MTP-25K Final Report

WAW MIDI Controller

Wes Marsen, Andrew Goertzen, Walker Bradley

ENGL 273

December 2025

Memorandum

To: Kimberly Lemieux, Mel Dundas, Justin Curran, Wayne Mayes
From: Wes Marsen, Andrew Goertzen, Walker Bradley
Date: December, 2025
Subject: WAW MIDI Controller – Final Report

We are happy to present you with our final report on the MTP-25K MIDI controller. This report will document the methods, outcomes, and challenges that we encountered during the project's creation.

We set an ambitious goal for our project at the outset. While we didn't get everything quite where we hoped it would be by the end, we have made massive strides to getting the project to completion. We're very proud of the work we've done.

All things considered, we're highly confident we'll have the MTP-25K working flawlessly with some more work, based on the successes of our testing and the operational state of our firmware. We are grateful for the support and guidance in carrying out our ambitions over these past 14 weeks.

Thank you!

Team WAW MIDI
WM, AG & WB

Executive Summary

This report documents our process of research and development for our custom MIDI keyboard instrument, the MTP-25K, over the semester of our Capstone project. Starting from only an initial concept, we designed the needed elements from the ground up. While not in a state of total completion, we have accomplished most of the major milestones in our design plan. After all the work we've put in, we've created a fun and flexible electronic instrument.

We overcame numerous challenges during our design process but did not end with a fully functional prototype built on our circuit board. This was due to repeated issues with total failure of our microcontrollers during the testing and assembly process. However, we did create a working breadboard prototype which allowed us to develop the core functionality of the device firmware, while attempting to resolve the hardware issues we experienced.

Our key accomplishments include:

- Designed successful capacitive touch surfaces for the keyboard and other controls into a custom printed circuit board.
- Created a plug-and-play interface for sending MIDI data to a computer or external hardware.
- Designed fast and effective firmware which met our goals for functionality.
- Designed and constructed a custom LED lighting matrix to indicate all settings to the user.
- Created a custom enclosure design to house the device in a robust and portable package.

We fully intend to continue development of the MTP-25K beyond the scope of this semester, and have identified several opportunities for improvement. We believe this project demonstrates innovative use of modern electronics technology to meet the needs of a significant niche of users, where very few open-source and customizable solutions are available.

Table of Contents

Table of Figures

Introduction

The WAW MIDI team was brought together by a shared interest. Wes has been an electronic musician since his teenage years, Walker has spent years playing a variety of musical instruments, and Andrew is a music lover who grew up with musicians. With a shared love for both music and electronics, we agreed on creating a MIDI instrument for our Electronics & Computer Engineering Technology capstone project. We set out to design a two-octave keyboard instrument with 25 keys and multiple modulation controls, all using capacitive touch technology and based around the Teensy 4.1 microcontroller.

Background

Musical Instrument Digital Interface (MIDI) is an industry standard digital music expression language [1]. The protocol conveys a musician's physical input on a MIDI controller as a series of 1s and 0s that computers and other electronic instruments understand. Controllers come in all shapes and sizes, from electronic saxophones (Figure 1) to stage lighting controllers (Figure 2). The most common form borrows from the piano (Figure 3), with keys ascending in pitch from left to right along the Western chromatic scale. Many add additional user inputs for additional control: knobs, buttons, sliders, etc.

Figure 1: Akai EWI 5000 [2]

Figure 2: Elation MIDICON-2 [3]

Figure 3: Akai MPK MINI MKIII [4]

Musicians often use MIDI controllers in tandem with Digital Audio Workstations (DAW), computer programs that emulate recording studios. Musicians can modify the sound profile of the notes they play on their controller with a wide variety of digital modulators. Most sound generators, like synthesizers and drum machines, also understand the MIDI language.

Keyboard MIDI controllers predominantly employ mechanical keybeds, ranging in quality based on price point. Low-quality keybeds in the $100-200 dollar range (Figure 4) fall short of simulating a piano's keys and are susceptible to damage, often rendering controllers with perfectly functioning electronics unusable. High-quality keybeds are more faithful to the acoustic experience and offer more reliability; however, they can cost several thousand dollars (Figure 5), creating a substantial barrier to entry for beginners and hobbyists.

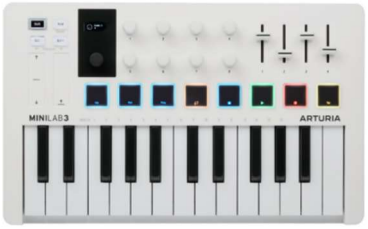

Figure 4: Arturia MiniLab 3 ($159.99) [5]

Figure 5: Native Instruments Kontrol S88 MK3 ($1699.00) [6]

Purpose & Scope

We wanted to do something different; start with this familiar layout, which musicians could instantly understand, but ditch the idea of making it appear like a traditional piano keyboard. Instead, we aimed to design a practical, robust, and expressive instrument that we could build with simple electronic components and minimal mechanical elements. We chose to design our controls around capacitive touch surfaces to create a fluid and responsive instrument with as few moving parts as possible.

The prime design considerations for the device were:

- It should feature a responsive and comfortable 25 key (two-octave) capacitive-touch keyboard.
- It should only require a USB connection for power.
- It should pass MIDI data through both its USB connection and a physical MIDI TRS jack.
- It should clearly indicate relevant control settings to the user.
- It should support velocity sensitive note input.
- It should include multiple capacitive-touch modulation surfaces videlicet sliders.

As the project continued, our vision grew more ambitious. We recognized a gap in the market for open-source firmware on MIDI controllers and designed our firmware with user modification in mind. We extended this design philosophy to our enclosure, creating 3D CAD files that users could print and assemble at home. Our goal was an instrument that

felt like a hobbyist playground, rather than the proprietary, or “walled garden”, approaches on the market.

Hardware

We spent a good deal of time deciding on the hardware we needed to realise our vision. Our first consideration was the “brain” of the controller: the microcontroller.

Teensy 4.1 Microcontroller

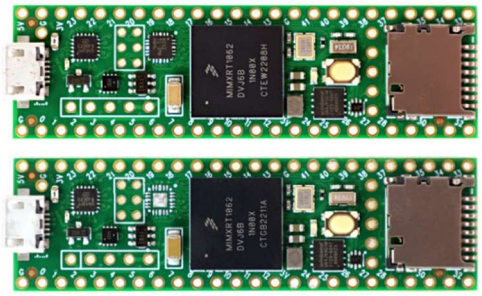

Figure 6: Teensy 4.1 Microcontroller [7]

Our key considerations in choosing our main processor for our project were speed, power usage, and General Purpose Input/Output pins (GPIO). We settled on the Teensy 4.1 (Figure 6), an impressive microcontroller powered by the NXP RT1062 ARM Cortex-M7 and designed by PJRC. This suited our needs well, supporting: a 600 MHz maximum clock speed for fast response times; a sub-100 mA current draw for efficient power use; and 55 GPIO pins including Analog to Digital Converters (ADCs), interrupt compatibility for even faster response, and three Inter-Integrated Circuit (I2C) channels for communication with sub-processors [7].

Additionally, the Teensy 4.1 is compatible with the Arduino Integrated Development Environment (IDE), a beginner and hobbyist friendly interface for programming electronic devices. This lent well to our design philosophy and gave us access to useful Arduino libraries, prebuilt code bases with useful functions, including MIDI libraries that do the heavy lifting in generating MIDI signals.

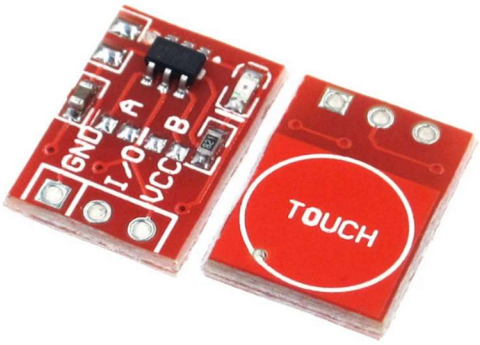

Figure 7: TTP223 Touch Sensor Modules [8]

We purchased our first Teensy 4.1 processor over the summer, which worked perfectly with our breadboard prototype. We did encounter problems when integrating it with our final PCB, which we cover in more detail in our Final PCB section.

TTP223 Touch Sensor Modules

We purchased 30 TTP223 touch sensor modules (Figure 7) with our first Teensy 4.1 board. These breadboard-ready touch sensing circuits were perfect for initial proof-of-concept work. With our microprocessor and touch pads, we started building our first breadboard.

Early Breadboard Prototype

We hit the ground running at the beginning of the semester by building our first breadboard test bed (Figure 8). We used this to test basic functionality of the touch sensors and send our first MIDI signals. The Arduino libraries made the process simple, allowing us to add complexity to our firmware. We discuss this further in our Breadboard Firmware section.

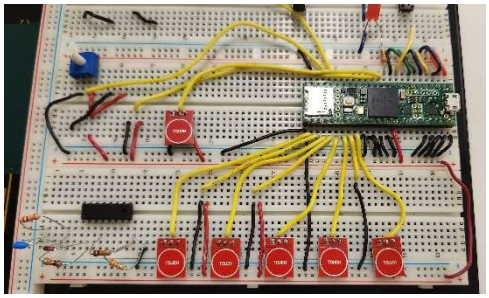

Figure 8: Early Breadboard Prototype

While we had good success with sending MIDI data, we were underwhelmed by the TTP223 touch sensors. The sensors were relatively slow to respond, timed out after six seconds, and were prone to misfires. We decided to design our own touch sensors and build them directly into our PCB. We discuss this further in our Custom Touch Sensor section.

With our initial findings on our test breadboard, we moved on to selecting components to power our capacitive-touch inputs.

Figure 9: PCB Keyboard Touch Electrodes

Custom Touch Sensors

We chose to use AT42QT1010 touch sensors for our capacitive-touch keys. These chips generate electric fields around electrodes (Figure 9), flat copper plates sitting just below the surface of our first PCB. When a conductive material, like a person's finger, gets close to the electrode, the sensor detects a disturbance and sends a signal. We included three electrode styles on the PCB to narrow down to our final design.

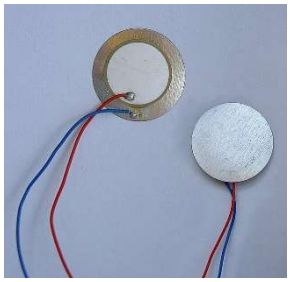

Figure 10: Piezoelectric Sensor Disc [9]

Piezo Electric Sensors

We knew early in the project that we wanted to include velocity data in our MIDI signals but took some time in deciding how we

would achieve it. We opted for piezo sensors (Figure 10) at the suggestion of Fil from Capital City Transistor & Valve (CCTV), a local electronic repair shop that specializes in audio equipment.

Figure 11: Azoteq IQS7211A Sensors [10]

Custom Modulation Surfaces

We chose to integrate capacitive-touch sliders and a trackpad directly into our PCB with Azoteq IQS7211A sensors (Figure 11). These sophisticated Integrated Circuits (ICs) connect to rows and columns of electrodes and track movement across them. They communicate this movement through the Inter-Integrated Circuit (I2C) protocol. I2C is a short-range communication standard available on many ICs, see Appendix A for more information. We included small sliders and a trackpad to test on our first PCB.

Test PCB

We ordered our first PCB (Figure 12) in week five and received it in week seven. We found that the hatched electrode style did not offer the sensitivity we wanted and decided to split the size difference between the two solid style keys for our final design.

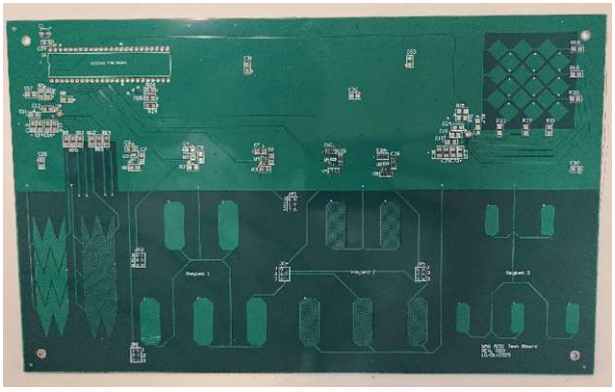
Figure 12: Test PCB

We attached the piezo sensors to the bottom of our PCB and used an oscilloscope to test their reaction to input. We got good results, seeing oscillating waveforms at different peak voltages, but knew we would have trouble getting the highest value consistently in firmware. We opted to build peak detecting circuits to measure and hold the maximum voltage reached (Figure 13). See Appendix B for our piezo circuit's schematic diagram.

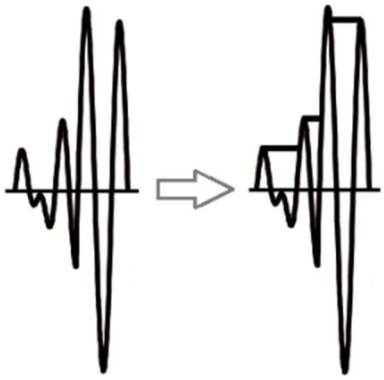
Figure 13: Peak Detection [11]

We tested the trackpad using an IQS7211A evaluation kit and a CT210A programmer. The programmer allowed us to configure rows and columns and adjust sensitivity. Once we tuned it to our liking, the programmer generated a

configuration file for our Arduino firmware, which we discuss in more detail in our Modulator Firmware section.

After finalizing our capacitive-touch inputs, we started building up our breadboard to test the full scope of our controller.

Helper Boards

As we built up our breadboard, we quickly realized that the Teensy 4.1's 55 GPIO pins were not sufficient to control our many inputs and outputs. We decided that we needed additional sub-processors, or helper boards.

MCP23017 GPIO Expander

To monitor the keys, we chose to use two Adafruit MCP23017 GPIO expanders (Figure 14). These each support up to 16 inputs or outputs; perfect to monitor our 25 keys. These ICs also communicate via I2C.

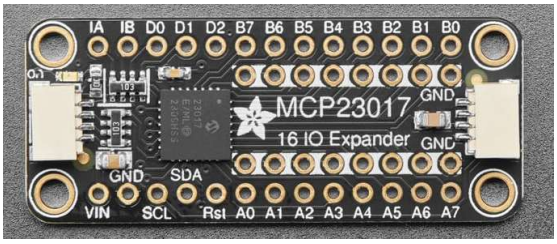

Figure 14: MCP23017 GPIO Expander [12]

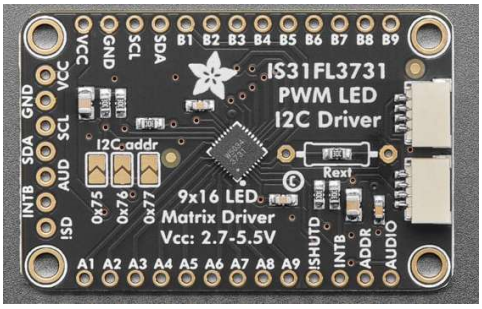

Figure 15: IS31FL3731 Charlieplexing Driver [13]

IS31FL3731 Charlieplexing Driver

We mentioned earlier that we wanted to clearly communicate relevant control settings to our user. We settled on using LED indicators for settings, active notes, and active modes, totalling to 81 LEDs. To control these indicators, we chose the Adafruit IS31FL3731 charlieplexing driver (Figure 15). This IC communicates over I2C and can drive up to 144 LEDs. For more information on charlieplexing, see Appendix C.

Our Charlieplexing Array

In week eleven, we started designing our charlieplexing array. We used TinkerCAD, a free drafting utility, to design our charlieplexing schematic (see Appendix B). Once we confirmed the functionality of our design, we wired our 81 LEDs together and inserted them into the top of our enclosure. We built a small LED Finder program to test the LEDs, which we discuss in more detail in our LED Finder section.

We ordered the helper boards in week seven, and they arrived in week eight.

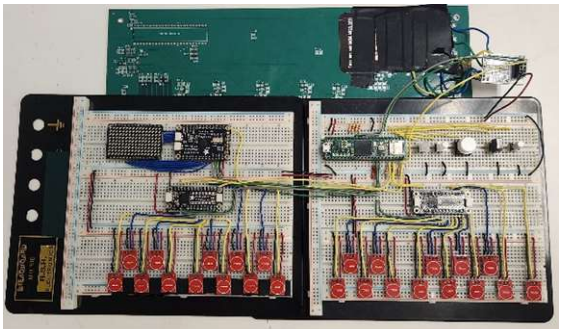

Figure 16: Full Breadboard Prototype

Full Breadboard Prototype

With our helper boards in hand, we finished building our full breadboard prototype (Figure 16). We included 25 TTP223 sensors for keys, an LED matrix to simulate our LED indicators, five push buttons to control modes and features, and a trackpad from our test PCB. Other than the TTP223 problems we mentioned earlier, we had excellent success at this stage.

Mechanical Push Buttons

When we visited Fil at CCTV, he donated a handful of mechanical inputs. This included push buttons, which we used to control our modes and features on the breadboard. For our final PCB, however, we decided to purchase lower profile buttons to better fit our low-profile form factor.

TRS Jack

We also chose to include a physical MIDI Out port on our final PCB, allowing the user to send MIDI data to an external hardware device over a MIDI cable. The traditional connector for this is the 5-pin DIN jack; however, this is a larger connector than we wanted to use. We chose to use a 3.5mm TRS jack instead, as is common with low-profile controllers (Figure 17). Our device conforms with the MIDI Association recommended TRS-A wiring scheme, enabling it to connect to hardware instruments like synthesizers and drum machines directly to send MIDI data.

Figure 17: MIDI Adapter Cable with DIN (Left) and TRS (Right) Connectors [14]

After selecting and testing our components, we got to work designing our final PCB.

Final PCB

We encountered some challenges in designing our final PCB (Figure 18) due to the number of components included on the board. We opted to increase the number of PCB layers from four to six and build our LED indicator array separately. We ordered the PCB in week nine. See the schematic diagram in Appendix B.

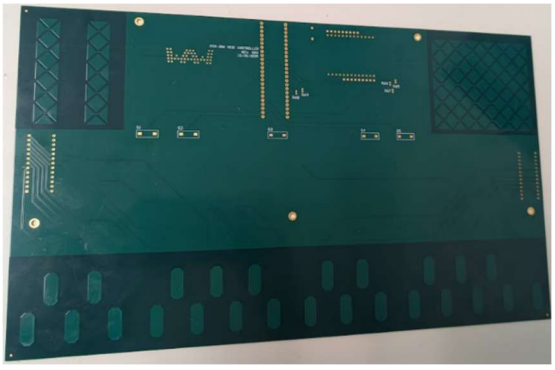

Figure 18: Final PCB

We assembled our final PCB in week eleven and, due to its size, manually populated the components. Unfortunately, we warped the board in this process, encountering communication issues. We populated a second PCB more carefully but warped it as well. The communication issues persisted on the second board, and additional power issues caused damage to our Teensy board. In the process of troubleshooting, we lost six Teensy boards to power surges and short circuits. We determined that the piezo circuits were the likely culprits, as we had not populated them on the first board.

We had some success after removing the piezo circuits from our second board. Our keys responded well, with great sensitivity and speed, but often lost communication. We suspect that power fluctuations were causing the GPIO expanders to lose connection. We also experienced inconsistencies with our IQS7211A sensors and push buttons.

While we are confident that we can fix these problems by redesigning the PCB and piezo circuits, we did not have enough time to complete these fixes before Symposium.

Enclosure

With our final PCB in hand, we got to work on designing a compact, durable, 3D printed enclosure to house it. Due to the size of our PCB, we needed to use the Electronics Department's Ultimaker S7 3D printer. The S7 offers a 330mm print bed, 2mm larger than our board. Despite the adequate size, we were not happy with the quality of the print. We decided to reach out to Camosun Innovates for advice. They offered to print our enclosure on one of their larger Elegoo printers and we were much happier with their results.

The final design has two pieces to the hinged front panel, one part flat on the keybed and the other raised 5mm from the PCB surface to make space for the lighting. The two pieces

attach with dovetails. We separated these pieces so that they could be printed top side down for a better surface finish. On the keybed portion, there are two clasps that close the top, at the front towards the user opposite the hinge. The bottom section has a lip around the perimeter, so the PCB sits inside it. There are standoffs to further lock it in place. The side panels are glued onto the bottom. We also 3D printed button caps which sit inside the case. See Appendix E for a full rendering of the enclosure.

Hardware Conclusion

While we ran into problems with our final PCB, we are happy with the components we chose. We had great success in our breadboard prototype, sending note and modulation data exactly as intended. We are confident that, with some redesigns, we can fully realize our vision. We discuss these redesigns in more detail in our Future Plans section.

Now that we know what physical components make up the controller, let's talk about the firmware that drives them!

Firmware

In the early stages of our project, we focused on firmware to test individual components. We built small Arduino sketches, or programs, that converted physical input on keys and buttons on a breadboard into MIDI signals. As previously mentioned, the Teensy 4.1 is equipped with a built-in MIDI library, a code base that handles the more complicated aspects of MIDI signal generation.

Early Breadboard Firmware

With the MIDI commands under control, we started thinking about the features and play modes we wanted to implement. We settled on expanding our 25 key range with octave controls, emulating the sustain pedal of a piano with a HOLD mode, and creating a simple arpeggiator with an ARP mode. The two modes, and their sub-modes, exhibit different key behaviours, outlined fully in Appendix D.

Test PCB Firmware

We started testing our modulation surfaces when our test PCB arrived in week 7. We connected an IQS7211A evaluation board to our trackpad and used a CT210A programmer to fine-tune the response. After some physical modifications to the board to improve the response, we generated a configuration file and added it to our firmware. Using the

IQS7211A's Arduino library, we successfully converted movement on the trackpad to pitch bend signals. This was our first venture into Inter-Integrated Circuit (I2C) communication, a short-range serial protocol allowing Integrated Circuits (ICs) to talk to each other on a circuit board. See Appendix A for a detailed description of I2C.

Full Breadboard Firmware

Our helper boards (MCP23017 GPIO expanders and IS31FL3731 charlieplexing driver) arrived in week 8, and we quickly built up the rest of our breadboard. We modified our early breadboard firmware to include the full 25-key range, five push buttons (Octave Down, Octave Up, SHIFT, HOLD, and ARP), an LED matrix, and the trackpad from one of our test boards.

GPIO Expanders Integration

Integrating the GPIO expanders went seamlessly due to their Arduino library. We initially used the MCP23017s' interrupt feature, which notified the Teensy 4.1 immediately if the state of a key changed. This gave us incredibly fast response times, allowing us to reduce the Teensy's clock speed and conserve power.

Charlieplexing Driver Integration

To test our charlieplexing driver, we connected it to a compatible LED matrix designed by Adafruit. The IS31FL3731's Arduino library did the bulk of the work, adjusting each pin to light the correct LED based on a specified XY coordinate. We used these LEDs to indicate user settings (mode selection, tempo, and customization options), and active keys.

Initially, we reset the LED indicators for the keys when changing octaves; however, we found that this conflicted with the HOLD mode. As outlined in Appendix D, HOLD mode allows users to toggle notes on and off by repeated touches, and we wanted to help the user keep track of these active notes. To improve the user experience, we associated the LEDs and notes through arrays, letting us shift the indicators as the user changes octaves.

LED Finder

Once we felt confident in controlling the Adafruit LED matrix, we designed our charlieplexing array to fit into our enclosure. After wiring the LEDs together, we built a basic program to test each XY coordinate and assign labels to each individual LED.

With all our features integrated on the breadboard, we were excited to get everything up and running on our final PCB, which we ordered in week 9.

Final PCB Firmware

As previously discussed, our final PCB had some unexpected hardware issues. The main problem on the firmware side was spotty I2C communication. The helper ICs would regularly stop responding, resulting in the Teensy sending note on commands for every note. We made some hardware modifications that improved communication; however, we could not fully mitigate the problem. We were happy with the sensitivity and responsiveness of the keys when the communication worked; however, with frequent communication glitches, we determined that we would not have the board demo-ready in time for symposium.

While we could not get our PCB demo-ready for users to play, we did have good success with modifying our charlieplexing code to drive our own LED array, pressure-fit into our enclosure. To demonstrate this, we used an Arduino Uno to drive our LED array in a preset sequence. The Arduino Uno does not support MIDI, so we could not generate MIDI signals, but we believe this was an acceptable compromise given our challenges.

Firmware Conclusion

We had great success in developing the firmware to power the MTP-25K. All of our features and modes are working as we intended, and our preliminary test code for the modulation surfaces and piezo sensors was promising. We could not implement these features on our final PCB due to hardware problems; however, we are confident that the code will work well to control the hardware after a redesign.

Finances

We began with a self funded budget of $500 with the expectation that we would end up under that for our first prototype. We did also plan to build multiple units once we had the design down, so knew we might spend above our starting budget in the long run. We ended up going over our budget due to unforeseen problems. One additional cost was the amount of Teensy boards we needed to purchase. We had lost four by the end of week 13 and needed more to continue developing our unit. After all the expenses we incurred, we spent $863.56 this semester.

To make one of our MTP-25Ks it would cost approximately $125. This could be lowered by purchasing components and PCBs in bulk and making changes to some of the

components we used, particularly replacing the development boards with discrete components.

Moving Forward

Despite the challenges we encountered, we are eager to continue working towards completing our vision. We have four primary redesign plans, outlined below.

PCB Design

We came to find during development that the decision to put our whole device together on one PCB became unwieldy and problematic. We would like to separate the different elements of the device such as the keyboard, modulation surfaces, lighting, and the core board with the MCU, connectors and other peripherals into different PCB modules within the enclosure. This would allow for easier assembly and repair should any part become damaged later. This would also allow us to use different power and ground plane stack-ups and layer counts on the different boards, instead of having one large six-layer PCB, and hopefully improve the flexibility of the keyboard to give the piezo sensors increased sensitivity. Additionally, this would allow for a modular approach to creating further iterations on the design, such as breaking the keyboard into 1-octave modules, and then being able to add another octave or two to achieve a larger MIDI controller of the same design.

Piezo Sensor Circuits

Velocity sensitivity is a key part of our design, and we have every intention of getting it fully functional in the final version of the instrument. The piezo sensor & peak detector circuits did function well in our testing prior to implementing them on the PCB, so beyond the time constraints of the 14-week semester we see no reason we can't work out the hitches in our design.

Modulation Surfaces

We did not get all the modulation surfaces completely functional as we wanted them during our semester. However, we made good progress on getting a working trackpad functional, and with more time should be able to have all the sliders working well also. However, we did have to make quick decisions when choosing chips to control our capacitive touch elements as there were many choices, and it was a bit hard to narrow down which would give us the best results without the opportunity to try any of them. In the

future, we would like to look at other possible options and determine which will give us the best combination of cost, performance, and ease of design to accomplish the touch surfaces we would like to have on the finished device.

LED Indicators

The LED indicators being soldered with stranded wire sitting bundled in the enclosure was functional enough for a prototype, but with more development time we would like to add in a PCB to contain the charlieplexing matrix wiring for all our indicator LEDs, which would be surface mounted to the board just under the front panel. This would make the interior much cleaner and more resilient than our current solution.

Future Plans

We intend to make the finalized version of this instrument available as a DIY kit for electronics hobbyists and electrically inclined musicians, allowing them to build and develop a personal relationship with their instrument. This would also allow for customization, such as allowing the user to choose what size of keyboard they want. We would provide the 3D printing files for our version of the enclosure for those who wish to fabricate it themselves, or the design specifications if they wanted to make an enclosure from other materials like wood or sheet metal. As the firmware is open source with an explanatory tutorial on our GitHub, users will be able to crack open the code of the device and add to it themselves or simply use it as a learning tool to discover more about the functioning of modern music technology and electronics.

Conclusion

Over the course of this semester, we set out to create a working MIDI controller and by the end we did just that. Our goal was 25 touch keys which we delivered, four sliders which we had to scale down to 2 due to the IQS7211 chips, and feature buttons for arpeggio mode, hold mode and octave controls. We had a lot of successes and lots of problems along the way. We started off not knowing how to make a touch sensor pad and we had to do a lot of research to figure out the right values for the components of the touch pad circuit. We only had issues with the sliders once we tried to get them working on the final PCB. We had a big scare during week 14 when the final PCB fried the only Teensy boards we had at the time. There were a lot of challenges with this project, but we managed to make it in time for the symposium. Looking forward we plan to further develop the MTP-25K and make it more hobbyist focused with modules and customizability.

References

[1] MIDI Association, "About MIDI - Part 3: MIDI Messages," 2025. [Online]. Available: https://midi.org/about-midi-part-3midi-messages. [Accessed 11 12 2025].

[2] Akai, "EWI 5000," 2025. [Online]. Available: https://www.akaipro.com/ewi5000.html. [Accessed 11 12 2025].

[3] Elation, "MIDICON 2," [Online]. Available: https://www.elationlighting.eu/midicon-2. [Accessed 11 12 2025].

[4] Akai, "MPK MINI MK3," [Online]. Available: https://www.akaipro.com/mpk-mini-mk3. [Accessed 11 12 2025].

[5] Long & McQuade, "Arturia Minilab 3 25-Key MIDI Controller w/Software - White," [Online]. Available: https://www.long-mcquade.com/303386/Keyboards/MIDI-Controllers-Interfaces/Arturia/MiniLab-3-25-Key-MIDI-Controller-w-Software-White.htm. [Accessed 11 12 2025].

[6] Long & McQuade, "Native Instruments Kontrol S88 MK3 88-Note Keyboard Controller," [Online]. Available: https://www.long-mcquade.com/347491/Keyboards/MIDI-Controllers-Interfaces/Native-Instruments/Kontrol-S88-MK3-88-Note-Keyboard-Controller.htm. [Accessed 11 12 2025].

[7] PJRC, "Teensy 4.1 Development Board," [Online]. Available: https://www.pjrc.com/store/teensy41.html. [Accessed 11 12 2025].

[8] ROBU.IN, "TTP223 Touch Key Module - 2Pcs," [Online]. Available: https://robu.in/product/ttp223-touch-key-module-2pcs/. [Accessed 11 12 2025].

[9] S. Reipl, "File:Piezo.jpg," 25 7 2007. [Online]. Available: https://commons.wikimedia.org/wiki/File:Piezo.jpg. [Accessed 11 12 2025].

[10] Azoteq, "IQS7211A," [Online]. Available: https://www.azoteq.com/product/iqs7211a/. [Accessed 11 12 2025].

[11] P. Khatri, "Peak Detector Circuit," Circuit Digest, 6 12 2018. [Online]. Available: https://circuitdigest.com/electronic-circuits/peak-detector-circuit-using-op-amp-lm741. [Accessed 11 12 2025].

[12] L. Clark, "Adafruit MCP23017 I2C GPIO Expander," Adafruit, 23 3 2022. [Online]. Available: https://learn.adafruit.com/adafruit-mcp23017-i2c-gpio-expander. [Accessed 11 12 2025].

[13] Adafruit, "Adafruit 16x9 Charlieplexed PWM LED Matrix Driver- IS31FL371 - Stemma QT / Qwiic," [Online]. Available: https://www.adafruit.com/product/2946. [Accessed 11 12 2025].

[14] Boss, "BMIDI-1-35," [Online]. Available: https://www.boss.info/ca/products/bmidi-1-35/media/. [Accessed 11 12 2025].

APPENDIX A: INTER-INTEGRATED CIRCUITS (I2C)

Introduction

Inter-Integrated Circuit (I2C) is a short-range serial communication protocol that connects a main processing unit (MPU), or controller, to multiple peripheral devices, or targets, over a two wire bus. I2C was first introduced by Philips Semiconductors (now NXP Semiconductors) in 1982 and was considered an industry standard by 1998 [1]. This standard is critical for our project, the WAW MTP-25K, in allowing us to use peripheral ICs to track and control our many inputs and outputs. We are using five I2C compliant ICs to communicate with most of our user inputs and LED indicators, spread across three I2C buses on our MPU.

Standard

The I2C bus is made up of two lines, a clock signal (SCL) and a data signal (SDA) (Figure 1). These use an open drain, holding the line high when not in use. To start communication, the controller pulls the SDA line low, followed by the SCL line. To stop communication, SCL is released high, followed by SDA. I2C compliant ICs are designed to easily identify these messages, whereas other devices need to sample at twice the clock frequency to detect them [2].

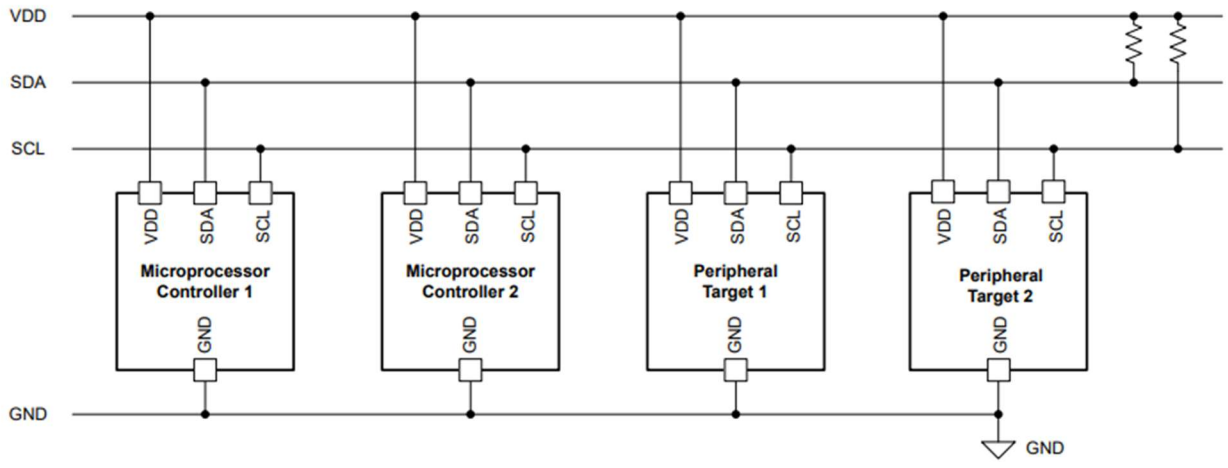

Figure 19: Typical I2C Integration [1]

After the start message, communication is broken up into frames (Figure 2). The first frame contains the target's seven-bit address and a read/write bit. This is followed by one or more data frames, single bytes of serial data. Each frame is concluded with an acknowledge bit

to signify that the target has received the message. Each IC on a bus needs a unique address. Of the 128 available addresses, eight are reserved for other purposes [1].

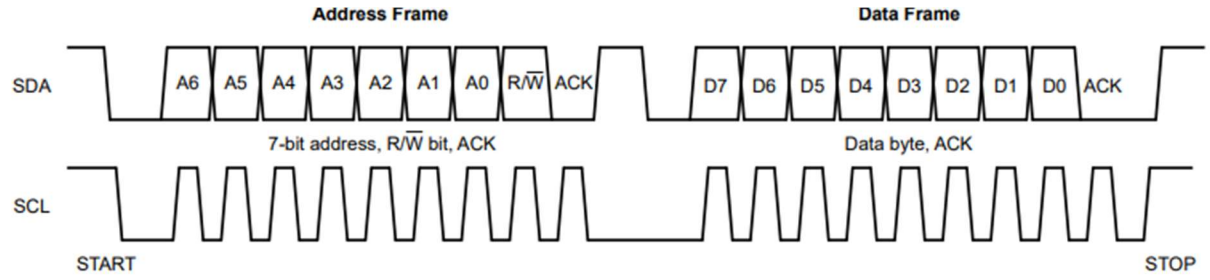

Figure 20: I2C Address and Data Frames [1]

I2C currently supports five clock modes. Standard-mode (100kbps), Fast-mode (400kbps), and Fast-mode Plus (1Mbps) are the most widely supported. High-speed mode allows for data rates up to 3.4Mbps but often requires active pull-ups and an additional command from the controller. Ultra-fast mode goes up to 5Mbps but is write-only [2].

Conclusion

I2C is an instrumental tool in allowing us to manage the MTP-25K's many inputs and outputs. It allows us to conserve GPIO pins on our MPU, using only two pins for each bus. While we encountered hardware problems that restricted our use of I2C on our final PCB, we believe it will be more effective after some redesign work.

APPENDIX B: SCHEMATIC DIAGRAMS

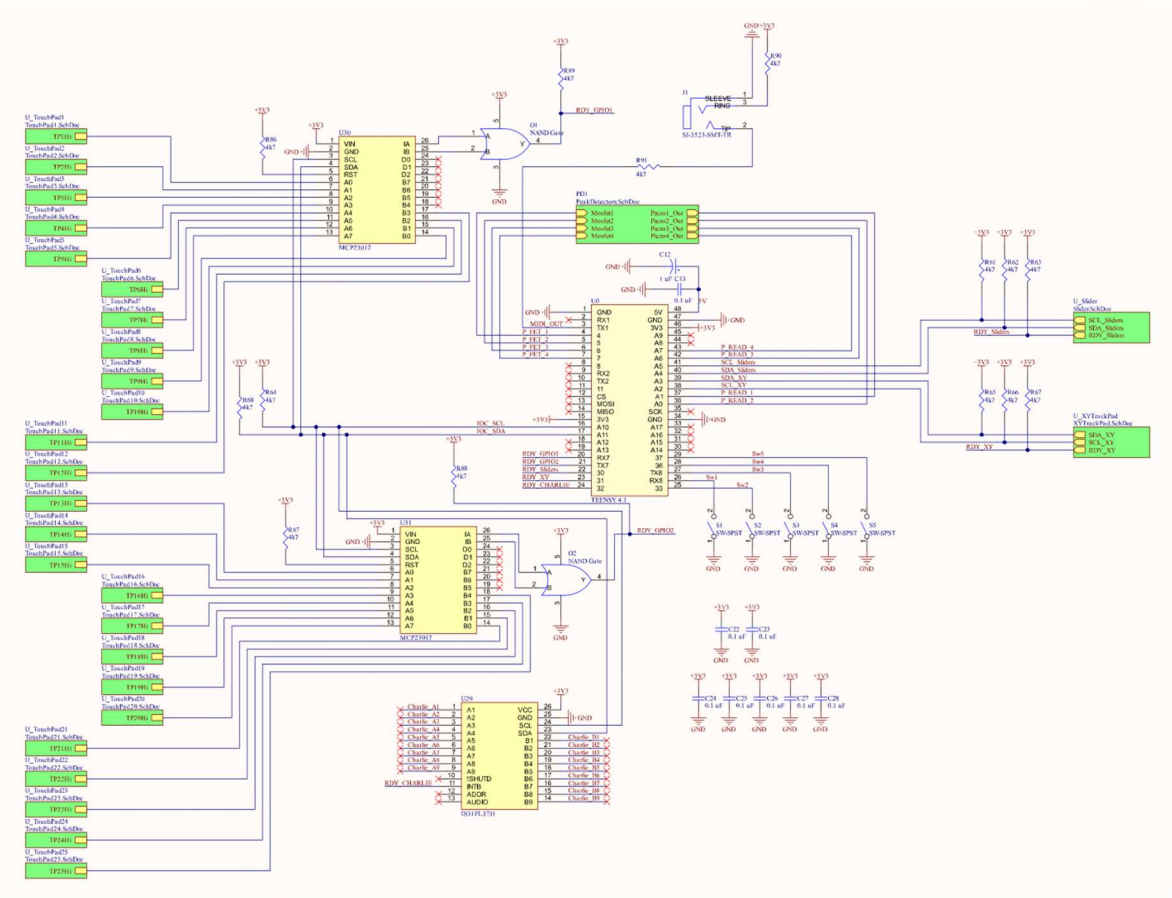

Figure 21: Top Level PCB Schematic

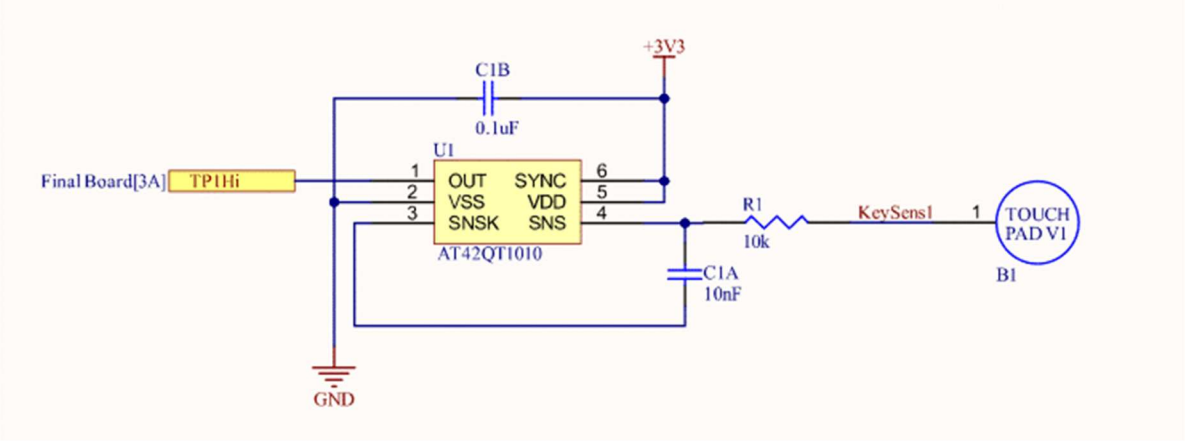

Figure 22: Touch Sensor Schematic

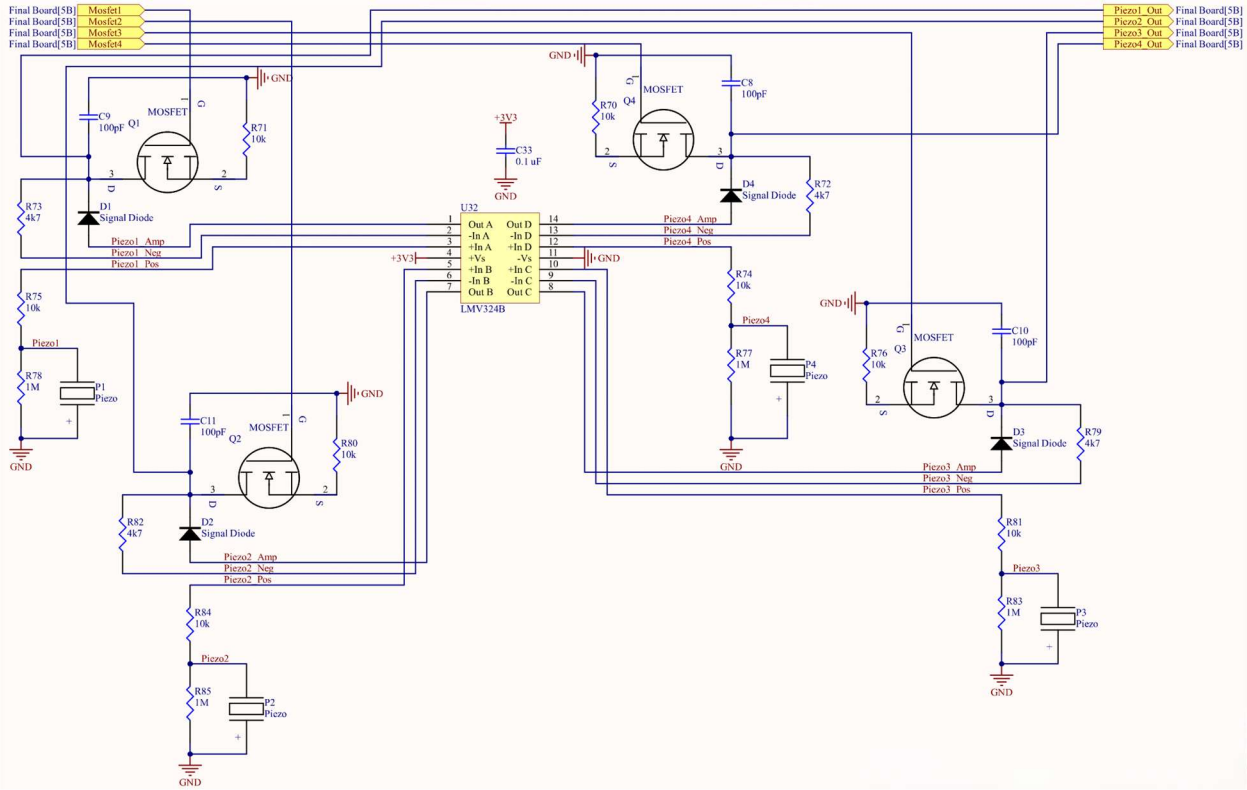

Figure 23: Piezo Peak Detection Circuit

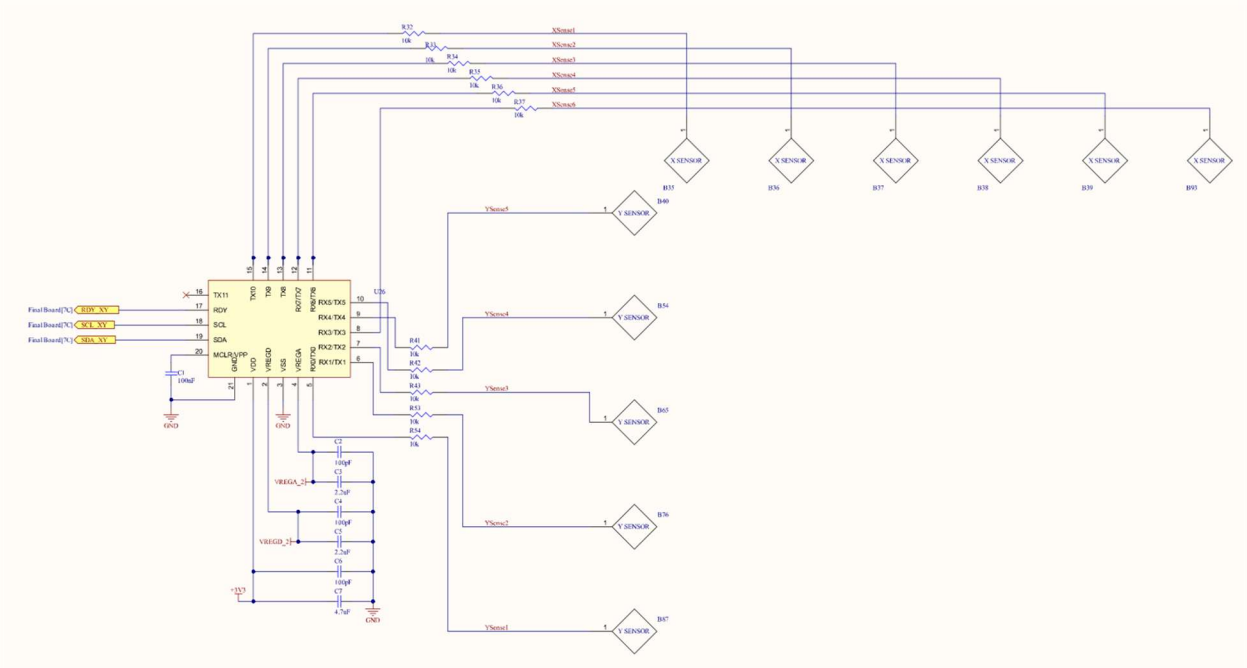

Figure 24: Trackpad Schematic

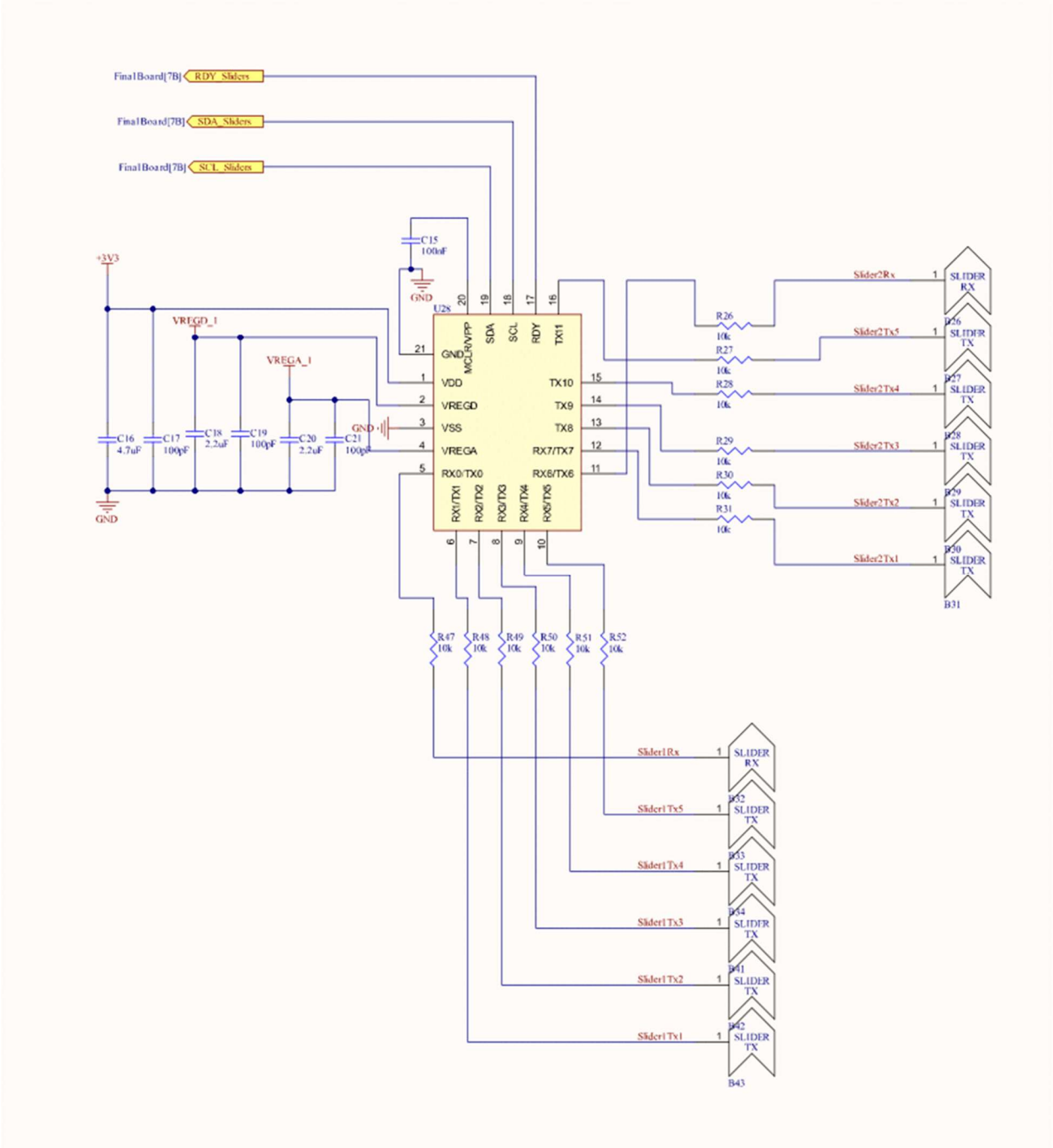

Figure 25: Slider Schematic

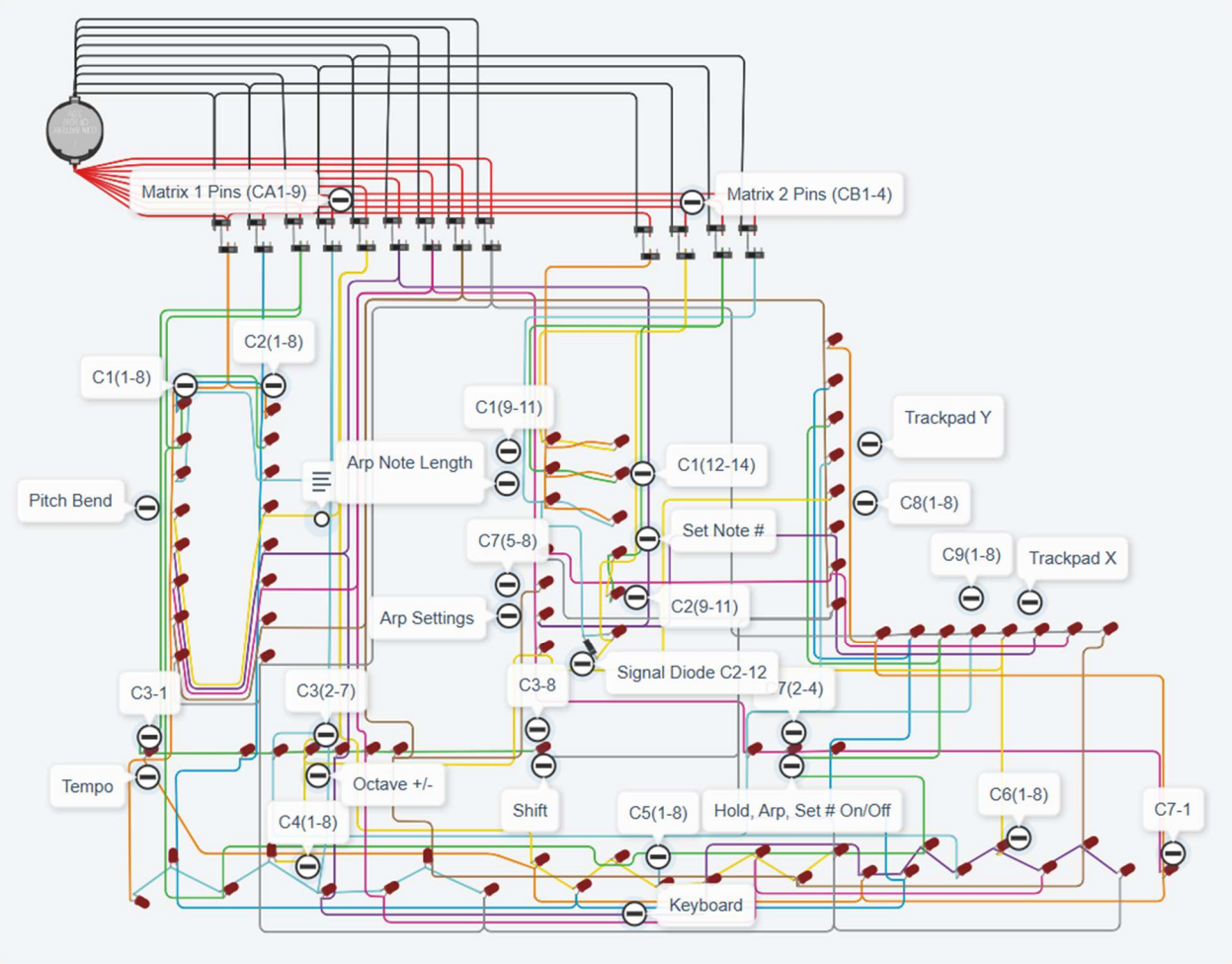

Figure 26: Charlieplexing Array Schematic

APPENDIX C: CHARLIEPLEXING

We used 81 LED indicators across the front panel which display the status and settings of the device. To control these, we chose the Adafruit IS31FL3731 Charlieplexing PWM LED Driver. The advantage of this device is that it can individually control each LED at any location in a charlieplexed matrix of up to 144 LEDs. We had not used charlieplexing before this project and had to spend a few days working out the matrix of wiring before we could solder it together by hand.

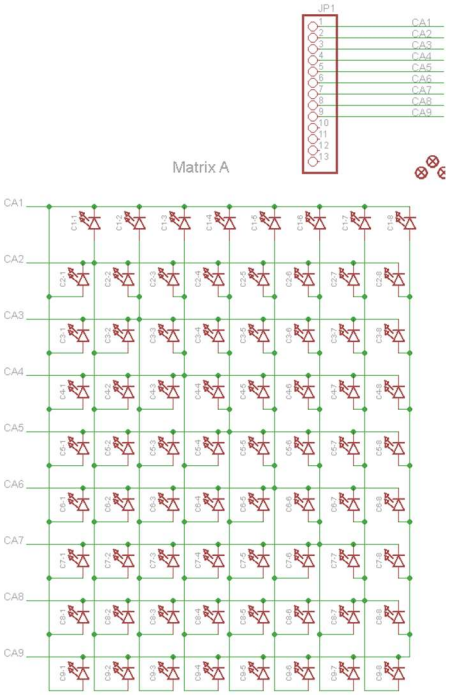

Figure 27: First LED Matrix

Charlieplexing is a form of tri-state multiplexing which takes advantage of the polarized nature of LEDs. Any light in the matrix can be switched on by turning one pin high, another pin to ground, and the rest to high impedance.

There are two matrix drivers on the IS31FL3731, made up of 9 pins, each controlling up to 72 LEDs. Since we only have 81 LEDs, we used all of one matrix and only four pins of the other. We applied the wiring scheme (seen in figure 9) to the large matrix, and for the second half we used a much simpler matrix, (seen in figure 10).

To track how this would work in practice, we designed the matrix in TinkerCAD. This allowed us to simulate not only a schematic design, but the actual placement of wiring for our physical enclosure. After working the logic of the wiring out, we spent a couple days soldering it all together. In the end, it all worked as intended, largely thanks to the upfront work of laying out the matrix in a virtual representation before attempting it physically.

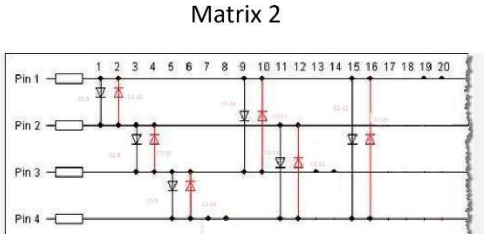

Figure 28: Second LED Matrix

APPENDIX D: PLAY MODES AND FEATURES

Our controller is equipped with octave controls, adjustable ARP modes, and adjustable HOLD modes. Users can customize the modes further with SET, ARP Note Length, and ARP Pattern (labeled ARP MODE). The SHIFT button provides access to the customizations.

Octave Controls

Our octave down and up buttons adjust the MTP-25K's note range. At its lowest setting, the notes extend from C1 to C3. At its highest setting, the notes extend from C6 to C8. This gives the user access 85 total notes, three short of a full sized piano.

ARP

The ARP button turns on the arpeggiator, which takes chords of multiple notes and plays them in a sequence. The firmware removes the notes from the arpeggio when the user releases the keys and resorts the remaining notes. The arpeggio properties can be altered with:

- SET #: This setting adjusts the number of notes that can be included in the arpeggio. The current options are 1, 2, 3, 4, 6, and 8.
- ARP Note Length: This setting adjusts the length of the notes in the arpeggio relative to the TEMPO indicator. The current options are quarter note, eighth note, and sixteenth note.
- ARP Mode: This setting adjusts the arpeggio's pattern. The current options are ascend, descend, ascend-descend, and custom.

HOLD

The hold button enables and disables the HOLD feature. When enabled, notes will continue to play after the user releases the keys, emulating a piano's sustain pedal. In our application, the user can toggle a note back off by touching its key again, relative to the octave range selected. Our firmware tracks these active notes, illuminates indicator LEDs above the key to help users can keep track of sustained notes.

There are two sub-modes for the HOLD feature: HOLD_ALL and HOLD_SET. The former sustains as many notes as the user presses, up to the full 85 note range of the controller. The latter limits the number of notes sustained to the user's SET selection, allowing the user to play notes freely above a latched chord. In either sub-mode, disabling HOLD mode will turn off all latched notes.

ARP_HOLD

ARP can be used in conjunction with HOLD to enter ARP_HOLD mode, which latches notes into an arpeggio sequence. Similar to HOLD mode, the user can unlatch the notes by touching the relevant key again. that any combination of notes played will still be arpeggiated after being released and until they are latched off again, or ARP mode is disabled. The Any notes beyond that number will be played normally. This allows the user to, for instance, arpeggiate a chord with one hand, while playing a melody or accompaniment overtop.

SHIFT

The SHIFT key acts as a modifier for the other function keys, allowing the user to change the remaining settings which don't have dedicated buttons.

- If SHIFT is enabled and both HOLD and ARP are disabled:
  - Octave Up increases the displayed TEMPO.
  - Octave Down decreases the displayed TEMPO.
- If SHIFT and HOLD are enabled and ARP is disabled:
  - Octave Up sets the HOLD sub-mode to HOLD_ALL.
  - Octave Down sets the HOLD sub-mode to HOLD_SET.
- If SHIFT and ARP are enabled and HOLD is disabled,
  - Octave Up cycles through the ARP patterns (ARP MODE).
  - Octave Down cycles through the ARP note length.
- If SHIFT, HOLD, and ARP are enabled:
  - Octave Up increases the SET number.
  - Octave Down decreases the SET number.

APPENDIX E: ENCLOSURE

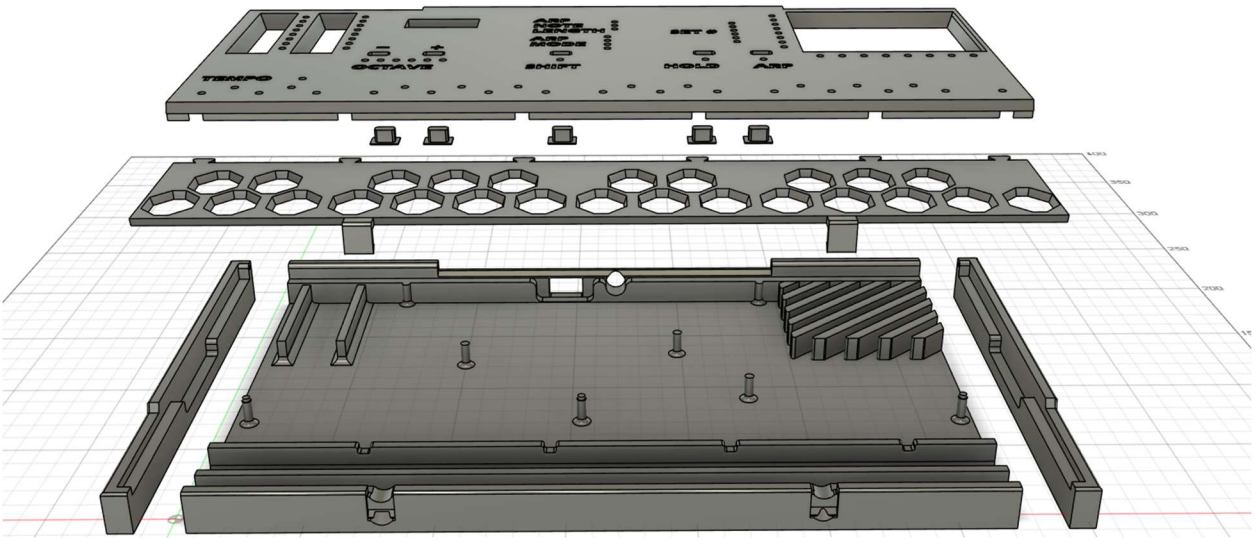

Figure 29: Our custom 3D-printable enclosure.

We can see the enclosure exploded in Figure 29. Visible above the keybed cover are the dovetail joints which lock into the front panel with complementary dovetail recesses. The whole front cover is hinged as a unit and clasps into the front of the bottom section. Holes can be seen on the rear of the unit for connections to either the Teensy (left) or the TRS jack (right).