MyoEffect: Muscle Controlled MIDI Device

ECE 445 Design Document - Spring 2025

Team #82

Paul Matthews, Joseph Schanne, Karan Sharma

Professor: Arne Fliflet

TA: Eric Tang

Contents

1 Introduction **3**
1.1 Problem 3
1.2 Solution 3
1.3 Visual Aid 4
1.4 High Level Requirements 5
2 Design **6**
2.1 Block Diagram 6
2.2 Subsystem Overview 7
2.3 Cost Analysis and Schedule 17
3 Ethics **19**
4 References **22**

1 Introduction

1.1 Problem

Most musicians working in digital audio workspaces (DAWs) desire fine control over the effects they use and the intensity at which those effects operate. Most of the time, in order to quickly assign effects to controls and to fine tune them, musicians will make use of MIDI controllers; however, these controllers are often both clunky to use and expensive to obtain.

1.2 Solution

Our design offers a cheaper and more expressive method of using effects by bypassing the use of a separate controller entirely and controlling effects with the motion of the musician's own body. Instead of having to turn knobs or press buttons, a musician can simply close their fist, or extend their arm, or twist their leg; our design's goal is to give complete freedom to the musician as far as what motion they want to control their effects. Our design makes use of electromyography (EMG) sensors to map contractions of users' muscles to voltage readings; these readings will then be converted into corresponding MIDI parameters. Our device can then be plugged into a DAW and operate in exactly the same way as a normal MIDI controller.

As far as our implementation of this idea, we will create a bracelet or watch-like attachment to house the circuitry needed for our EMG sensors. These attachments will be designed to be easily

adjustable and movable for reading inputs from arm and leg muscle groups. Later sections will talk about how many sensors each attachment will house and use, as well as the trade-offs for each implementation.

Our process for converting the analog voltage read from the EMGs to a corresponding MIDI signal is as follows: read the analog voltage from the EMG, convert it to a digital signal with an onboard ADC from our ESP32 chip, map that digital value to a corresponding MIDI parameter intensity, and then wrap that parameter in the correct MIDI communication protocol. The specifics of exactly how this process will be implemented is explained in later sections.

1.3 Visual Aid

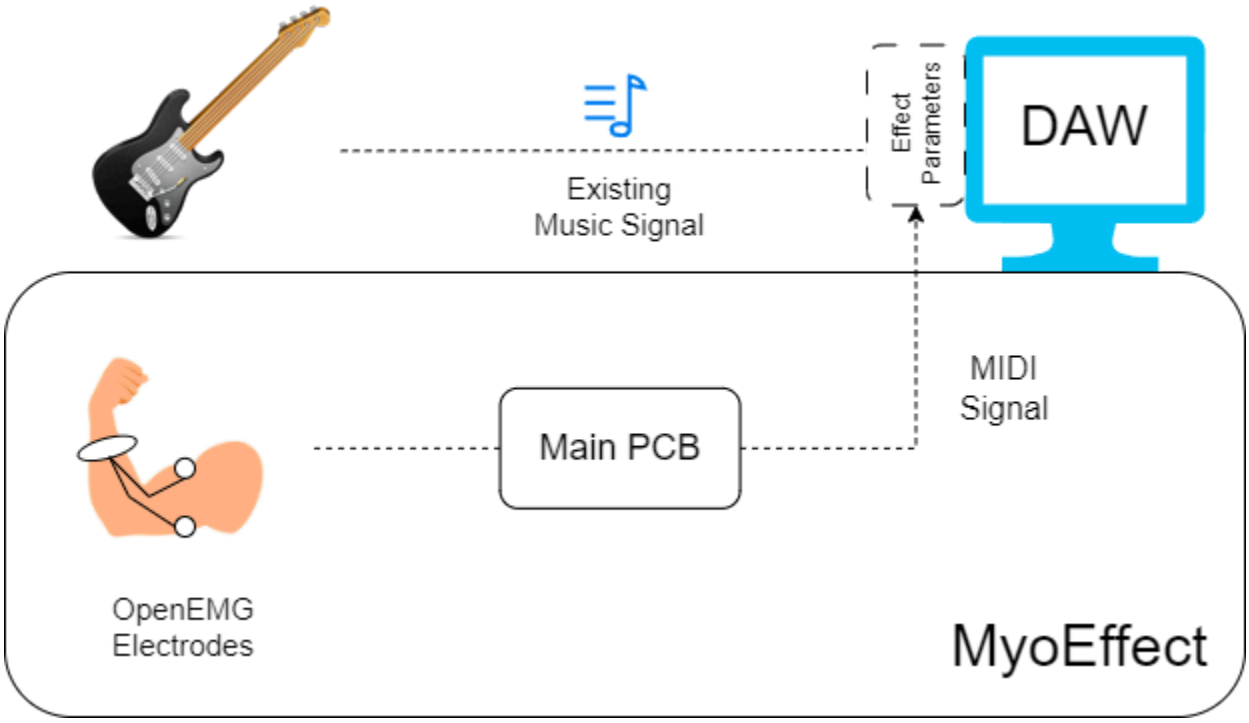


Figure 1: Visual depiction of high level functionality

1.4 High Level Requirements

- Requirement 1: Trigger one effect (ex: MIDI code 0x90, play note) based on a contraction of any given muscle.
- Requirement 2: Device must be compliant to all communication protocols that we plan to use, including (but not limited to) ESP NOW, USB, and/or Bluetooth.
- Requirement 3: EMG sensors must be tunable to each muscle group. Each muscle group will exhibit different threshold voltages based on the size of the muscle. Therefore, the gain of our sensors must be adjustable either automatically or by the user in order to account for this.

2 Design

2.1 Block Diagram

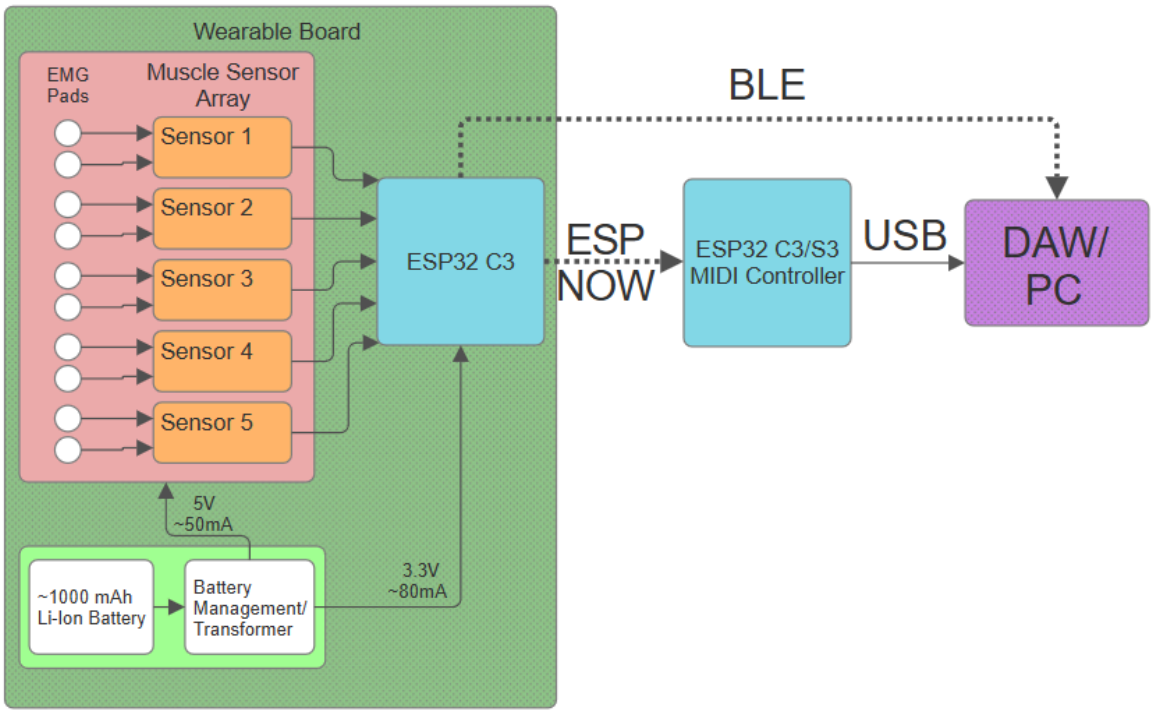
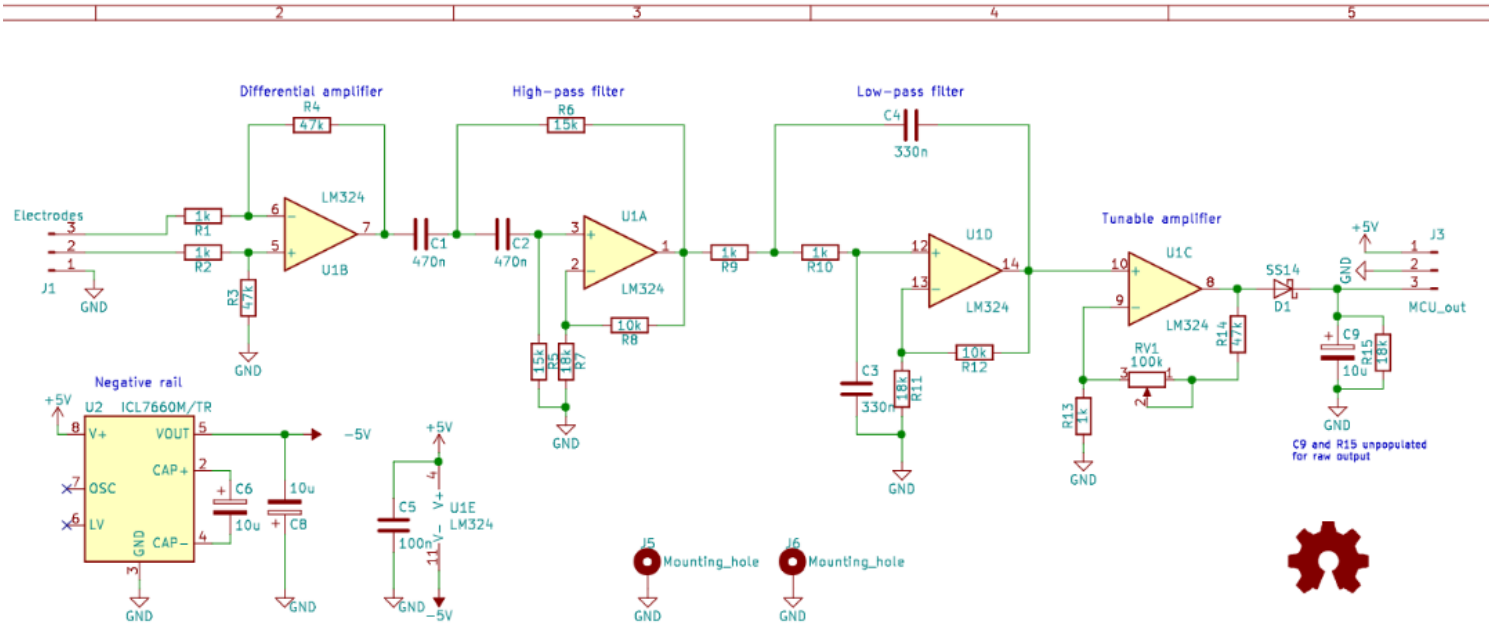


Figure 2: Block diagram of sensor array module, ESP32 transmitter module, and necessary connections between modules and PC

2.2 Subsystem Overview

Sensor Schematic



Overview

The wearable sensor array will be a watch-type design with multiple muscle sensor interfaces aligned on the skin facing side of the device. Each individual sensor will interface directly with its own channel of the ESP32 ADC. Sensors can either be distributed evenly on the same bracelet or be made into their own individual boards and bracelets, with the ability to connect or disconnect these “modular” sensors to the main ESP32 C3 board.

The wearable sensor array is a critical component of the system, responsible for capturing muscle contraction signals (EMG) from the user’s arms and legs. The array consists of multiple sensors, each interfacing with its own ADC channel on the ESP32 C3 microcontroller. The sensors are designed to be modular, allowing them to be distributed evenly on a bracelet or

connected individually to the main board. Each sensor includes EMG pads, a differential amplifier, and a series of filters to ensure the signal is clean and within the acceptable voltage range for the ESP32 ADC.

Requirements and Verification

- Output Voltage: The output of the sensor system must not exceed 3.3V. Diode D1 in the schematic handles this.
    - Verification: With a signal generator set to an impulse (modeling a heartbeat) as the input, measure the output signal and verify that it does not exceed 3.3V under any circumstances.
- Minimum Detection: Sensor must be able to pick up and amplify a muscle signal of amplitude ~1mV
    - Verification: Feed the sensor array a range of voltages from 1-2 mV with a frequency in the pass band of the filters (100 Hz), record the output voltages and compare with expectations. Adjust amplification as necessary.
- Power Consumption: Chosen batteries/supply must be able to sustain the 10mA of consumption that each sensor requires for at least a few hours.
    - Measure the current flowing through the sensor array for a few minutes and find the average draw. Use this to estimate battery draw.
- Filtering: LPF and HPF should exhibit correct amplification (cutoff) behavior at sub 20 Hz, and greater than 500Hz
    - Verification: Feed input signals of 0-20Hz and 500-750Hz into the sensor module, verify that these signals are not allowed to pass.

- Gain Control: Verify that output gain is correct and produces appropriate signals without clipping
    - Hook up a control signal or real electrodes to the sensor, measure output voltages on an oscilloscope and record minimum and maximum voltage readings. Ensure that the unflexed muscle is as close to 0V as possible, and make sure that the full flexion of the muscle does not exceed 3.3V.

Tolerance Analysis

Using the 2 EMG pads, each sensor will first determine the differential in voltage at the sites of the pads and amplify the signal using a differential Op-Amp. After this, the signal passes through a second order active high pass filter with a cutoff frequency of ~22.575 Hz and a pass band gain of 1.55, and a second order low pass filter with a cutoff frequency of 482.288 Hz and pass band gain of 1.55. This step filters our raw signal into something more usable, as muscle impulses are known to occur in 0-500 Hz frequency band, with an amplitude of around 0.1 to 1 mV depending on the muscle being measured. After the signal is filtered, and at a voltage of around 10-100 mV, the final tunable opamp is used to scale up the output because different muscles have different sizes, and thus different voltage ranges. It should be noted that this may be too much gain, as the ESP32 can only read analog signals up to 3.3V, so we opted for a 10k resistor value for R_{14}. The calculations for this section can be seen below.

$$\frac{10k}{1k} < A_{tuner} < \frac{110k}{1k} \sim 10 < A_{tuner} < 110$$

$$A_{Diff} = \frac{47k}{1k} = 47$$

$$A_{Filters} = 1 + \frac{10k}{18k} = 1.55$$

$$f_{HPF} = \frac{1}{2\pi\sqrt{R_5 R_6 C_1 C_2}} = \frac{1}{2\pi\sqrt{(15*10^3)^2(470*10^{-9})^2}} = 22.575\ Hz$$

$$f_{LPF} = \frac{1}{2\pi\sqrt{R_9 R_{10} C_3 C_4}} = \frac{1}{2\pi\sqrt{(1*10^3)^2(330*10^{-9})^2}} = 482.288\ Hz$$

Battery Management System

Overview

The power management system is critical for safely managing the power supply to the ESP32 microcontroller and the wearable sensor array. The system uses a 3.7V Li-Ion battery to power the ESP32 and the sensor array. Since the sensor array requires 5V, a boost converter is used to step up the voltage from 3.3V to 5V. The BMS ensures safe charging, discharging, and voltage regulation for both the ESP32 and the sensor. This system will use a TP4056 or similar battery charger circuit, along with a PAM2401 Boost converter to generate a reliable 5V signal which can be fed to the sensor array.

Requirements and Verification

- Output Voltage: Boost converter and battery charger must not generate excessive heat, and should output a reliable 3.7V and 5.0V signal simultaneously.
    - Verification: Apply 3.7V constant DC signal to the PAM2401 circuitry, and ensure that the output is consistently 5.0V. Hook up a sensor array and measure current draw from the battery to the boost converter to the sensors.

- Charge-discharge reliability: Because this device will be attached to the user's body, it is of paramount importance that the battery not fail under any circumstances
    - Verification: Discharge battery fully, and wire it into the TP4056. Apply 5.0V USB charging to the board, measure the current to the battery over the full charge cycle. After it reaches 3.7V, it should stop receiving current. Measure Charge cycle time. After full charge, leave the array running at idle with an arbitrary signal input and measure time to discharge. Ensure that discharge time is above 2-3 hours. Make sure that no components during the full charge-discharge cycle are heating up excessively.
- Power Consumption: Chosen batteries/supply must be able to sustain the ESP32 and sensors for 2-3 hours at least.
    - Verification: Complete assembly of the BMS system, use it to power both the ESP32 dev board and sensor module. Ensure expected power draw to both the ESP32 and sensors while broadcasting over ESP-NOW to the receiver module.

Wearable ESP32-C3

The ESP32 C3 microcontroller serves as the central processing unit for the wearable sensor array. It continuously reads input from the sensor array, processes the signals, and translates them into MIDI commands for use in digital audio workstations (DAWs) or other music production software. The ESP32 C3 has 6 separate ADC channels, allowing for up to 6 sensors per controller. The system supports both BLE (Bluetooth Low Energy) and ESP-NOW wireless communication protocols, with the final choice depending on bandwidth and range requirements

Requirements and Verification

- ADC Sampling: The ESP32 C3 must continuously read input from at least one muscle sensor channel at a sampling rate of 1kHz.
    - Verification: Use a signal generator to simulate muscle signals and measure the sampling rate using the ESP32's internal timer. Confirm that the sampling rate is consistently 1kHz

- **MIDI Command Generation:** The system must translate muscle signals into MIDI commands (e.g., note on/off, parameter changes).
    - **Verification:** Simulate muscle signals of varying intensities and verify that the ESP32 generates the correct MIDI commands (e.g., 0x90 for note on, 0x80 for note off).
- **Wireless Communication:** The system must support both BLE and ESP-NOW protocols for data transmission.
    - **Verification:** Test the wireless communication range and data transfer speed for both BLE and ESP-NOW. Confirm that ESP-NOW achieves a range of at least 200 meters and a data transfer rate of 214-512kbps.
- **Data Integrity:** The system must ensure that MIDI commands are transmitted without corruption.
    - **Verification:** Transmit MIDI commands over ESP-NOW and verify that the receiver correctly interprets the commands.
- **Power Consumption:** The ESP32 C3 must operate efficiently to minimize power consumption.
    - Verification: Measure the current draw of the ESP32 C3 during operation and confirm that it is within acceptable limits for the chosen battery.

Tolerance Analysis

- Sampling Rate: The ESP32 C3 must sample each sensor at 1kHz to ensure accurate signal capture.
    - With 6 sensors, the total data rate is (3 bytes per MIDI command) * (1000 samples/s) * 6 sensors = 18,000 bytes/s (144kbps), well within the ESP-NOW bandwidth.
- Wireless Range: ESP-NOW must achieve a range of at least 200 meters in long-range mode.
    - Verification: Test the wireless range in an open area and confirm that data transmission is reliable up to 200 meters.

ESP32-C3/S3 Receiver

Overview

The ESP32-C3/S3 receiver board is responsible for receiving data from the wearable sensor controllers and interfacing with a PC via USB. This subsystem is necessary if ESP-NOW is chosen as the wireless communication protocol, as BLE does not require a separate receiver board. The receiver board acts as a router, handling data from one or more sensor controllers and wrapping MIDI commands in USB protocol standards for compatibility with music production software.

Requirements and Verification

- **ESP-NOW Routing:** The receiver must act as a properly configured ESP-NOW router, capable of handling data from multiple sensor controllers simultaneously.
    - **Verification:** Connect multiple sensor controllers to the receiver and verify that it can handle data from all controllers without loss or corruption.

- **USB Compatibility:** The receiver must wrap MIDI commands in USB protocol standards for compatibility with PCs.
    - **Verification:** Connect the receiver to a PC and verify that MIDI commands are correctly interpreted by the DAW.
- **Data Integrity:** The receiver must ensure that MIDI commands are transmitted without corruption.
    - **Verification:** Transmit MIDI commands from the sensor controllers to the receiver and verify that the commands are correctly interpreted by the PC.

Tolerance Analysis

- **Data Handling:** The receiver must handle data from multiple sensor controllers without exceeding the ESP-NOW bandwidth. With 6 sensors transmitting at 1kHz, the total data rate is 144kbps, well within the ESP-NOW bandwidth of 214-512kbps.
    - **Verification:** Test the receiver with multiple sensor modules to ensure concurrent functionality.
- **USB Protocol:** The receiver must comply with USB protocol standards for MIDI command transmission.
    - **Verification:** Test the receiver with multiple DAWs to confirm compatibility.

MIDI / USB Compliance

Overview

Once the EMG detects a measurable muscle movement, there are additional steps required to register the system as a MIDI compliant device over USB. Our chosen microcontroller (ESP-32

C3) already has native USB Serial functionality; this means that the steps necessary for registering our device as a MIDI over USB device is relatively simple. Once the correct pins on the ESP are properly connected as a differential pair (GPIO18 and 19 for USB D+ and D- pins), in addition to providing proper power (USB specifies 5V despite ESP being powered with 3.3V), we will register our device with Espressif's TinyUSB stack. TinyUSB is an open source USB stack that supports multiple device classes, including USB. Registering the device just requires an installation of ESP-IDF (making sure it's the proper version for our ESP32 C3) and completing the proper steps to configure our device. We can then use built in C functions (ex: tud_midi_write(...)) to handle our MIDI events.

Once our device is properly seen as a MIDI device, our firmware will have the following workflow:

1. Initialize USB stack
    a. Check if stack is configured: TinyUSB provides a callback function once the device is properly enumerated.
2. Have users set thresholds for their desired muscle group.
    a. Users set low and high points of muscle extensions. We expect our OpenEMG sensors to have a range of inputs from 200mV to about 2.6-2.7V, but it will still be up to the user to set how big or small of a range of muscle contraction they want to exhibit.
        i. Users will also have to set what kind of control they want to exhibit with the given muscle group, as this will change what kind of thresholding / triggering we will need to implement. For example, a note on signal

functions more as a switch, while a control change signal functions more as a slider.

3. Make sure proper connections are made to the computer and within the DAW.
    a. After plugging in, the DAW will recognize our system as a MIDI device. It is then up to the user to set what kind of parameters they want to use the system to change. This will need to be done for multiple MIDI channels if they want to use more than one muscle sensor to control more than one parameter.
4. Read EMG signals and compute if muscle extensions lie within the threshold range.
5. Convert raw voltage readings from EMG into digital signal through ADC.
6. Convert digital signal into an integer value and assemble into a MIDI message
    a. For example, a control change signal on channel 0, controller 7, with an integer value of 64 would be sending three bytes: 0xB0, 0x07, 0x40.
7. Repeat step 6 indefinitely.

Requirements

This subsystem must be able to both properly register the system as a USB device from the perspective of the computer and the DAW as well as send MIDI signals freely and without delay.

Tolerance Analysis

As laid out above, this subsystem will need to handle sensor input, properly convert it, and send MIDI signals over USB to the DAW.

2.3 Cost Analysis and Schedule

Labor:

Thus far, we estimate that each of us has spent approximately 4 hours a week since putting in the RFA working on the project for a total of 48 hours. We expect the weekly hours to increase to about double that once we receive our parts and begin our assembly and testing period. Assuming we stay on that pace for the rest of the semester, that brings the hourly total per person to 48 + 8 hours * 8 weeks = 112 hours for the semester.

The Grainger College states that the average starting salary for electrical engineering graduates is $88,321. Assuming 40 hour work weeks, this is an hourly rate of $43. Therefore, the total cost for labor for the three of us would be 3*$43*112 hours = $14,448.

Parts:

- LiPo Battery (1000mAh, 3.7V): ~$8.00
- MT3608 DC-DC Step-Up Boost Converter (3.7V to 5V): ~ $1.50
- AMS1117-3.3V LDO Voltage Regulator: ~$.50
- ESP32-C3 module: ~$4.00
- OpenEMG Muscle Sensor module: ~$4.80
- 3m Red Dot Monitoring Electrode - Model 2560: ~$.27 per count

Assuming that the user desires to use all open channels, it will cost roughly $2.00 in electrodes per use of the device (assuming the user does not attempt to reuse them). Therefore, total cost for

the entire product (without factoring in reusability for electrodes and ignoring USB cables) is $20.80.

Schedule:

Below are some deadlines that we would like to meet. They are listed by the start of each week; by this date, we expect to have the given deadline completed:

- 3/10: Fully completed PCB design, including final layout, battery module, and dimensions for both the housing and mounting holes.
- 3/17: Preliminary design of “watch” strap attachments to the housing
- 3/31: First assembly of first full “watch”, with housing, PCB, and electrodes attached.
    - Goal for this assembly is to have a completed working muscle sensor attachment with a working power supply.
- 4/7: Completed C firmware, with full MIDI/USB compliance.
    - Goal for this assembly is to be able to change at least one parameter with at least one muscle sensor attachment.
- 4/14 - 4/21: Testing period; iron out any errors, prepare for final demos.
    - Possible bonus: adding an OLED screen to make configuration of thresholds easier.

3 Ethics

Our team deeply values ethical behavior, and at every step of the way, we pause and ask ourselves, “Are we doing the right thing?” We regularly consult with both the IEEE and ACM code of ethics, but not only that, we strive to go beyond merely “complying” with them, and instead seek to live to the standard of a higher benchmark, holding the values of fairness and integrity deeply in our hearts.

As the IEEE code of ethics says, our team strives “to treat all persons fairly and with respect, and to not engage in discrimination based on characteristics such as race, religion, gender, disability, age, national origin, sexual orientation, gender identity, or gender expression. (IEEE Code of Ethics 2.7)” We take steps in order to ensure that we include as many people as we can, regardless of their physical ability. Although our device will be tied to our end user’s movements, it is designed to work anywhere on the body, allowing anyone who can move at least one muscle to use it.

Also taking guidance from the ACM code of ethics, we must ensure that we “respect the work required to produce new ideas, inventions, creative works, and computing artifacts (ACM Code of Ethics 1.5)” We value the creative spirit and seek to give credit where credit is due. Our project makes use of many open source projects published online, and we made sure to properly cite them and abide by their licenses. By respecting copyright law, we promote fairness for creators, innovators, and their ideas. Additionally, we used Google’s extensive patent database (patents.google.com) to verify that we are not infringing on anyone’s ideas.

Another essential but often overlooked part of ethics is to “respect privacy (ACM Code of Ethics 1.6)” Our device is designed from the ground up to respect the users privacy. By keeping everything on the device local, we avoid many of the privacy issues involved with cloud services and the Internet of Things. We also use the ESP NOW protocol, which has built-in AES-128 encryption, to ensure user privacy throughout the entire experience.

It is important to preserve user control over their own device. We will use encryption and verification to ensure that only authorized users can communicate with the MIDI receiver. We also take steps to ensure that no foreign signals will interfere with the device while we use it. In order to maintain a culture of transparency, we will also make our source code and process available upon request, to allow our users to make extensions and come up with novel ways to use our device. This preserves user ownership of their own device.

One safety concern is the fact that we are using potentially flammable batteries on our device. If the batteries are improperly handled, they could catch fire and/or explode. We will enclose our batteries in flame resistant materials to reduce the harm they can cause. We will also include a warning of the dangers of flammable batteries in the manual to our device. Our batteries will also be user serviceable, allowing the user to exchange defective batteries for new, safer, and functioning ones.

Our team cares deeply about the environment. Batteries tend to make their way into landfills, where their dangerous chemicals can leak into the ground and possibly a drinking water source. To ensure the proper disposal of these dangerous chemicals, we will include a manual with our

device, guiding the user to places where they can safely dispose of their batteries. We also made sure to use interchangeable and user serviceable parts to reduce e-waste, allowing our user to repair or send in their device for repair, instead of them throwing out the whole device when just one part is broken.

4 References

Charles Labs.

“Project – OpenEMG Arduino Sensor.” *Charles Labs*, n.d., https://charleslabs.fr/en/project-OpenEMG+Arduino+Sensor. Accessed 6 Mar. 2025.

Espressif Systems.

“ESP32-C3 Datasheet.” *Espressif Systems*, n.d., https://www.espressif.com/en/products/socs/esp32-c3. Accessed 6 Mar. 2025.

“ESP-NOW.” *ESP-IDF Programming Guide*, n.d., https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/network/esp_now.html. Accessed 6 Mar. 2025.

“ESP-IDF USB Device Documentation.” n.d., https://docs.espressif.com/projects/esp-idf/en/latest/esp32c3/api-reference/peripherals/usb_device.html. Accessed 6 Mar. 2025.

Ha Thach.

“TinyUSB: MIDI Test Example.” *GitHub*, n.d., https://github.com/hathach/tinyusb/tree/master/examples/device/midi_test. Accessed 6 Mar. 2025.

MIDI Manufacturers Association.

“MIDI 1.0 Detailed Specification.” *MIDI.org*, n.d., https://www.midi.org/specifications-old/item/the-midi-1-0-specification. Accessed 6 Mar. 2025.