microphone, calculate periods between rising edges, take an average based on a number of samples, and use this to calculate the frequency of the note. Of course, there is more nuance that we are glossing over, but the basic idea is there. But with the issues we encountered and the time sucked away due to other obligations, we were unable to finish the auto pitch selection. This is something we would implement very early on in the next version of the tuner however, as it is a unique feature our low-cost strobe tuner could bring to the table.

Another improvement we plan to make on the next version is the implementation of an actual e-stop. This was supposed to be a requirement for this project, and we did account for it on our board via the RC controller (we even have a driver for the RC controller ready), but we considered it to be very low priority as our project had one low-torque motor with very little inertia. But, an e-stop of some kind to stop the motor would be something we add in the future, whether through the RC controller as a debug feature or a physical switch on the tuner itself.

An additional feature we accounted for but did not implement was current sensing on the motor through ADC. Due to the motor driver not working, it was impossible to implement this during this iteration of the tuner, but a future tuner would include current sensing of the motor. Even though the motor is very low current, if there is a reason the motor is drawing higher current, we want to be able to shut off the motor so as to not burn it out – it is a very small motor.

Finally, the biggest software improvement we would make is making the I^2C display work. As mentioned previously, the code to determine what pitch to use works, we are just transmitting the correct pitch over UART currently. Determining the cause of the I^2C issues and making the display work would be the final step in software of the project, and we are already most of the way there.

Project Takeaways

Although we had to go to great lengths and improvise last-minute solutions for motor control, we were astonished to see that the project actually works as intended.

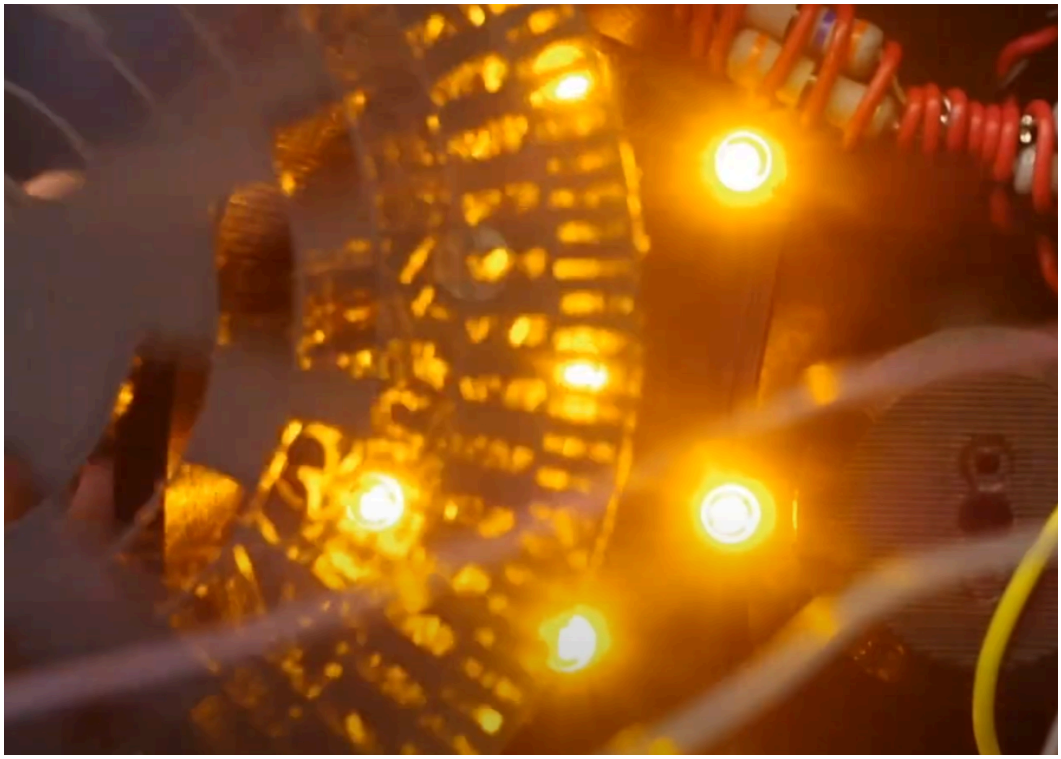


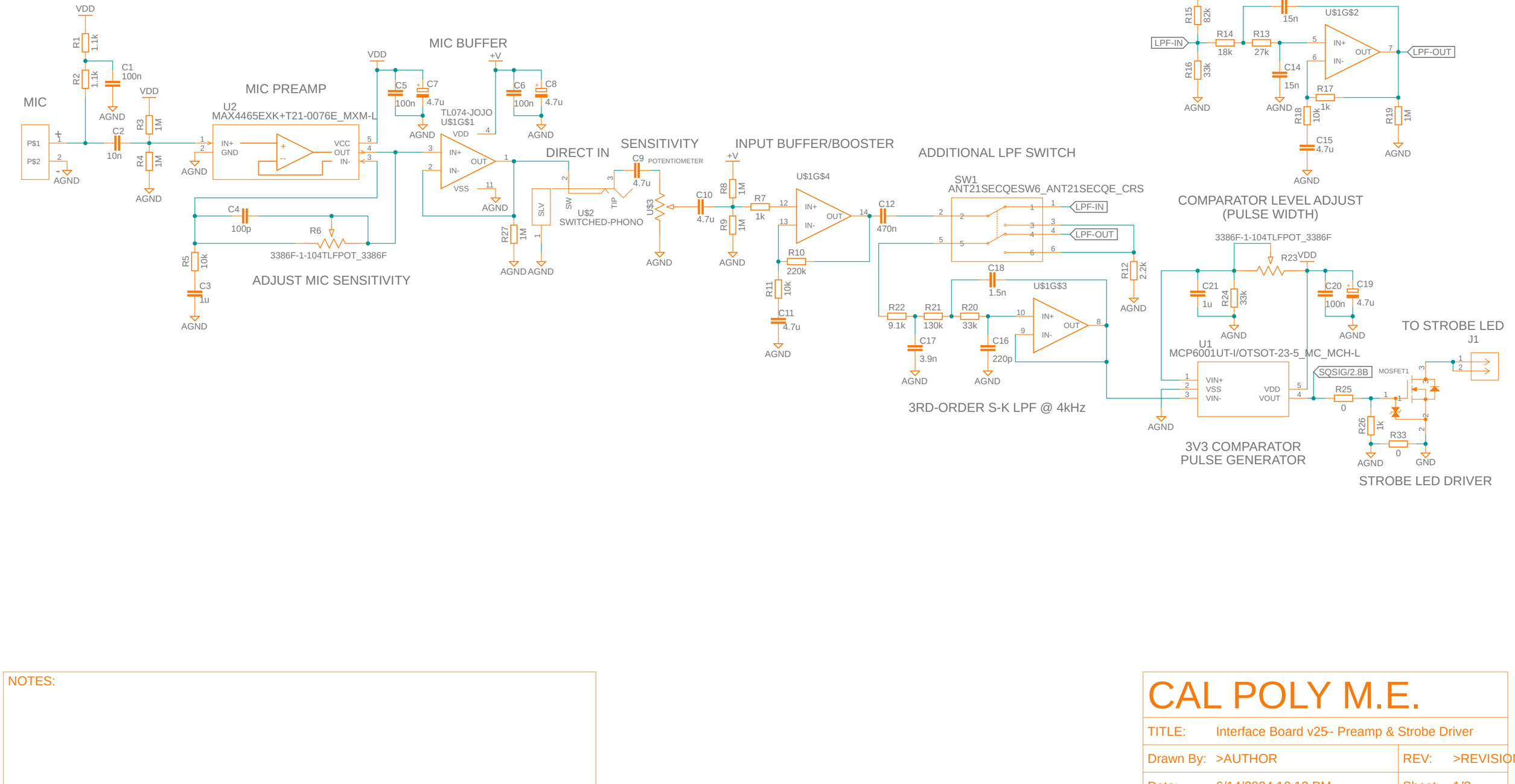
Figure 34. A freeze-frame from a demonstration video showing the strobe slots clearly visible.

Using a little bit of math to convert the speed calculations from our old motor to the gear ratio and encoder counts of our temporary demonstration motor, we were quickly able to set the disk speed to the note A and sing the note into the microphone. We saw the image of the frozen slots appear on the disk, and it slowly rotated up and down depending on whether we sang sharp or flat. As a proof of concept and very first prototype, we would say that the project was an absolute success. Aside from issues with the interface, *functionally*, the tuner does *exactly* what we wanted it to do. With more tuning, a more robust controller, and a redesigned strobe disk and light, we strongly believe we can compete with the current retail offerings on the market for a low-cost, high-precision instrument tuner.

Appendix

Appendix A - Printed Circuit Board Schematic Drawings

Appended to the following pages are the schematic drawings for the two printed circuit boards used in our design. They were created in Fusion 360 under an Educational license. The complete Gerber file archive will be submitted alongside this report.



NOTES:

CAL POLY M.E.

TITLE:	Interface Board v25- Preamp & Strobe Driver		
Drawn By:	>AUTHOR	REV:	>REVISION
Date:	6/14/2024 10:12 PM	Sheet:	1/2

16-SEGMENT DRIVER

U3
TLC59116IPWRPW28-L

SHARP/FLAT LEDS

+V
FLAT TH-LED
SHARP TH-LED
R49 330, 1/5W
R48 330, 1/5W
R29 1k
GND

REXT, A0, A1, A2, A3, OUT0, OUT1, OUT2, OUT3, GND_2, OUT4, OUT5, OUT6, OUT7

VCC, SDA, SCL, RESET, GND, OUT15, OUT14, OUT13, OUT12, GND_3, OUT11, OUT10, OUT9, OUT8

C26 10u
R30 10k
R31 10k
VDD
SDA
SQSIG/1.9D

MCU BOARD CONNECTOR
J4

R35, R37, R32, R34, R36, R33, R40, R41, R42, R43, R44, R45, R46, R47 — 330, 1/5W x 14

LTP-587HRLTP-587HR_LTO
LED1
DP U T S R P N M K H G F E D C B A

C25 10u

16-SEGMENT LETTER DISPLAY

3.3V REGULATOR

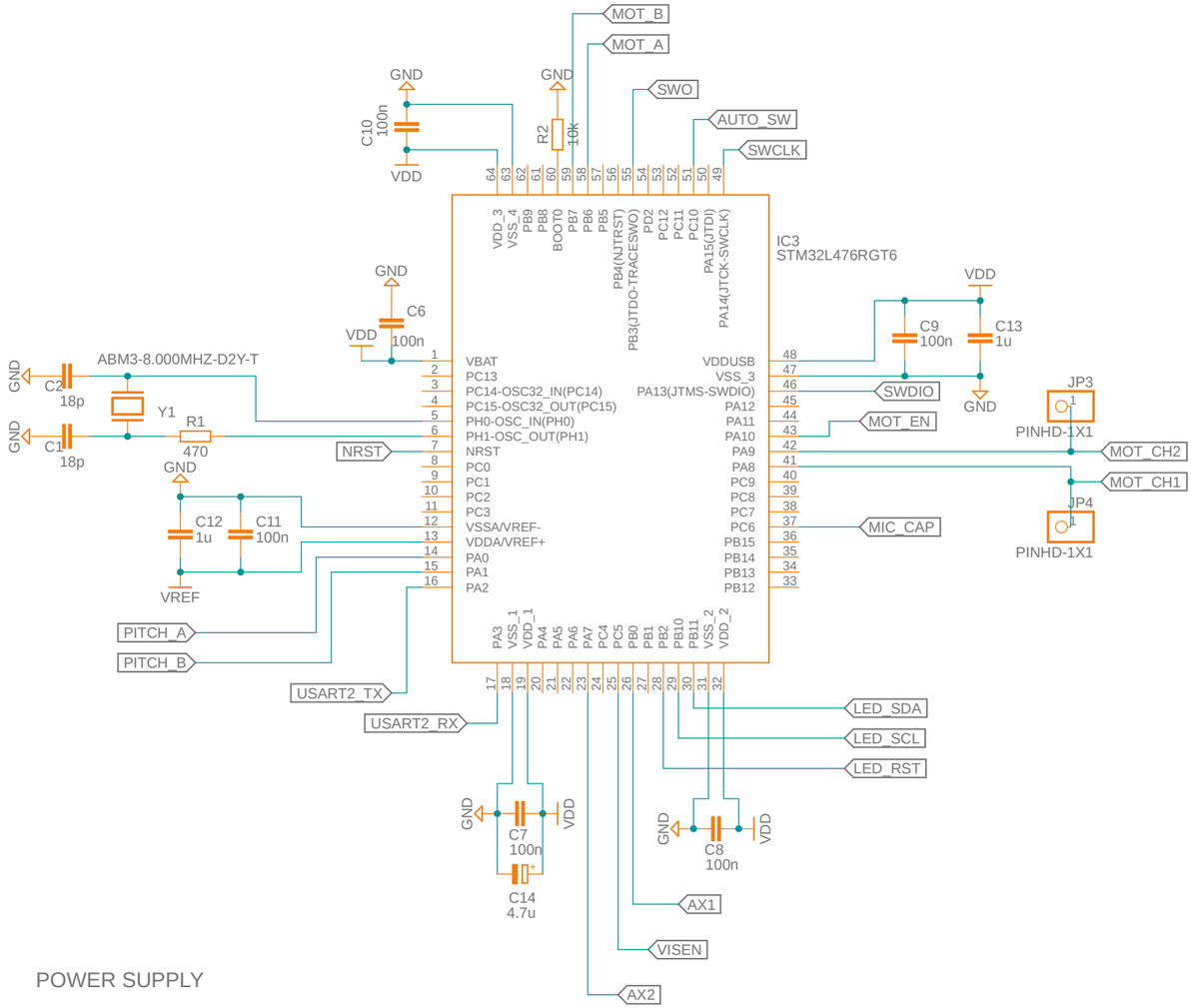
J2
Q1 FDS4435BZ
CR2 LD1117S33CTRSOT-223_STM-L
VIN
VOUT ADJ/GND
R28 10k
C22 10u
C23 4.7u
C24 4.7u

NOTES:

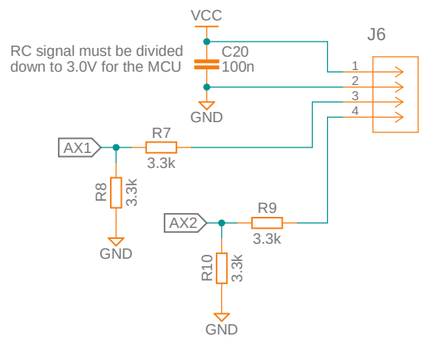
CAL POLY M.E.

TITLE:	Interface Board v25- Power Supply & I/O		
Drawn By:	>AUTHOR	REV:	>REVISION
Date:	6/14/2024 10:12 PM	Sheet:	2/2

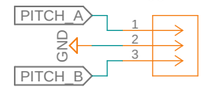
MCU



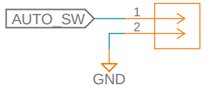
RC RECEIVER



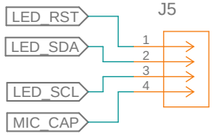
J3 PITCH ENCODER



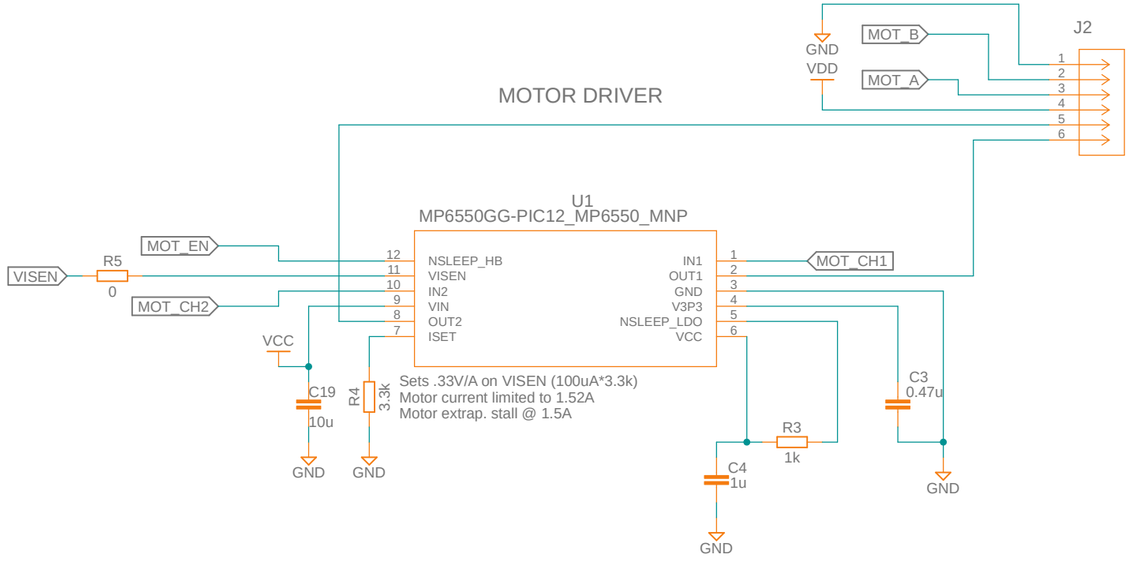
J4 AUTO-PITCH SWITCH



J5 INTERFACE BOARD CONNECT

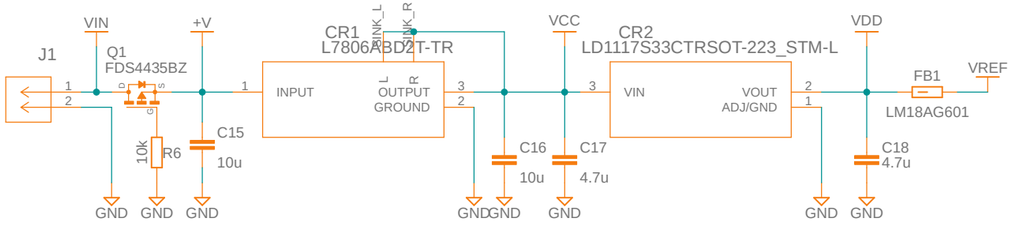


MOTOR DRIVER



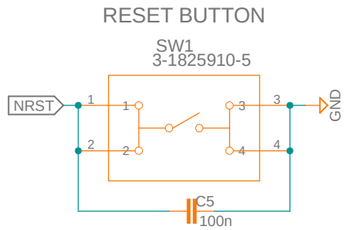
POWER SUPPLY

6V LDO

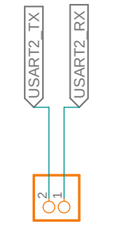


3.3V LDO

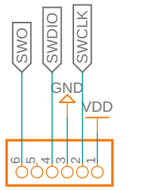
RESET BUTTON



ST-LINK UART (PuTTY)



ST-LINK



NOTES:

CAL POLY M.E.

TITLE:	MCU Board v24		
Drawn By:	Jojo Penrose	REV:	HW 3
Date:	6/14/2024 10:14 PM	Sheet:	1/1