ECE 445: SensorFX

By

Joseph Schanne

Karan Sharma

Paul Matthews

Final Report for ECE 445, Senior Design, Spring, 2025

TA: Eric Tang

7th of May, 2025

Project No. 82

Abstract

Musicians working with Digital Audio Workstations (DAWs) will often use MIDI (Musical Instrument Digital Interface) controllers to apply and modulate audio effects applied to their prerecorded or live music. One problem, though, is that these controllers, taking the shapes of knobs or buttons, are often clunky and hard to integrate into a musical act, especially a live performance. Our device allows musicians to control audio effects by contracting and releasing any muscle on their body.

By hooking up an EMG (electromyogram) sensor to a transmitter device, and sending a signal to an ESP32 emulating a compliant USB MIDI device, we can map the electrical readings of their muscles to a device simulating a knob, allowing a user clenching one of their fists, for example, to control the intensity effects (such as echo) applied to digital audio signals. Since we chose to use the industry standard MIDI protocol to communicate with a DAW, instead of directly processing audio signals, these effects can be arbitrarily chosen by the user for whatever suits their needs. A singer could control pitch by closing and opening their arm. A guitarist could kick to trigger a cymbal sound. The possibilities are endless.

Through this report, we prove that our device is cost-effective, capable of operating in real world situations, and could easily be used for live performances.

Contents

1. Introduction

1.1 Background

In the early days of digital music production, various manufacturers ran into problems where their instruments, their synthesizers and other hardware, and their software was often incompatible with other vendor's products or even earlier versions of their own products. By the late 80s/early 90s, the MIDI standard was decided on by the vast majority of the music industry for digital music devices to communicate with each other, and remains the standard to this day. One advantage of this is that any device or controller made to work with the MIDI standard can interoperate with all other devices using this same standard. [1]

Musicians working with Digital Audio will often use software called a Digital Audio Workstation (DAW) to record, playback, and edit their music. In most DAWs, you can also apply effects on the fly, taking in an audio signal input, and transforming it to a different output with effects applied in real time. Nearly all Digital Audio Workstations will be built to interact with MIDI devices, so any MIDI controller such as a knob or a button would work in all DAWs. The effect of the button or knob being interacted with is configured by the user within the DAW itself, so an action applied to a controller could trigger or modulate any arbitrary effect. [2]

1.2 Problem

While there are countless MIDI controllers on the market, there is a high barrier of entry in the price for new musicians. Additionally, in practice, it is often clunky or inconvenient to have to twist a knob while performing music; if recording music for later use, then applied effects can be changed in production, but any fluidity in the performance itself is lost. Where the problem of the current interface for MIDI controllers is most apparent is with live performances. Newer musicians often don't have access to a sound engineer for their performances, so it's impractical to expect that they could change effects while performing in any meaningful way besides just turning them on and off, especially if they are playing a two handed instrument like the electric guitar, or a digital piano. How can musicians affordably apply and change digital effects to their music *in real time* while freeing up their hands to focus solely on making music?

1.3 Solution

The solution we propose is SensorFX, a revolutionary new device that allows you to control digital audio effects by flexing your muscles. Our design makes uses of Electromyograph (EMG) sensors, which map contractions in a user's muscle readings to voltage readings. We transmit these muscle readings to a receiver module that the user simply plugs into their computer and the receiver converts these voltage readings into a MIDI signal for the DAW to interpret.

The EMG electrodes strap directly to the user's body, and are connected to the transmitter module, which normalizes these signals and sends them to the receiver module. The transmitter module is

housed in a wrist-band like device that one can strap on anywhere on the body. This allows the user to have full mobility away from the DAW, and to focus entirely on the production of music.

The full pipeline of the signal is as follows. First, the EMG sensors receive a voltage level from the electrodes. This signal is then sent through various Op Amps to get the signal in a readable range for the onboard ADC on the ESP32. The ESP 32 within the transmitter diode receives this signal and converts it to a range within 0 to 127 inclusive. Using the ESP-NOW protocol, this signal is then sent wirelessly to the receiver module. Finally, the receiver module converts this integer between 0 and 127 inclusive to a MIDI control change signal, which is interpreted by the DAW as a value on a MIDI knob. A visual representation of this process is shown in Figure 15.

2 Design

2.1 Transmitter Design

The initial implementation of the muscle sensor relied on a breadboard setup based on OpenEMG, an open-source budget-friendly EMG device. The schematic for the OpenEMG device is shown in Figure 16. Components were sourced and the sensor was built without initially planning for microcontroller integration, using Amazon EMG pads with connectors salvaged from a knockoff TENS unit. The breadboard demo functioned successfully with an Arduino Uno (since ESP32-C3 Devkits weren't yet available), featuring analog input on pin A0, three blue LEDs for visual feedback, and a mode-selection switch to support both calibration and display functionality.

The system evolved through several software iterations, progressing from fixed thresholds to dynamic calculation that automatically divided the calibrated voltage range into equal segments. Version 3.0 introduced PWM-based LED control for progressive illumination, while Version 4.0 added easily-configurable voltage settings for fine-grained control over LED brightness. Testing verified correct LED activation based on calibrated thresholds, with detailed output visible on the serial monitor for verification during development.

Following the software development, the project migrated from Arduino to ESP32-C3 requiring significant adjustments for the different architecture. The ESP32-C3's 12-bit ADC provided finer resolution compared to Arduino's 10-bit ADC, while the PWM implementation needed complete refactoring to accommodate ESP32's channel-based architecture. A key enhancement in Version 2 was implementing a sequential fill-up pattern for LEDs that mimics filling a container, creating a more intuitive visual interface that progressively illuminated from bottom to top as input values increased.

The PCB design was the next major step, centered around the ESP32-C3-WROOM-02 module with careful attention to analog signal conditioning. The circuit incorporated a differential amplifier stage with the LM324, high-pass and low-pass filters, and a tunable amplifier stage for final signal adjustment. Power management used the LM1117DT-3.3 voltage regulator, and the CH340C provided the USB-to-UART bridge for programming. The design faced several challenges including improper thermal relief and signal routing issues, which were addressed in subsequent revisions. Testing showed the system successfully captured EMG signals with sensitivity to muscle contractions, though further optimization could improve interference handling and overall signal quality.

2.1.1 Transmitter PCB

The transmitter PCB design represents the physical implementation of the EMG circuit schematic, utilizing a two-panel PCB layout for efficient manufacturing and assembly. The dual-panel approach allowed for optimal space utilization while keeping manufacturing costs reasonable.

The layout carefully separates the analog and digital portions of the circuit to minimize interference, besides a single trace on the back side of the PCB for the EMG electrode passing below the ESP32. The analog section containing the LM324 quad op-amp and associated filtering components occupies the left side of the PCB, with special attention paid to trace routing for the differential amplifier inputs, which are placed at alternate edges of the PCB. The high-pass and low-pass filter sections are positioned to maintain signal integrity, with components arranged to minimize parasitic effects and optimize the frequency response. Ground planes are strategically placed to provide proper shielding between analog and digital sections.

The ESP32-C3-WROOM-02 module is positioned with adequate clearance for its integrated antenna, on the right side of the board, with the supporting components including the CH340C USB-to-UART bridge and programming circuitry arranged for optimal signal routing from the USB port. Power distribution is handled through dedicated traces, with the LM1117DT-3.3 voltage regulator and decoupling capacitors placed to ensure stable power delivery throughout the circuit. The ICL7660N/TR charge pump for negative voltage generation is isolated from sensitive analog components to prevent switching noise from affecting measurements.

User interface elements including the tactile switches and status LEDs are arranged along the accessible edges of the PCB for convenient operation. The USB connector and battery connection points are positioned for easy external connections, with mounting holes provided for secure attachment in the final application. The two-panel design not only improved manufacturing efficiency but also allowed for design variations or revisions on each panel without requiring a completely new production run.

2.2 Transmitter/Receiver Communication

Once we have received the normalized voltage signal, the next thing we need to do is transmit the Data to the receiver module. Since we were already using the ESP32 CPUs, there were two options that we could use for communication: ESP-NOW or Bluetooth. Looking at both methods of communication, there were advantages and disadvantages for each. See figure 2 for a chart comparing the two.

Bluetooth is an industry standard, and very common, so developing products utilizing this technology would be a relatively smooth process. However, it offers high latency of up to 300 ms, and it could not guarantee a high range on the ESP32 platform.

ESP-NOW is a proprietary standard developed by expressif for communication between two ESP devices. We would be locked in to using expressif products in the future if we wished to develop subsequent revisions that were compatible with the first iteration of the product. Another issue is that since it's relatively less common, we would have to do some independent research to figure out how to integrate it with our product. However, it offered an impressive 5ms of latency, and made claims of supporting up to 200 ft of range.

We settled on using ESP-NOW, since having low latency and high range are fundamental for this device to be useful for live performers.

Having a low latency is important because at a standard tempo for a song, around 120 beats per minute, common note lengths can be very short. An 1/8th note could last for 250 ms, and a 1/16th note could last for 125ms. If our bluetooth implementation could be as slow as 300ms, then an effect could be completely off by 3 notes, creating an unpleasant experience for the performer. By using ESP-NOW which offers a latency of 5ms, we can greatly improve the perceived delay. Since the human reaction time can be as low as 13ms, by using ESP-NOW, the effect could appear instantaneous to the user.

A live performer would also need a high range. 30 feet might be enough to cover a whole stage, but when you include the fact that there might be interference from all the other electrical devices on stage, you would want a healthy buffer to ensure a robust solution when outside of ideal environments.

2.2.1 Transmission Side

On power on or on signal loss, each transmitter initializes ESP-NOW on the board and continually scans for a receiver board until it finds one. Once a connection has been established with the receiver board, it then reads the digital values from the onboard ADC, which shows the voltage levels of the EMG sensors, normalized and made digital. Once it receives this number, it converts it to a number between 0 and 127. The antenna is built into the ESP32C3, so no separate one is needed.

2.2.2 Data sent

Every 10ms, Transmitter sends a 32 byte status message (plain text ASCII) along with a 1-byte value 0 to 127 showing the strength of the EMG signal.

The data sent to the receiver board is a 33 byte structure. There are 32 bytes to indicate a status message, useful for debugging or any other needed communication between the sender and the receiver, and a one byte for the intensity of the EMG voltage reading. This message is deconstructed and sent as 33 raw bytes over to the receiver, which then loads the 33 bytes back into memory and casts it back to a congruent structure. We use the ESP-NOW protocol to encrypt the messages between the transmitter and receiver using the AES-128 algorithm to ensure there is no unwanted interference between the receiver and other devices.

2.2.3 Receiver Side

On boot or when a signal is lost, the receiver is set to wait for any attempts to establish a connection from a transmitter.

Rather than acting on a fixed schedule, the receiver is set to asynchronously load the received message into its memory whenever a message is received. The code to do this is in a callback function that operates whenever the receiver receives a message. This allows the receiver to run code related to MIDI communication with the DAW completely separately from the receiver communicating with the transmitter.

The receiver is set to store the last received message in memory, that way if there is a temporary drop in the connection, the last known value will be held, rather than simply cutting to zero or having the program stop functioning entirely. To see how these messages are implemented in C++, see figure 4.

The receiver uses a Shenzhen MyAntenna M01-0600890R0A (figure 3). We chose this antenna because it came bundled with the ESP32S3 that we were using on the receiving side, and in testing it gave good results. For example, while testing in the lobby of the Electrical and Computer Engineering building in the University of Illinois, Urbana-Champaign, we were able to communicate with someone on the ground floor and someone on the balcony of the second floor. This was inside the building, which of course was full of plenty of devices and materials that would cause electrical interference with the transmission. Since it was able to operate flawlessly in such a noisy environment, we knew that we could achieve good range in live performances, even in non-ideal locations.

Overall, the design of the Transmitter/Receiver Implementation was built with robustness in mind, tolerating drops in connections by holding the last known value, constantly attempting to reconnect if the connection cuts out, and degrading gracefully in a way musicians would appreciate.

2.3 MIDI / USB Compliance

This section describes the firmware and hardware needed to make our device compliant to USB and MIDI standards, as well as a brief overview of the MIDI protocol, DAWs, and other musical / production terminology.

2.3.1 MIDI and DAW Overview

Musical Instrument Device Interface, or MIDI, has remained the standard communication protocol for all musical instruments and devices for decades. The original MIDI 1.0 protocol created in 1983 defines a real-time serial byte stream travelling at 31250 baud on a 5-pin DIN cable. Each cable supports 16 logical channels, numbered channels 1-16, allowing simultaneous sending and receiving. Messages are composed of one start bit, eight data bits, and one stop bit. Every status byte identifies both the message type and the channel, with the most common message types being NoteOn / NoteOff messages to handle the use of actual instrument-like devices and ControlChange (CC) messages to handle various effects and their intensities. For example, a full standard CC message consists of three bytes: 1. the aforementioned status byte to identify the message type, 2. the controller number to specify which ‘control’ is being adjusted (common assignments are 1 for ModWheel, 7 for Volume, 10 for Pan, and more), and 3. the controller value to convey the new setting for the given control. [10]

Modern devices make use of the MIDI protocol with USB communication, commonly referred to as USB-MIDI or MIDI over USB, by wrapping standard MIDI messages inside USB packets and handling communication through the USB protocol instead. Each full therefore USB packet contains one or more 4-byte “event packets”, with the first packet being a Code Index Number (CIN) to identify the length of the embedded MIDI message (the CIN for the given CC message above would simply be 3); the remaining three bytes make up the original MIDI message itself. [11]

A Digital Audio Workstation (DAW) is software that combines a multitrack tape machine, mixing console, outboard processors, and MIDI sequencer into one environment on any computer. Within that workspace, a track is a linear lane on the timeline that holds media items (audio files, MIDI regions, or video), plus its own signal path—input, processing chain, and output bus. Effects are loaded as plug-ins of various file types (VST, AU, AAX, and others) in the track’s insert chain. Each plug-in processes audio or

the MIDI stream in turn, with an order similar to pedals on a real pedalboard; this information will be pertinent to the following explanations. [2]

Many DAWs on the market today are quite costly, with the most expensive ones costing around $200 or more. Since our project aims to be a more affordable option for new musicians, the chosen DAW for this project is REAPER, which offers an unlimited trial period for the full functionality of the software. Additionally, it has an extremely small footprint of under 20 MB, and the software is known to run well on older hardware, making it ideal for musicians where equipment and computer quality could be a problem. [9]

2.3.2 MIDI over USB with TinyUSB

Once the user's muscle voltage readings have been obtained and a normalized value has been calculated for the desired intensity of the given effect, the ESP32C3 transmitter chip continuously sends this value to the receiving chip. An important aspect of our chosen receiver chip, the Seeed Studio XIAO ESP32S3, is the inclusion of a native USB-OTG peripheral that can present itself to a computer as a class-compliant MIDI device with nothing more than the board's USB-C connector and the use of TinyUSB, an open-source cross-platform USB stack commonly used for the ESP device family. Upon plugging in to the computer, the S3 is put into device mode and begins a standard enumeration handshake. Use of functions from the TinyUSB library, which is built in natively on-device, supplies the required USB descriptors: an Audio-Control (AC) interface that announces an audio device class and a MIDI-Streaming (MS) interface that allows one IN and one OUT bulk endpoint for MIDI packets. In our code, we simply include the TinyUSB package and make sure to enable a MIDI class with the use of a class identifier 'USBMIDI.' We then set USB-OTG mode in Arduino and call functions USB.begin() and MIDI.begin() in the setup portion of our code; these lines allow TinyUSB to entirely handle any complicated MIDI handshakes that need to occur with simple function calls such as MIDI.controlChange() or MIDI.noteOn(). [8]

Once the MIDI configuration portion of the code runs, the PC (and more importantly, the DAW) sees nothing different than a standard MIDI controller named 'TinyUSB MIDI,' as seen in Figure 5.

2.3.3 DAWs and MIDI Learn

Once the device is properly configured and the PC sees it as a MIDI device, it is then up to the user to set up the DAW and their desired effects properly. Once the user has set up their track and instrument input properly, they must then enable the device to be an effect controller, not just a MIDI input device, as shown in figure 6. After properly registering the device as an effect controller, everything is set for the user to make use of the MIDI Learn function.

Native in some capacity to most modern DAWs, MIDI Learn is a quick and easy way for musicians to connect digital effects to physical knobs or buttons on a MIDI controller. Upon opening a desired effect, some kind of visual interface is brought up (as shown in Figure 7), usually made to look similar to the existing pedal that the effect emulates. Once in this interface, the user makes use of MIDI Learn by simply clicking on the desired parameter in the effect window, bringing up the parameter settings window, and clicking "MIDI Learn." Upon clicking this, the DAW will automatically search for any

incoming CC messages on the specified channel as shown in figure 8; in a normal use case, the user would then move the knob or button that they want the desired effect to be controlled by, and the DAW automatically connects any incoming CC messages from that knob to the desired effect in real time. For our case, the receiver chip is sending CC messages to the DAW continuously, so once the MIDI Learn function is clicked the DAW immediately sees the incoming messages and automatically assigns the effect to our device. Having now entirely completed the set-up needed, the user can record or play audio from their instrument freely and change the effect in real time. [9]

3. Design Verification

The most important part of this design was, without a doubt, the sensor array. The precise design of this system has been thoroughly discussed in prior documents, but in short the system consists of an LM324 quad opamp along with analog filtering circuitry. The first filter is a differential opamp, followed by a second order high pass filter and a second order low pass filter. The schematic for this design is shown in the design section, along with the full list of parts and a picture of the circuit built on a breadboard(in the images section/parts section). This breadboard based circuit worked quite well almost as soon as it was completed, and minimal slowdown was encountered at this stage. The responses that were received from testing a variable frequency sin wave passed through the circuit showed the exact behavior we wanted, with above 500Hz waves being filtered out as well as waveforms below 22Hz. Additionally, a spectrum analyzer could have been used at this stage, but we didn't have easy access/requisite expertise to accomplish this in a reasonable timeframe. It was decided that manual testing revealed enough functionality for us to proceed. When tested, a sine wave of 5uV produced an output voltage of around 200-500mV, which was conducive to good functionality with real electrodes. When tested with electrodes, the system worked very well. It produced very high voltages, sometimes above 5V when the diode was removed from the circuit. Later down the line, it was decided through testing (shown below in Figures 9 and 10)that 3.3V would be a better voltage to run the sensor at, because it limited the output of the circuit to the ESP32's max ADC voltage of 3.3V and had comparable performance.

3.1 PCB V1.0 testing

This stage constitutes the initial PCB design and testing, which consumed the bulk of the design time for the hardware implementation of the transmitter board. The first iteration of the PCB (fig x) was designed before the decision was made to switch from 5V power to 3.3V power for the sensor array, but this design did not make it into production. The Second iteration of the PCB design took the power change into account, and used a USB-C port wired into an MCP73831-2-OT BMS chip. In retrospect, this was a bad design decision, as it added unnecessary complexity to the system that could have been avoided by simply using an external battery charger. The main issue with the BMS system was that it was wired in such a way that allowed the usb voltage to be applied in series with the battery voltage. This destroyed the first hand soldered version of the PCB. It is believed that this issue was due to improper thermal relief, or a short somewhere else in the system. Either way, this was circumvented by removing

the battery charger from the board, and wiring the power system directly into USB, keeping in mind not to plug in both a USB and a battery at the same time.

A USB C port was also added at this stage, as well as an LM1117-DT-3.3 for voltage regulation. The ESP32-C3-WROOM-02 was chosen because it was an easy package to hand solder, and because it has a built-in antenna. Three buttons were added to this design, one for boot, one for reset, and one for toggling between modes, which was wired to GPIO2. The USB bridge to the ESP32 was established by a CH340C, which was wired to USB D+ and D-. When the PCB arrived, it was soldered properly and tested by simply plugging into USB-C. The device registered as a CH340-C in windows, but was not recognized as an ESP32 family device. This was because D+ and D- from USB must be connected to both the programmer chip, as well as pins 13 and 14 (D- and D+) on the ESP32. In order to remedy this issue, stranded thin gage wire was used manually bridge the traces D+ and D- from the CH340C to the D+ and D- pins of the ESP32. This worked to get the device to register, but there was yet another issue: the ESP32 was not properly entering boot mode for programming upon attempting to upload code from the arduino IDE. Initially, it was believed that this was due to the absence of proper programming circuitry, so the proper circuitry was wired on a breadboard and hand soldered to the ESP32's DTR and RTS pins (as shown in Figure 12). This, however, was not the true issue. We were able to successfully manually put the board into boot mode manually by bridging the connections on the button traces manually to overcome the incorrectly wired button. Once the ESP32 was in boot mode, as indicated by the LEDs, the ESP32 flashed successfully and showed proper output through the serial monitor indicating that it was able to both receive the input from the sensor array, as well as transmit that value to the receiver (Figure 12).

Another issue with the first version of the PCB was the USB port, as it was positioned too far into the board to connect to a standard USB-C port. To remedy this issue, it was first attempted to manually lift the port further out from the board (Figure 14), when this was unsuccessful, it was decided to instead cut open a USB C cable to make it able to clear and plug in to the port. This was successful, surprisingly.

The last issue discovered in testing was improper/non functional sensor arrays. For some reason, the LM324 SMD package was responding differently than the THT variant built earlier in the prototyping stage. Sometimes this circuit would output negative voltages at idle, like a -2V pulse that corresponds to the charging and discharging of the final smoothing filter on sensor array, followed by a discharge when the voltage exceeded the Schottky diode's activation voltage. This was a very odd issue, and it was not able to be solved in testing. It would be prudent to add more spacing and better routing to the sensor array, as this was likely the culprit of our issues.

3.2 PCB design revision 2.0

The second design revision was quite imperfect, but made various improvements over the first design. Firstly, it separated the USB and battery lines physically with a switch, which physically decided whether to use USB or battery power, allowing the device to be programmed without removal of the battery. It was also decided that removing the BMS entirely was the best course of action given the short time window for this redesign, given the thermal and power issues it caused in the V1 board. Traces were added to fix the USB D+ and D- lines, allowing proper registration and USB communication. Programming

logic for DTR and RTS signals (from the Espressif documentation [13]) was fixed with the addition of two NPN transistors. The transistors sourced were as close to the spec of the ESP32-C3-WROOM-02 devkit's spec as possible to ensure compatibility.[13] Buttons were also moved to the bottom edge of the PCB for accessibility, but rather foolishly, the button issue still persisted here as it had not yet been discovered on the V1 PCB! The full schematic for this redesign can be found in the images section.

4. Costs

4.1 Parts

Table 4.1.1 Costs Price Per Unit (cost of components in final unit)

Part	Manufacturer	Retail Cost ($)	Units	Actual Cost ($)
PCB Case	CU Community Fab	2.15	1	2.15
1276-6840-1-ND (0.1uF Cap)	SAMSUNG	0.017	4	0.068
1276-6480-1-ND (0.47uF Cap)	SAMSUNG	0.037	2	0.074
1276-1123-1-ND (0.33uF Cap)	SAMSUNG	0.039	2	0.078
PCE3948CT-ND (10uF Cap)	PANASONIC	0.189	3	0.567
1276-2970-1-ND (4.7uF Cap)	SAMSUNG	0.19	1	0.19
160-1423-1-ND (Green LED)	LITEON	0.10	4	0.4
SS14CT-ND (Schottky Diode)	ONSEMI	0.29	1	0.29
900-2171790001CT-ND (USB-C Conn)	MOLEX	0.83	1	0.83
455-1719-ND (2-pin Header)	JST	0.10	1	0.1
CR0805-JW-331ELFCT-ND (330 Ω)	BOURNS	0.023	3	0.069
CR0805-FX-1002ELFCT-ND (10 kΩ)	BOURNS	0.026	2	0.052
CR0805-FX-1001ELFCT-ND (1 kΩ)	BOURNS	0.026	5	0.13
CR0805-FX-1802ELFCT-ND (18 kΩ)	BOURNS	0.035	3	0.105
CR0805-FX-4702ELFCT-ND (47 kΩ)	BOURNS	0.026	3	0.078
CR0603-FX-2001ELFCT-ND (2 kΩ)	BOURNS	0.10	1	0.1
CR0603-FX-5101ELFCT-ND (5.1 kΩ)	BOURNS	0.02	2	0.04
CR0805-FX-1502ELFCT-ND (15 kΩ)	BOURNS	0.026	2	0.052
TC33X-104ECT-ND (100 kΩ Trimmer)	BOURNS	0.25	1	0.25
P13349SCT-ND (Tactile Switch)	PANASONIC	0.723	3	2.169
1965-ESP32-C3-WROOM-02-H4 (MCU)	ESPRESSIF	3.51	1	3.51
ICL7660CSA+-ND (Charge Pump)	MAXIM	4.30	1	4.3
296-14597-1-ND (LM324 Op-Amp)	TI	0.20	1	0.2
MCP73831T-2ACI/OTCT-ND (Li-ion Charger)	MICROCHIP	0.76	1	0.76
TOTAL				**16.562**

4.2 Labor

Table 4.2 Labor Costs

Job	Hours Taken	Done By Whom	Employment Cost ($/hr)	Labor Cost ($)
PCB Case Design	8	Karan	75.00	600.00
PCB Design	48	Joseph	75.00	3600
MIDI Software	20	Paul	75.00	1500
Breadboard Prototyping	24	Joseph	75.00	1800
Research	60	All	75.00	4500.00
ESP-NOW Software	10	Karan	75.00	750.00
EMG Software	5	Joseph	75.00	375
Total				**131251**

4.3 Other Costs (incurred while Prototyping)

Table 4.3 Costs incurred while Prototyping

Reason	Expenditure ($)	Incurred By Whom	Actual Cost ($)
PCB Case Prototypes	2.00	Karan	2.00
CKN10397-ND Slide Switch (unused)	4.7	Joe	4.7
MMSS8050 NPN Transistors (unused)	1.25	Joe	1.25
1276-6455-1-ND 10 µF 0805 Caps (extra)	0.36	Joe	0.36
1276-1123-1-ND 0.33 µF Caps (extra)	1.09	Joe	1.09
1276-6480-1-ND 0.47 µF Caps (extra)	0.3	Joe	0.3
1276-6840-1-ND 0.1 µF Caps (extra)	0.53	Joe	0.53
Green LEDs (extra)	1.6	Joe	1.6
Blue LEDs (extra)	2.37	Joe	2.37
Panasonic tact switches (extra)	6.51	Joe	6.51
ESP32-C3 modules (extra)	14.04	Joe	14.04
ICL7660 charge-pumps (extra)	8.6	Joe	8.6
MCP73831 chargers (extra)	3.04	Joe	3.04
LM324s (extra)	0.8	Joe	0.8
USB-C receptacles (extra)	3.32	Joe	3.32
Resistors (extra)	4.79	Joe	4.79
Shipping	13.98	FedEx	13.98
Tariffs	3.27	US Government	3.27
Sales Tax	6.7	IL Government	6.7
TOTAL			79.25

4.4 PCB Costs

PCBWAY ORDERS

PART no. and description	Size	Qty	Unit Price (USD)	Amount (USD)
Printed Circuit 2 layers Material: FR-4 Gerber_Group82 no. :W969014AS3C2	41.4*79	5	7.79	36.96
Printed Circuit 2 layers Material: FR-4 Gerber_Group82 no. :W969014AS3C3	41.4*79	5	23.54	117.70

5. Conclusion

In developing SensorFX, we demonstrated that affordable, body-driven control of digital audio effects is both technically feasible and musically empowering. Our EMG front-end reliably captured muscle activity, the ESP-NOW link delivered sub-10 ms latency over stage-scale distances, and the XIAO ESP32-S3 receiver enumerated as a class-compliant USB-MIDI device that any DAW—here, REAPER—recognizes instantly. Breadboard and bench tests confirmed clean 20 Hz–500 Hz band-pass filtering and safe, 0–3.3 V ADC levels, while live trials showed performers could modulate delay feedback, distortion mix, and pitch in real time with simple gestures—freeing their hands for their instruments. Remaining uncertainties include long-term electrode comfort, battery endurance for multi-set gigs, and robustness in RF-dense venues; ethically, we must ensure clear safety guidelines for sensor placement and electrical isolation. Future iterations would explore adaptive gain to auto-calibrate for different muscle groups, BLE-MIDI for phone/tablet workflows, and a modular enclosure system to target additional limbs. Overall, SensorFX delivers a low-cost, high-expressivity interface that integrates seamlessly into existing production pipelines and opens new creative possibilities for musicians at every level.

5.1 Accomplishments

The project was quite successful in terms of proving the viability of the open source sensor design, and improving our understanding of how this could be used to create a wearable MIDI interface. However, what we believe to be more important is that our project marks the potential for a new class of MIDI controllers, and potentially a new class of instrument. The use of muscle sensors to provide musical change has not been seen in this context before, and we believe that the feeling of naturalness and fluidity that our device brings to a musician gives it advantages over a standard MIDI controller above the fact that it can be used while playing an instrument. Furthermore, with the ability to send both CC messages as well as note triggering messages, a door could be opened for a musician to be able to use our device as an instrument itself, not just as an effect controller; while we never tested it ourselves, there isn't a reason it couldn't be done in practice. For these reasons, we believe our device was a great

success, even though we didn't get to see the full potential of it by the end of the semester.

5.2 Uncertainties and Future Works

It should be noted that we were unable to complete assembly of the V2 PCB, as it arrived on the day of the final demo, so as of now it is still untested. The buttons on this version of the PCB are certainly still non-functional as well. A third PCB revision will most likely be required to achieve adequate functionality of the project. There is also great uncertainty as to what is causing the issues in the analog circuitry that caused the V1 and V2 PCB's sensors to not work properly. This could be assessed using both a spectrum analyzer and careful construction of the V2 PCB, as well as a manual diagram of the working finalized breadboard design for verification.

5.3 Ethical considerations

There were many ethical considerations that we took into consideration. We made our device as accessible as possible to individuals with a variety of physical ailments. By allowing the device to be strapped anywhere on the body, there is no shortage of creativity for a musician to make use of our device. For instance, even a paraplegic could sing into a microphone and attach electrodes to their neck to control a desired effect. All that is required for our device to work is one singular muscle group and one singular musical instrument, granting just about any musician use of our device.

Additionally, every part of our device in both hardware and software was carefully selected to be as affordable as possible. The basis of our design, OpenEMG, is possibly the cheapest self-assemblable EMG device on the market. Our chosen DAW, REAPER, has extremely low device specifications and is functionally free to use. The electrodes that we used in any testing environments were the cheapest electrodes we could find in bulk on Amazon. In short, we made our device from the bottom-up to be the most affordable it could possibly be.

References

[1] J. Chadabe, "The Electronic Century, Part IV: The Seeds of the Future," *Electronic Musician*, May 1, 2000. [Online]. Available: https://web.archive.org/web/20120928230435/http://www.emusician.com/gear/0769/the-electronic-century-part-iv-the-seeds-of-the-future/145415. [Accessed: May 7, 2025].

[2] MasterClass, "What Is a DAW? A Guide to Digital Audio Workstations," MasterClass, Aug. 25, 2021. [Online]. Available: https://www.masterclass.com/articles/what-is-a-daw. [Accessed: May 7, 2025].

[3] Charles Labs, "Project – OpenEMG Arduino Sensor," *Charles Labs*, n.d. [Online]. Available: https://charleslabs.fr/en/project-OpenEMG+Arduino+Sensor. [Accessed: May 7, 2025].

[5] Espressif Systems, “ESP32-C3 Datasheet,” *Espressif Systems*, n.d. [Online]. Available: https://www.espressif.com/en/products/socs/esp32-c3. [Accessed: May 7, 2025].

[6] Espressif Systems, “ESP-NOW,” *ESP-IDF Programming Guide*, n.d. [Online]. Available: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/network/esp_now.html. [Accessed: May 7, 2025].

[7] Espressif Systems, “ESP-IDF USB Device Documentation,” *Espressif Systems*, n.d. [Online]. Available: https://docs.espressif.com/projects/esp-idf/en/latest/esp32c3/api-reference/peripherals/usb_device.html. [Accessed: May 7, 2025].

[8] Ha Thach, “TinyUSB: MIDI Test Example,” *GitHub*, n.d. [Online]. Available: https://github.com/hathach/tinyusb/tree/master/examples/device/midi_test. [Accessed: May 7, 2025].

[9] Cockos Inc., “REAPER – Digital Audio Workstation,” 2025. [Online]. Available: https://www.reaper.fm/. [Accessed: May 7, 2025].

[10] MIDI Manufacturers Association, “MIDI 1.0 Detailed Specification,” *MIDI Manufacturers Association*, n.d. [Online]. Available: https://www.midi.org/specifications-old/item/the-midi-1-0-specification. [Accessed: May 7, 2025].

[11] USB Implementers Forum, “Device Class Definition for MIDI Devices,” *USB Implementers Forum*, 1999. [Online]. Available: https://www.usb.org/document-library/device-class-definition-midi-devices. [Accessed: May 7, 2025].

[12] Espressif Systems, *ESP32-S3 Series Datasheet*, ver. 1.6, 2023. [Online]. Available: https://files.seeedstudio.com/wiki/SeeedStudio-XIAO-ESP32S3/res/esp32-s3_datasheet.pdf. [Accessed: May 7, 2025]

[13] Expressif Systems, ESP32-C2-DevkitC-02 Docs https://docs.espressif.com/projects/esp-idf/en/v5.2/esp32c3/hw-reference/esp32c3/user-guide-devkitc-02.html, https://dl.espressif.com/dl/schematics/SCH_ESP32-C3-DEVKITC-02_V1_1_20210126A.pdf

Appendix A Requirement and Verification Table

Table A System Requirements and Verifications

Requirement	Verification	Verification status (Y or N)
1. Sensor Array Generates proper signals a. charge pump functions b. Sensor array responds correctly c. Final smoothing filter functions	1. Verification a. Measure for negative voltages b. Feed test signal of variable Hz sine wave c. Verify that heartbeat/pure signal is visible before smoothing, and is correctly smoothed (oscilloscope used)	Y Y Y
2. Wireless signals are transmitted a. Transmitter board sends messages b. Receiver board receives messages c. Range and transmission speed are adequate	2. Verification a. Observe Serial Monitor Output b. Observe Serial Monitor Output c. Observe Serial Monitor Output and move both boards very far away from eachother and make sure messages are still sent and received.	Y Y Y
3. MIDI protocol recognized in DAW a. DAW Registers MIDI device as MIDI controller. b. MIDI commands successfully interpreted by DAW	3. Verification a. Observe device being registered in DAW b. Observe MIDI Commands cause audio changes while performing	Y Y

Appendix B Figures

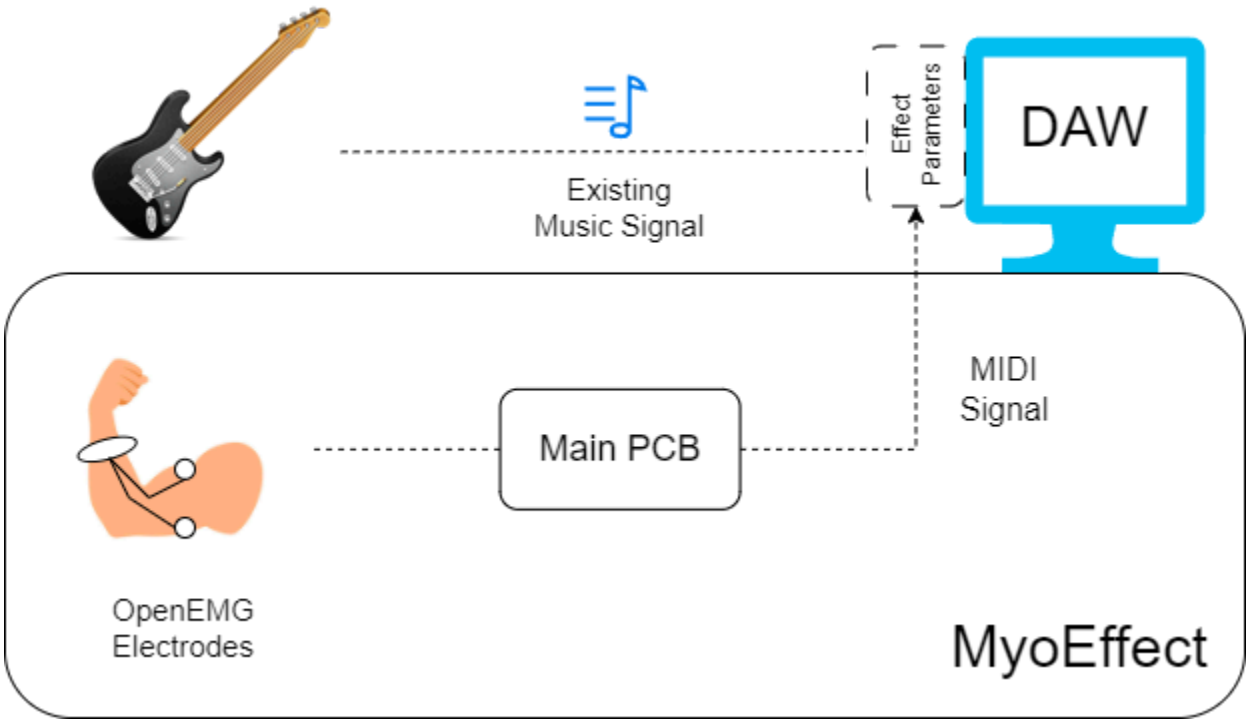

Figure 1. A figure demonstrating how the instrument and the EMG sensors connect to the DAW.

ESP-NOW	Bluetooth
Proprietary Standard – ESP Only	Open Industry Standard
Somewhat Uncommon	Very Common
Low Latency (5 ms)	High Latency (up to 300 ms)
Long Range (200+ ft)	Limited Range (20-30 ft)

Figure 2. Pros and Cons of ESP-NOW vs Bluetooth illustrated in a table.

Figure 3. Shenzhen MyAntenna M01-0600890R0A

```
// Structure to receive data
// Must match the sender structure
typedef struct struct_message {
    char message[32];
    unsigned char data;

} struct_message;
```

Figure 4. The structure sent between the transmitter and receiver.

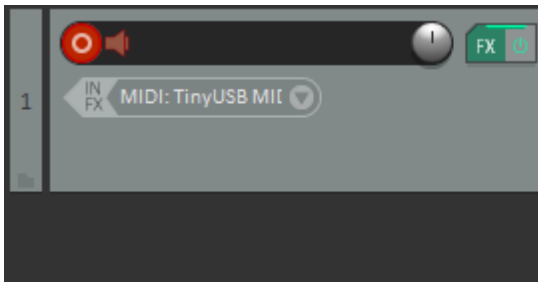

Figure 5. Device being seen by DAW as 'TinyUSBMIDI.'

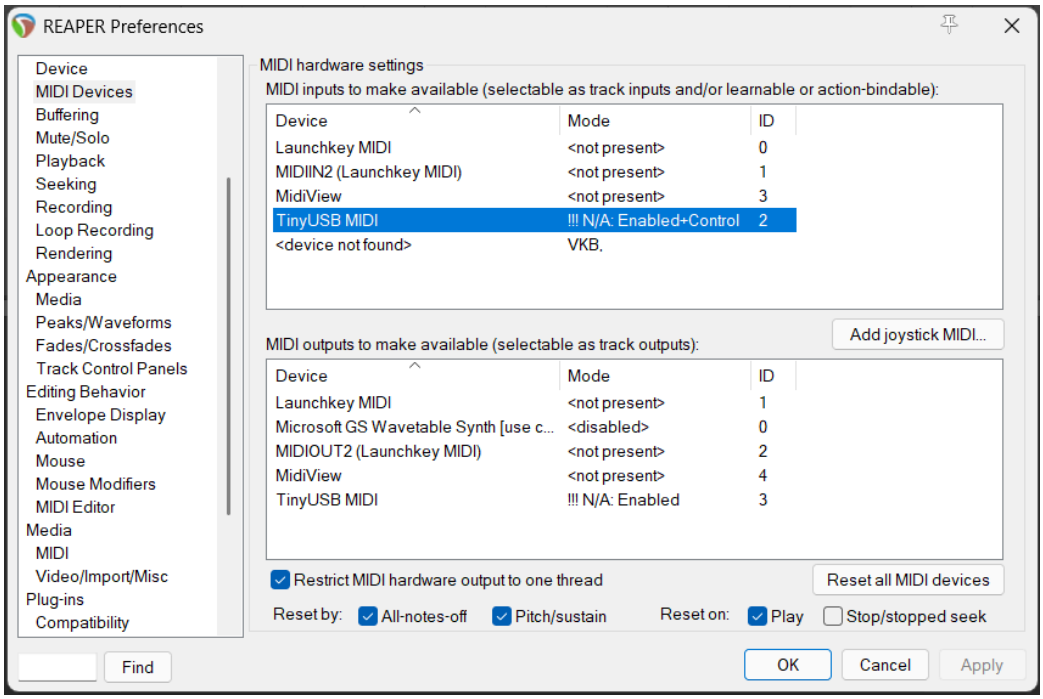

Figure 6. REAPER MIDI Devices settings window and the enabling of the TinyUSBMIDI device as an effect controller.

Figure 7. Effect UI for the Klangheim IVGI2 Distortion Pedal plug-in.

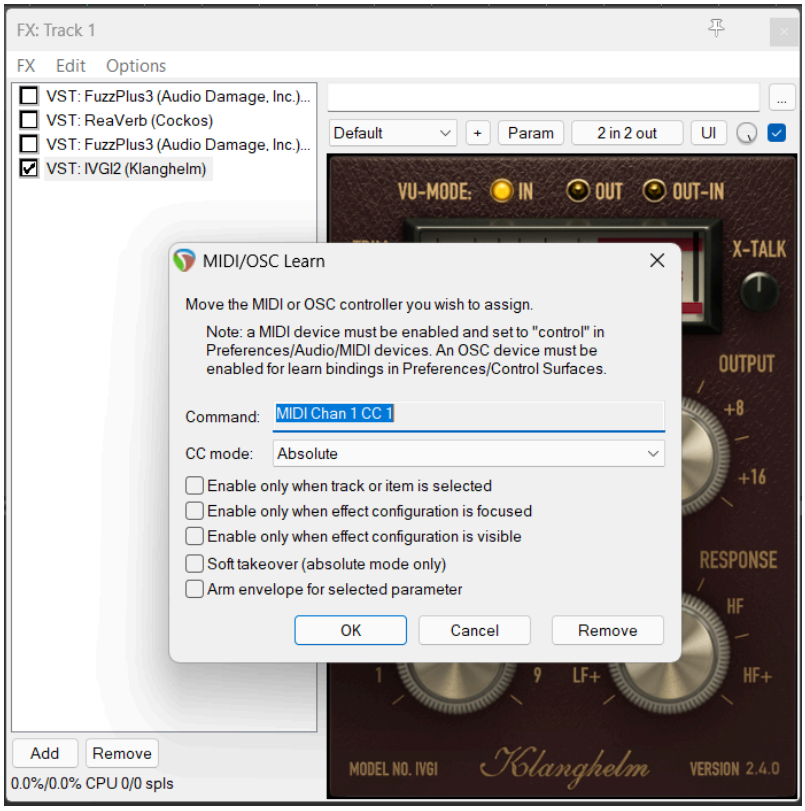

Figure 8. MIDI Learn UI page and the automatic detection of CC messages from TinyUSBMIDI.

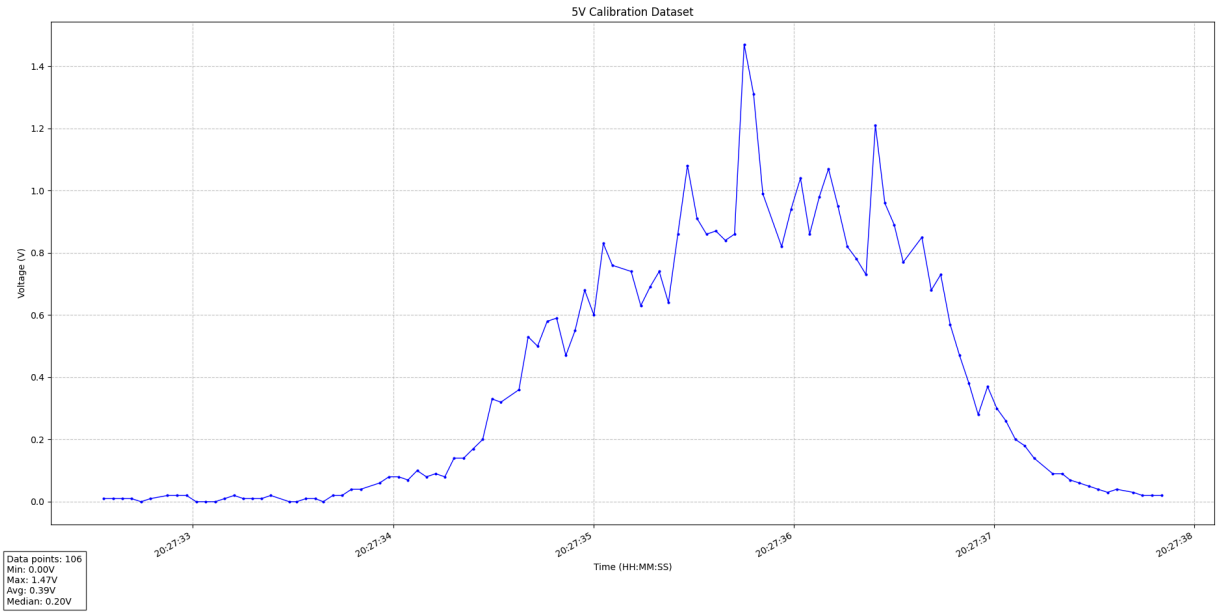

Figure 9. Electrode output voltage readings operating at 5V.

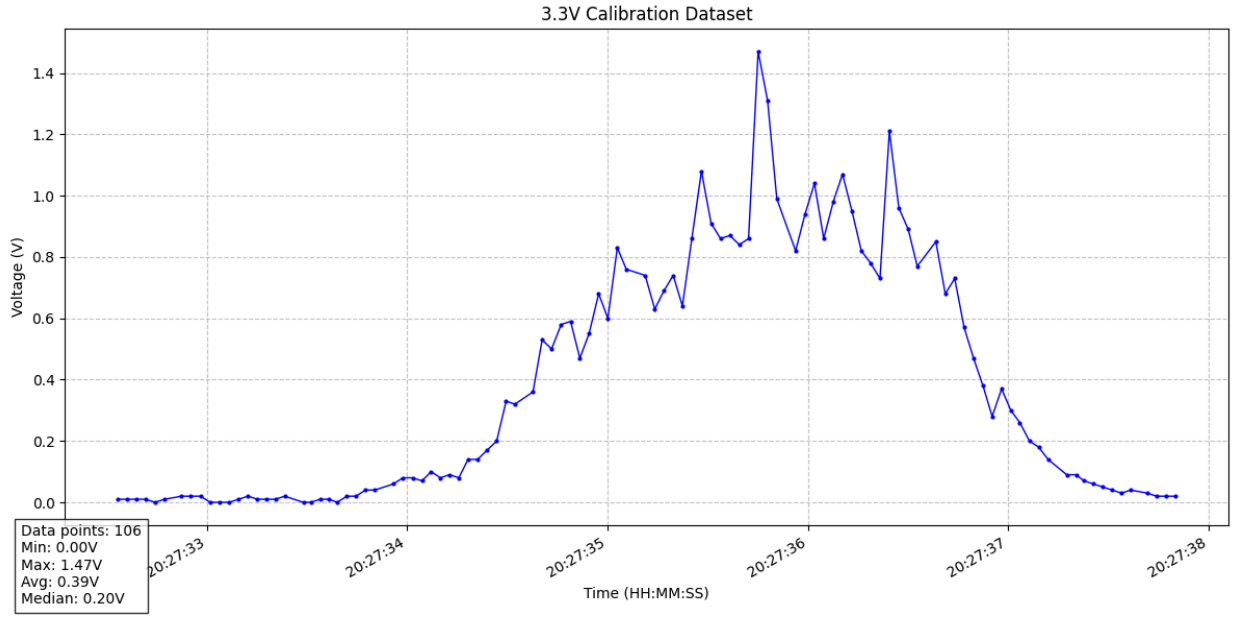

Figure 10. Electrode output voltage readings operating at 3.3V.

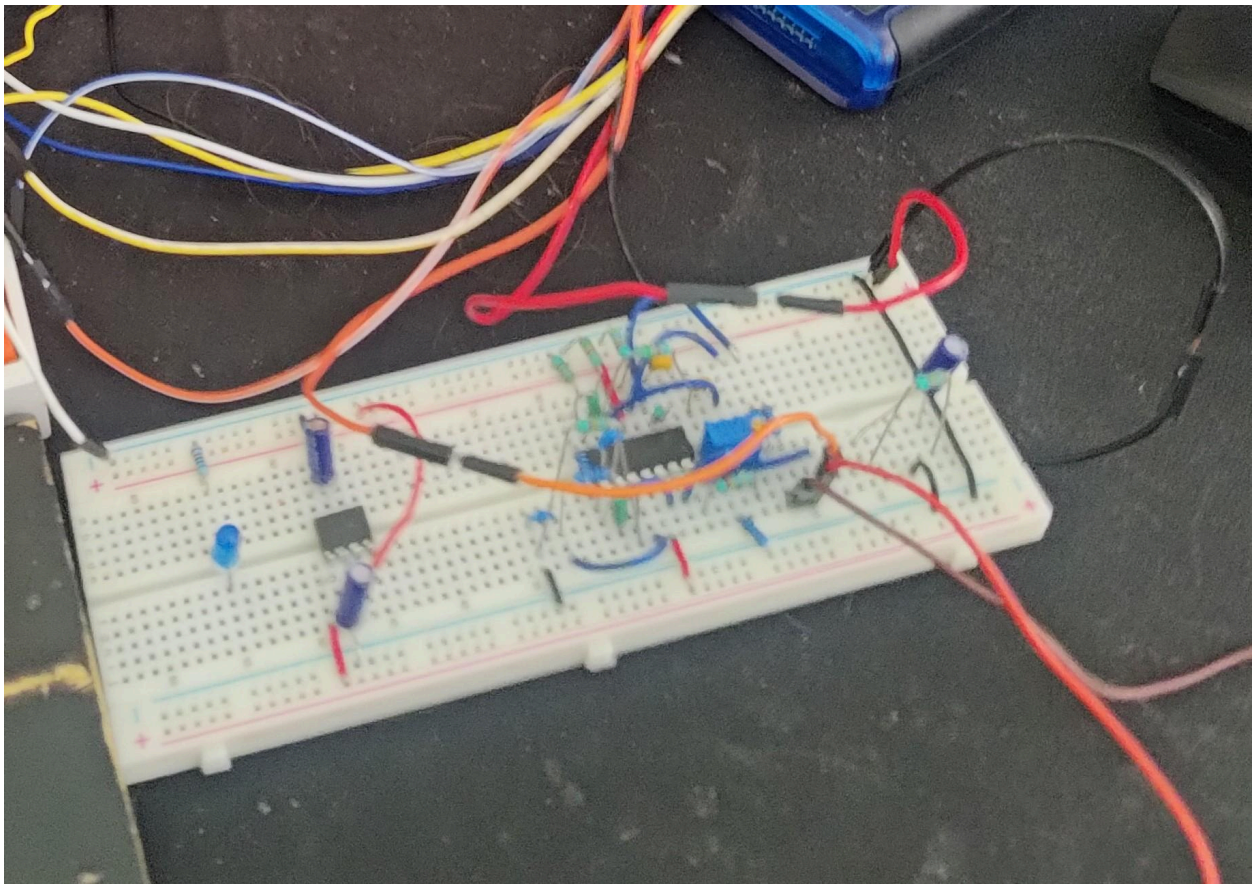

Figure 11. A Picture of the breadboard used in the breadboard.

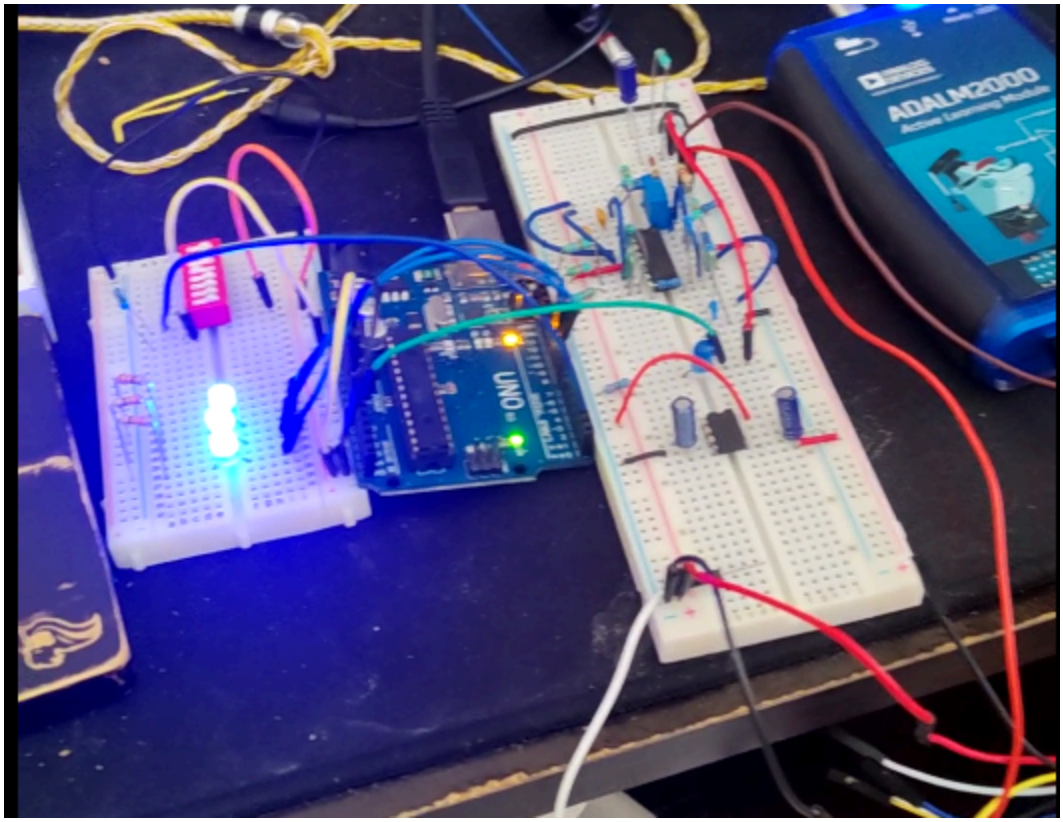

Figure 12. Breadboard demo with Arduino programming circuitry.

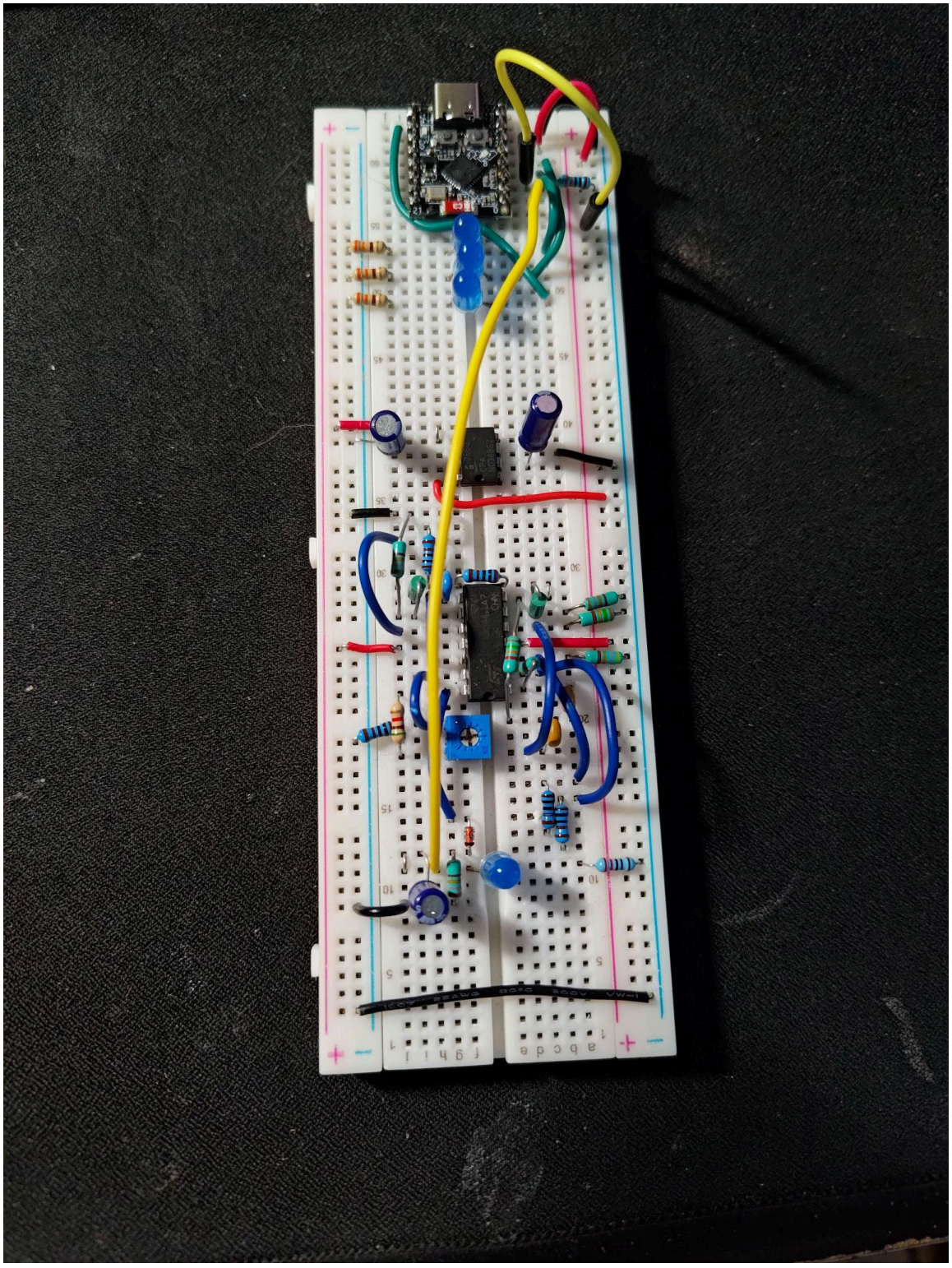

Figure 13. Final breadboard iteration of transmitter board.

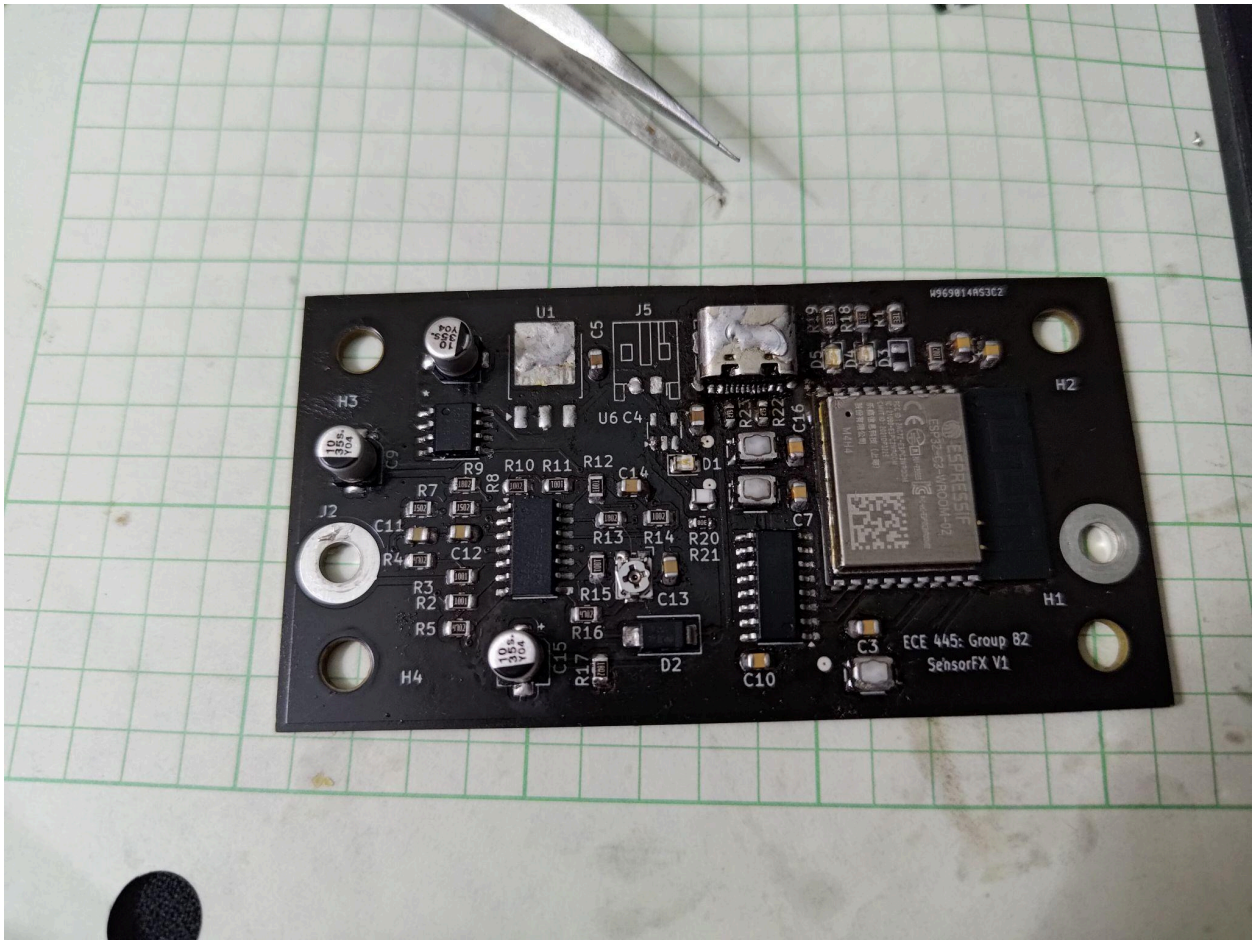

Figure 14. Attempts to physically lift the USB-C port of PCB V1.

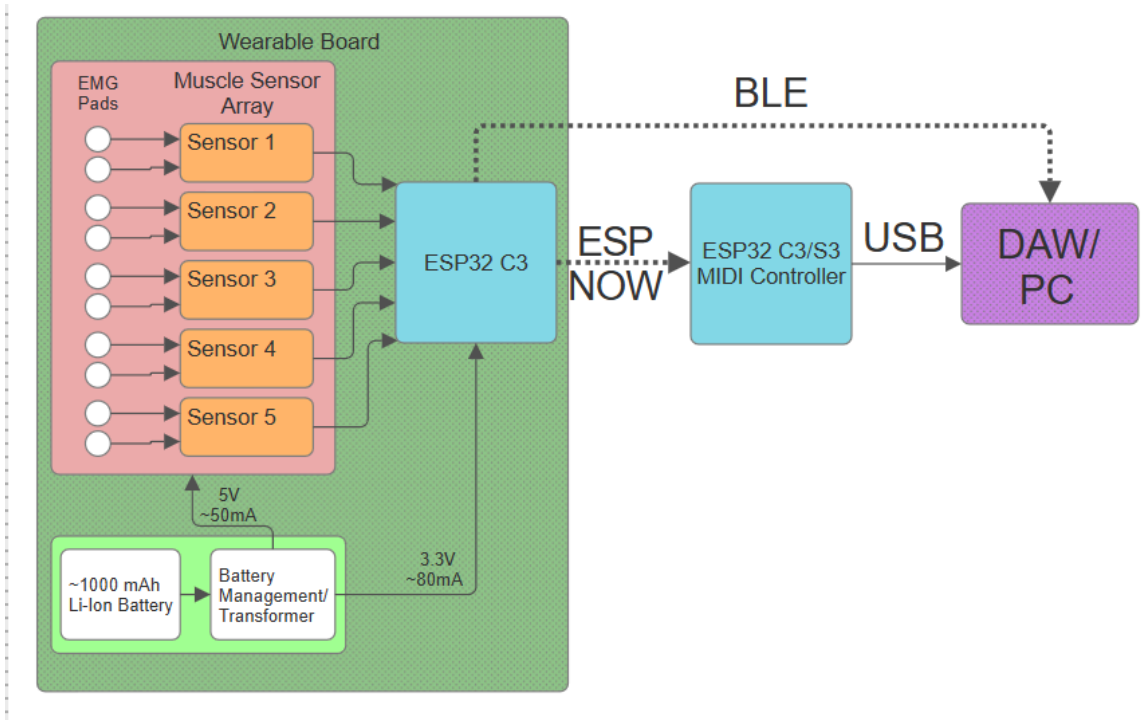

Figure 15. SensorFX Full Signal Chain.

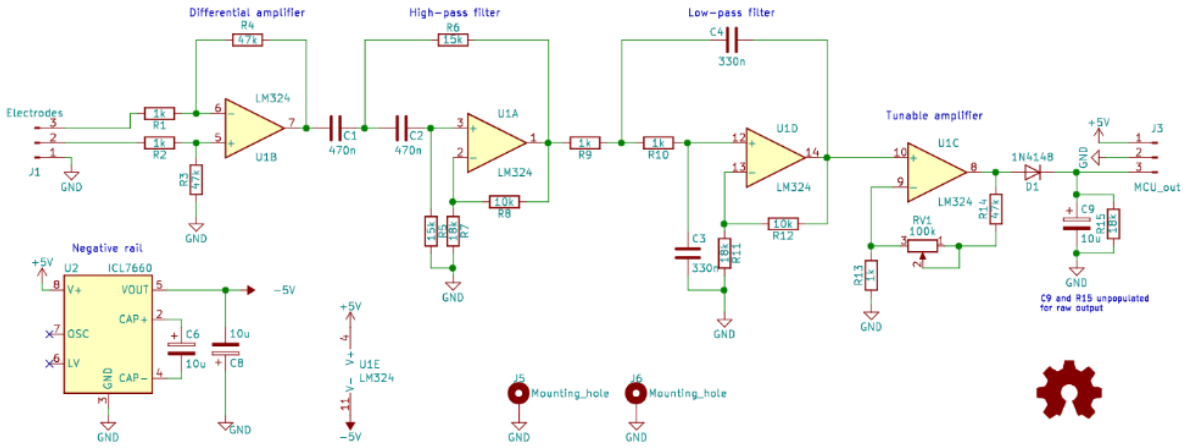

Figure 16. Original OpenEMG Schematic.

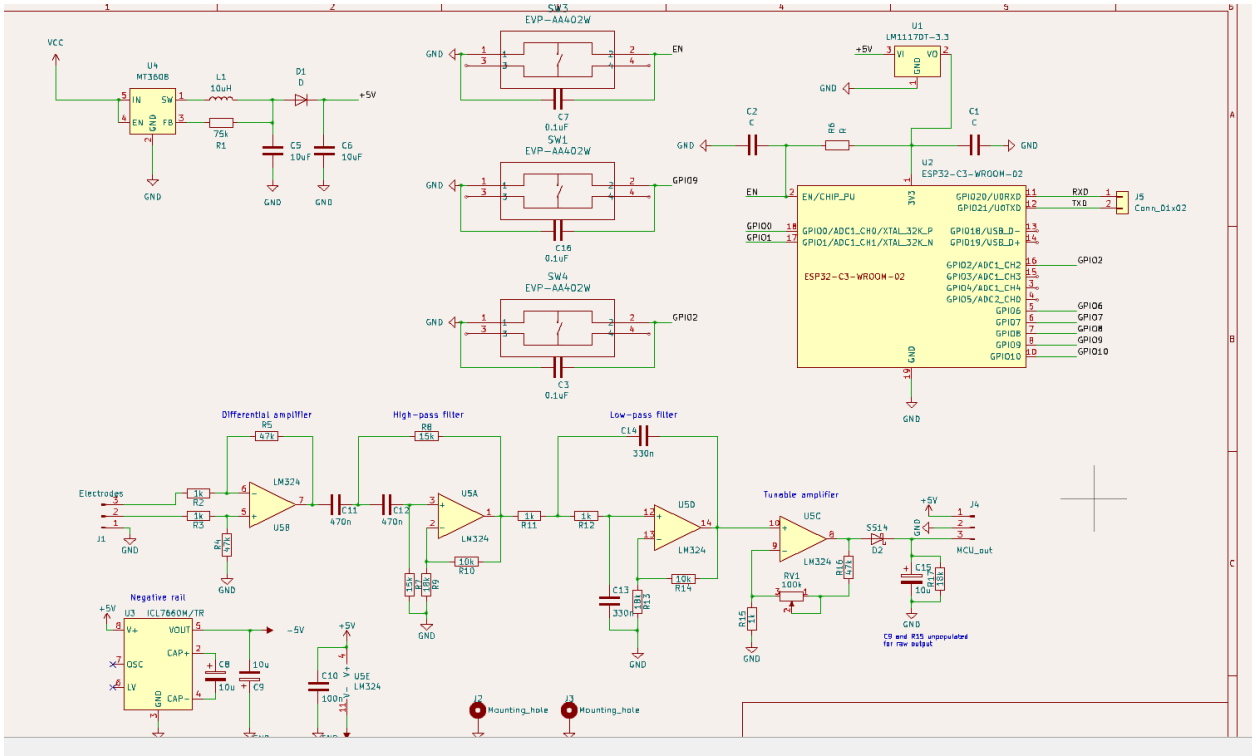

Figure 17. Schematic of PCB V0 with a step up converter for 5V still present.

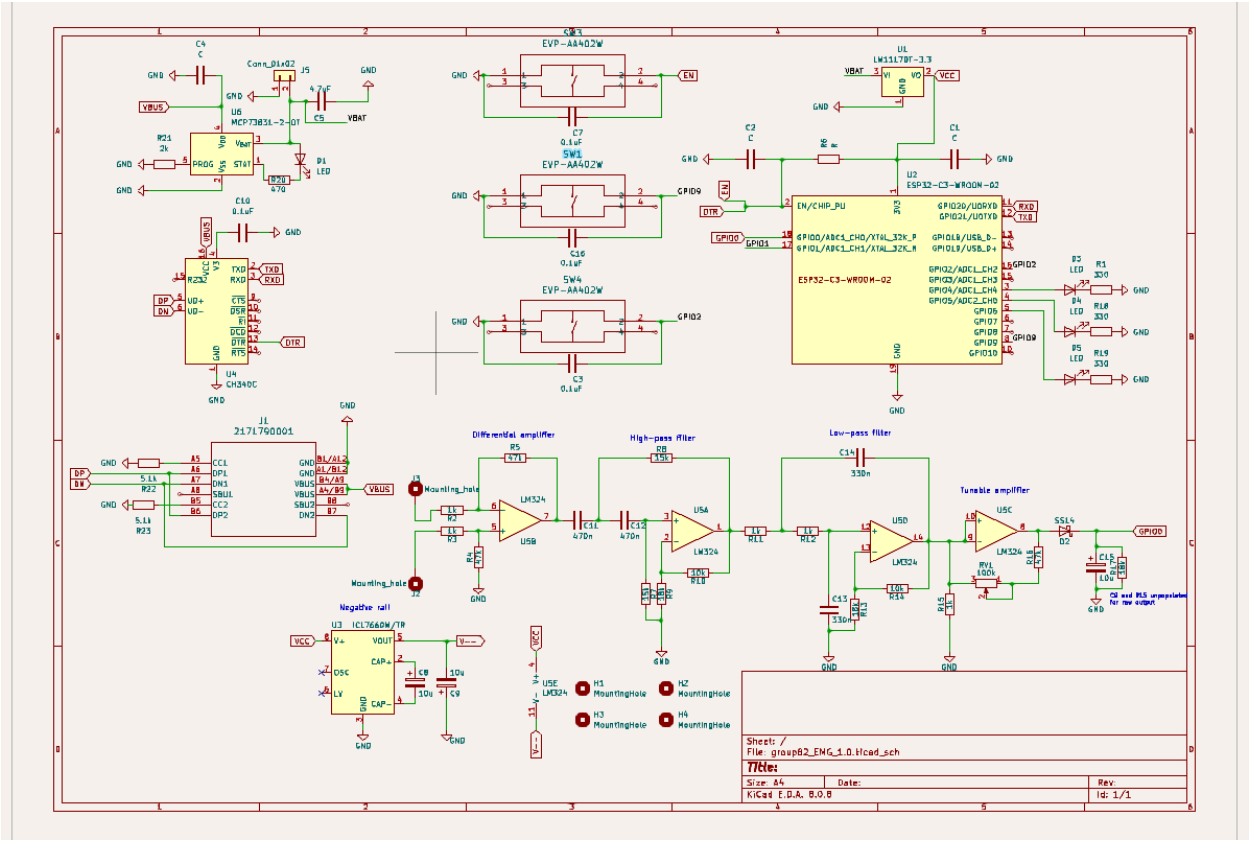

Figure 18. Schematic of PCB V1.

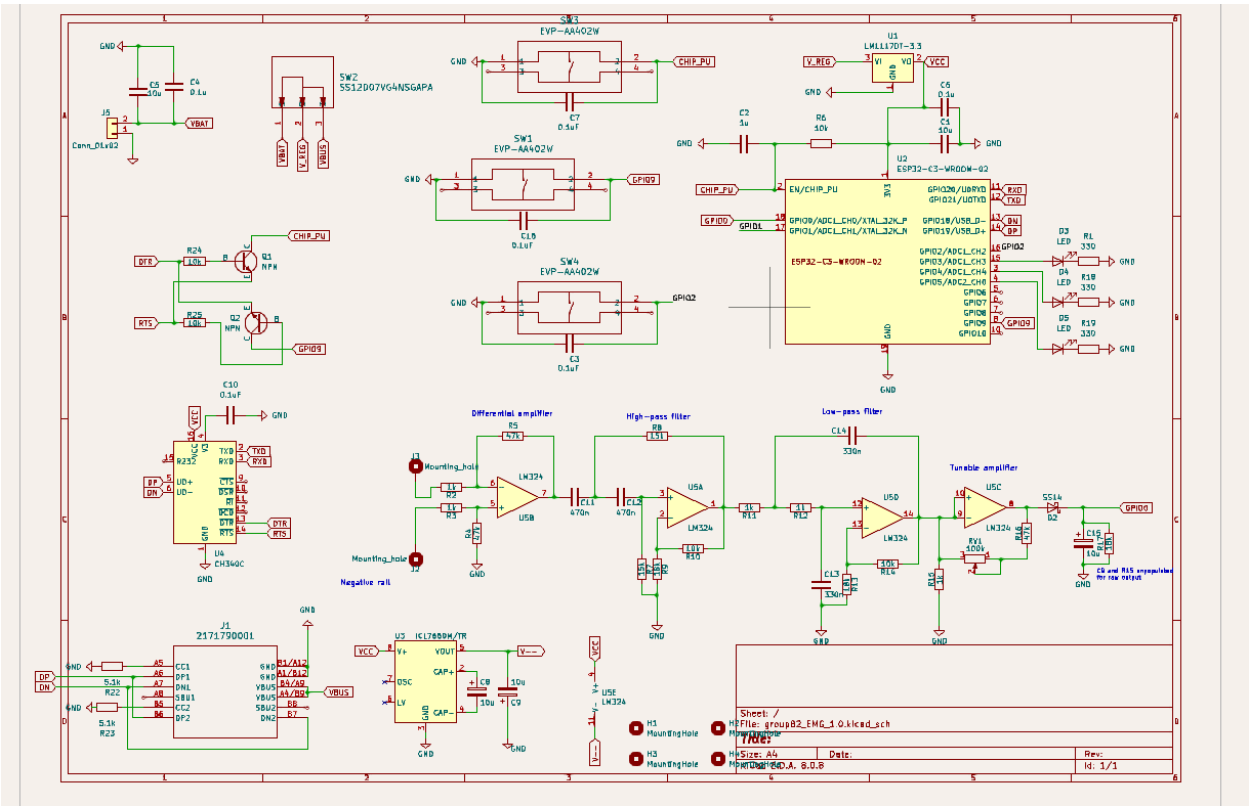

Figure 19. Schematic of PCB V2.

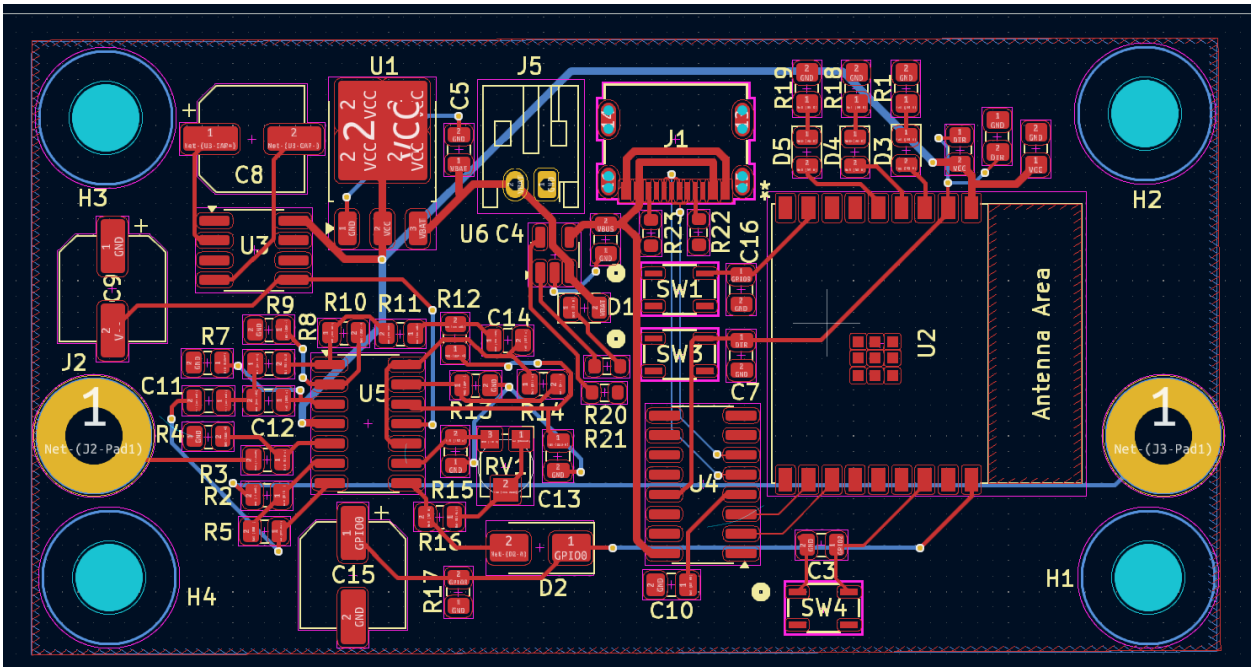

Figure 20. Final KiCAD schematic of PCB V1.

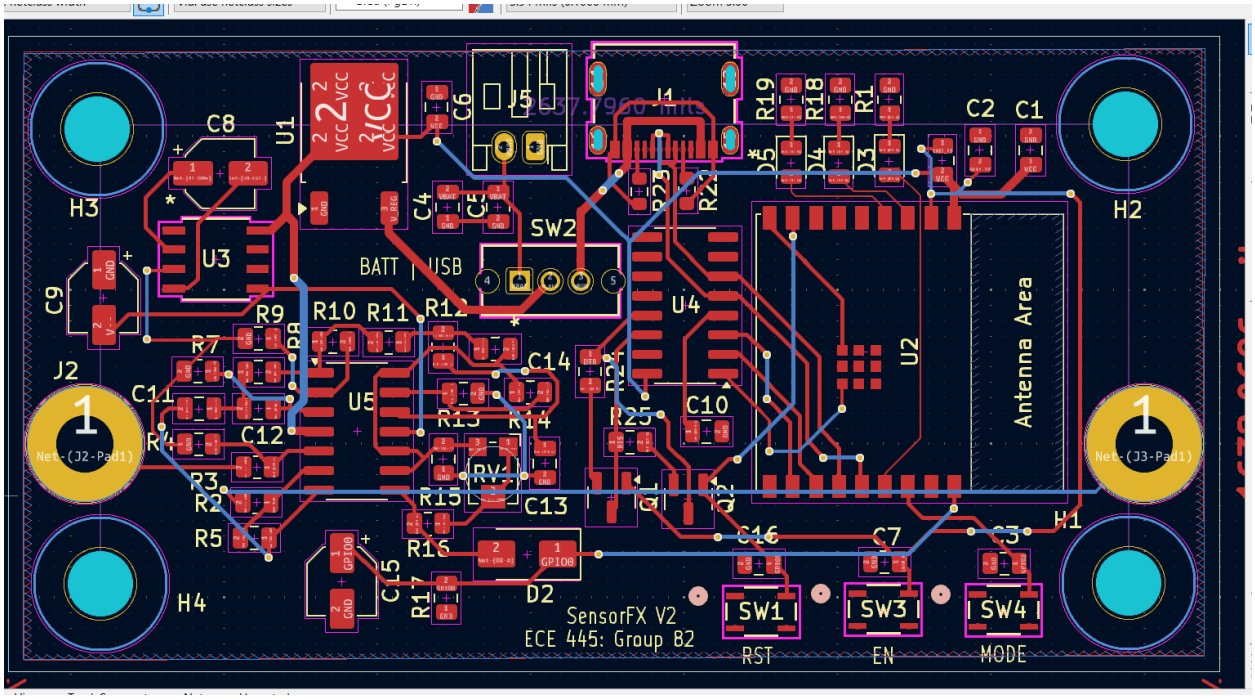

Figure 21. Final KiCAD schematic of PCB V2.

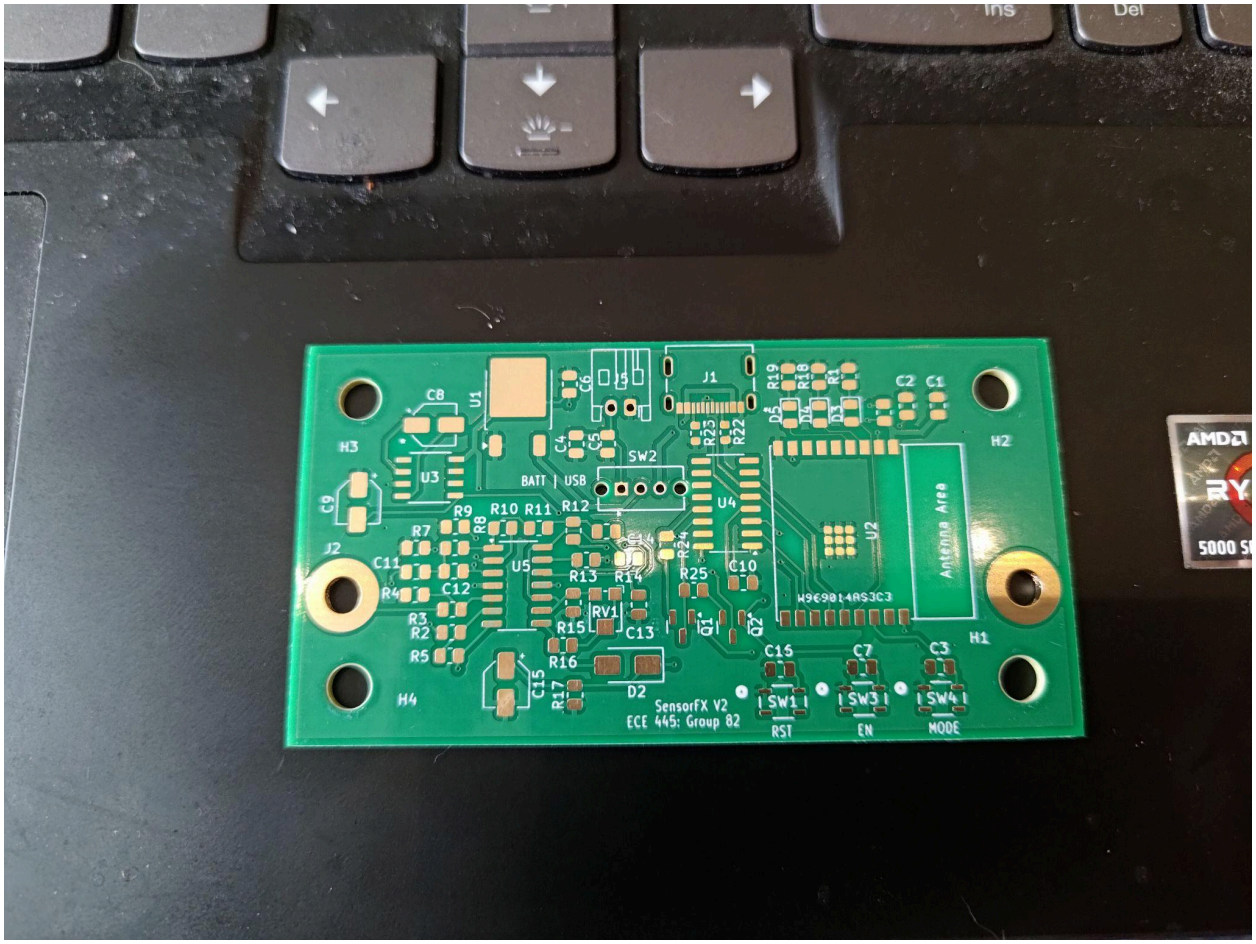

Figure 22. Final iteration of PCB.