ECD423 Novel MIDI Instrument

Project Report

2024-05-03

Submitted by:

Sean MacHaffie, Project Lead, Computer Engineering

Baron Li, Finance Manager & Testing Lead, Computer Engineering

Jolin Lin, Software Lead, Computer Engineering

***Rohan Giridharan, Electrical & Communication Lead, Electrical Engineering***

Project Advisor: Prof. Scott Craver

Approved for public release; distribution is unlimited.

Submitted in partial fulfillment of ECD academic requirements.

Thomas J. Watson College of Engineering and Applied Science

Binghamton University

Binghamton, NY

Abstract

ECD 423 Novel MIDI Instruments is a Senior Capstone Project from 2023-2024. The purpose of this project was to design a creative Novel MIDI Instrument capable of polyphony, pitch bend and variable volume control. The product would utilize the functions of a synthesizer. This project is using the open-source synthesizer, Element. The general design consists of a detachable L-shaped rail which can fit on any ordinary tripod boom mic stand. The product consists of 2 keypads and a base station. The base station contains a tilt detection system and is fixed to the center of the rod. Depending on the location of the keypad on the rail, both keypads’ orientation as well as angle of rod tilt, the MIDI device is able to switch between different instruments. The 2 keypads are able to freely slide up and down the rod and act as the inputs for the user to control the available instruments on the mic stand. The entire system consists of 3 microcontrollers and 3 nRF24L01 transceiver modules. The transceiver modules create a long signal transmission line which works to send MIDI signals from the moment the user presses on a key or input device to the moment a note is emitted from the synthesizer.

Table of Contents

[**1 Introduction 1**](#_heading=h.30j0zll)

[1.1 System Overview 1](#_heading=h.1fob9te)

[1.2 Document Overview 1](#_heading=h.3znysh7)

[**2 Referenced Documents 1**](#_heading=h.2et92p0)

[**3 Problem Definition 1**](#_heading=h.tyjcwt)

[3.1 Problem Scope 1](#_heading=h.3dy6vkm)

[3.2 Technical Review 2](#_heading=h.1t3h5sf)

[3.3 Design Requirements 2](#_heading=h.4d34og8)

[3.3.1 Original Proposed Requirements 2](#_heading=h.1b8ngk12yk6c)

[3.3.2 Derived Requirements 2](#_heading=h.epz0wqc7ui9t)

[3.3.3 Stretch Goals 2](#_heading=h.hv0yysu57j3u)

[**4 System Design 3**](#_heading=h.2s8eyo1)

[4.1 System-Wide Design Decisions 3](#_heading=h.17dp8vu)

[4.2 System Components and Interfaces 5](#_heading=h.3rdcrjn)

[4.3 Concept of Execution 6](#_heading=h.26in1rg)

[4.4 Hardware Design 7](#_heading=h.lnxbz9)

[4.5 Software Design 9](#_heading=h.35nkun2)

[4.6 Safety Considerations 11](#_heading=h.1ksv4uv)

[4.7 Environmental Impact 11](#_heading=h.44sinio)

[**5 Project Development 11**](#_heading=h.2jxsxqh)

[5.1 Risk Abatement 11](#_heading=h.z337ya)

[5.2 Project Schedule 12](#_heading=h.3j2qqm3)

[5.3 Project Finances 12](#_heading=h.1y810tw)

[**6 System Implementation 13**](#_heading=h.4i7ojhp)

[**7 Project Evaluation 13**](#_heading=h.2xcytpi)

[7.1 Overview 13](#_heading=h.1ci93xb)

[7.2 Testing and Results 13](#_heading=h.3whwml4)

[7.3 Assessment 13](#_heading=h.2bn6wsx)

[7.4 Future Potential 13](#_heading=h.qsh70q)

[**8 Notes 14**](#_heading=h.3as4poj)

[8.1 Acronyms and Abbreviations 14](#_heading=h.1pxezwc)

[8.2 Bibliography 14](#_heading=h.49x2ik5)

[8.3 References 14](#_heading=h.2p2csry)

[**9 Appendices 15**](#_heading=h.qb58evqq4nh7)

[9.1 Appendix A - Project Specifications Document 15](#_heading=h.dx4015tzyunj)

[9.2 Appendix B - Financial Status Table 17](#_heading=h.myotj4h2x65y)

[9.3 Appendix C - Code for Base Station(Receiver)](#_heading=h.cgrm8n4vjss) 19

9.4 [Appendix D - Code for Left Keypad(Transmitter)](#_heading=h.cgrm8n4vjss) 59

9.5 [Appendix E - Code for Right Keypad(Transmitter)](#_heading=h.cgrm8n4vjss) 63

9.6 [Appendix F -](#_heading=h.cgrm8n4vjss) INT Plan 67

9.7 Appendix G - Demonstration Saxophone Fingering Chart 77

9.8 Appendix H - Parts List 86

9.9 Appendix I - Project Standards 86

9.10 Appendix I - Requirements Impact Assessment 87

List of Figures

Figure 1: Top Plate

Figure 2: Base Plate

Figure 3: High Level Keypad Design

Figure 4: High Level Base Station Keypad Design

Figure 5: Final Keypad Design, showing the 4 lamellophone keys at the bottom and the location of the pitch bend joystick

Figure 6: Preliminary Keypad Design, modeled off a traditional keyboard

Figure 7: Second Keypad Design

Figure 8: Outermost white keys, including lamellophone key attachments. At the back is a hook to attach a rubber band. There is a corresponding hook on the base plate.

Figure 9: Center white keys. These two keys are a slightly different shape than the outermost keys, allowing them to fit better in the base plate

Figure 10. Black key. All keys have a post at the front, which allows them to make proper contact with the force sensitive resistors below

Figure 11. Data Structure that is received by the base station.

Figure 12. Data Structure that holds variables for each key and determines which notes are going to be sent to the synthesizer.

Figure 13. Gantt Chart for the Fall Semester

List of Tables

Table 1. Order Breakdown

# Introduction

## System Overview

This report is for the ECD423 Novel MIDI Instrument project of 2023-2024.

The purpose of this project is to create a Novel MIDI Instrument that is capable of polyphony, variable volume control and pitch bend. The design can be described as a detachable metal rail with 2 sliding keypads, a tilt detection system and a base station which is attachable to the lifting tube of the microphone stand. The base station sends signals to the synthesizer via USB connection. The device is capable of being attached to a tripod boom microphone stand without inhibiting the microphone stand's original functionality. The user is able to play a variety of instruments on the microphone stand. Keypad rotation, keypad location on the rail and rod tilt are taken into account in order to switch between different instruments. The user is able to control the type of notes being produced by the synthesizer using keys on the keypad as well as location of the keypad on the rail. Sliding the keypads up and down the rod can produce note changes depending on what instrument is being played.

## Document Overview

This report provides the essential project requirements in Section 3, Requirements, along with some non-essential stretch goals.

In this specification, a requirement is identified by “shall”, a good practice by “should”, permission by “may” or “can”, expected outcome or action by “will”, and descriptive material by “is” or “are” (or another verb form of “to be”).

# Referenced Documents

The following documents of the exact issue shown form a part of this report as specified herein.

1. ECD423: Novel MIDI Instrument Project Specifications Document

# Problem Definition

## Problem Scope

For musicians, there exists the problem of practicality when playing multiple instruments at the same time. This problem becomes more prevalent for people who enjoy playing as one man bands. Time wasted having to set up each individual instrument on stage, along with having to switch from one musical device to another as well as the difficulty of having to carry these instruments from place to place proves challenging and impractical. The goal of this project is to design and create a Novel MIDI Instrument capable of polyphony, pitch bend and variable volume control. The final product is able to imitate the abilities of a typical instrument as well as switch between multiple different instruments as well.

## Technical Review

The device was created with the intent of being an attachable object onto a typical microphone stand. The product should not inhibit the functions of a regular microphone stand whilst imitating instruments. One advantage that the consumer should notice includes time efficiency when switching from 1 instrument to another. For example, the user would simply need to tilt the metal rod 90 degrees, allowing it to point upwards towards the ceiling, and rotate the left keypad 180 degrees to transition from piano mode to bass guitar mode. This is assuming that the position the user was in prior to the transition allowed the device to be in piano mode. Another advantage includes the ease of learning how to use this product. Since the design is a collection of ordinary musical instruments, users will rarely feel like they need to learn an entirely different skill set to play this device. The instrument modes available on the device were designed to take after the orientation of the instruments and the positioning of the hands for a typical person playing that specific instrument. In this way, users need only remember how they would typically play an instrument and imitate that on the Mic Stand.

## Design Requirements

The contents below contain the original/derived requirements and stretch goals. A tabled version of this is included in the Appendix section of this document.

### 3.3.1 Original Proposed Requirements

1. The MIDI instrument shall be able to produce polyphony, pitch bend and variable volume control. {ECD423-R-001}
2. The MIDI instrument shall be able to produce pitch bend. {ECD423-R-002}
3. The MIDI instrument shall be able to produce variable volume control. {ECD423-R-003}
4. It shall output accurate MIDI signals to the synthesizer. {ECD423-R-004}

### 3.3.2 Derived Requirements

1. The device shall not inhibit the functions of an ordinary microphone stand. {ECD423-R-005}
2. Both keypads will be capable of rotation and sliding movement on the rod. {ECD423-R-006}
3. The device shall be able to switch between at least 6 different instruments based on keypad rotation, keypad location on the rod, and rod tilting motion. {ECD423-R-007}
4. There shall be wireless communication between the keypads and base station. {ECD423-R-008}
5. The device shall be able to detect the position of both keypads on the rod at any given time. {ECD423-R-009}
6. The base station shall have the capability to differentiate between 3 positions on the rod: horizontal, vertical and angled. {ECD423-R-010}
7. MIDI signals shall travel from the base station to a synthesizer via USB connection. {ECD423-R-011}

### 3.3.3 Stretch Goals

1. The response time between note pressing and note emission from the synthesizer should be less than 10ms. {ECD423-G-001}
2. The instrument should be able to synthesize sound directly from the product. {ECD423-G-002}
3. Both keypads should have an electronic locking/unlocking mechanism to secure its position on the rod when locked and allow for movement when unlocked. {ECD423-G-003}
4. The device should be able to switch between at least 10 different instruments based on keypad rotation, keypad location on the rod, and rod tilting motion. {ECD423-G-004}
5. The device should have 4 lamellophone style keys which are attached to force sensors and produce sound reminiscent of strumming string instruments. {ECD423-G-005}

# System Design

## System-Wide Design Decisions

* + 1. **Keypad Position:**  For deciding how to measure the keypads position along the rod there were three options: digital caliper, rotary encoder, and optical mouse sensors. The rotary encoder was chosen for ease of use. Using a readily available Encoder library, the rotary encoder was easily interfaced with the Teensy microcontroller. As the technology of an PS2 optical mouse is more outdated, it was difficult to find a suitable mouse and library. The final option was to imitate a digital caliper. There are no readily available components to make this option possible. This option was dismissed due to time constraints.
    2. **Tilt Detection:**  To measure rod tilt, there were 3 different options. The first option was a mercury tilt sensor, a time-of-flight sensor, and a metal ball tilt sensor. The option that was chosen was the metal ball tilt sensor. This option was chosen due to the simplicity of its implementation. The mercury tilt sensor was dismissed due to environmental concerns, in case the sensor were to break.. The time-of-flight sensor was difficult to connect with the base station in an accurate way. The constantly changing environment of the tilting microphone stand caused inaccurate measurements.
    3. **Key Measurement:** To measure variable volume, the project utilizes force sensitive resistors. These were chosen due to the ease of implementation. Pushing on these force sensitive resistors with the piano keys gives a variable analog value based on the force of the key press. This analog value was easily tied to the variable volume in the synthesizer software.

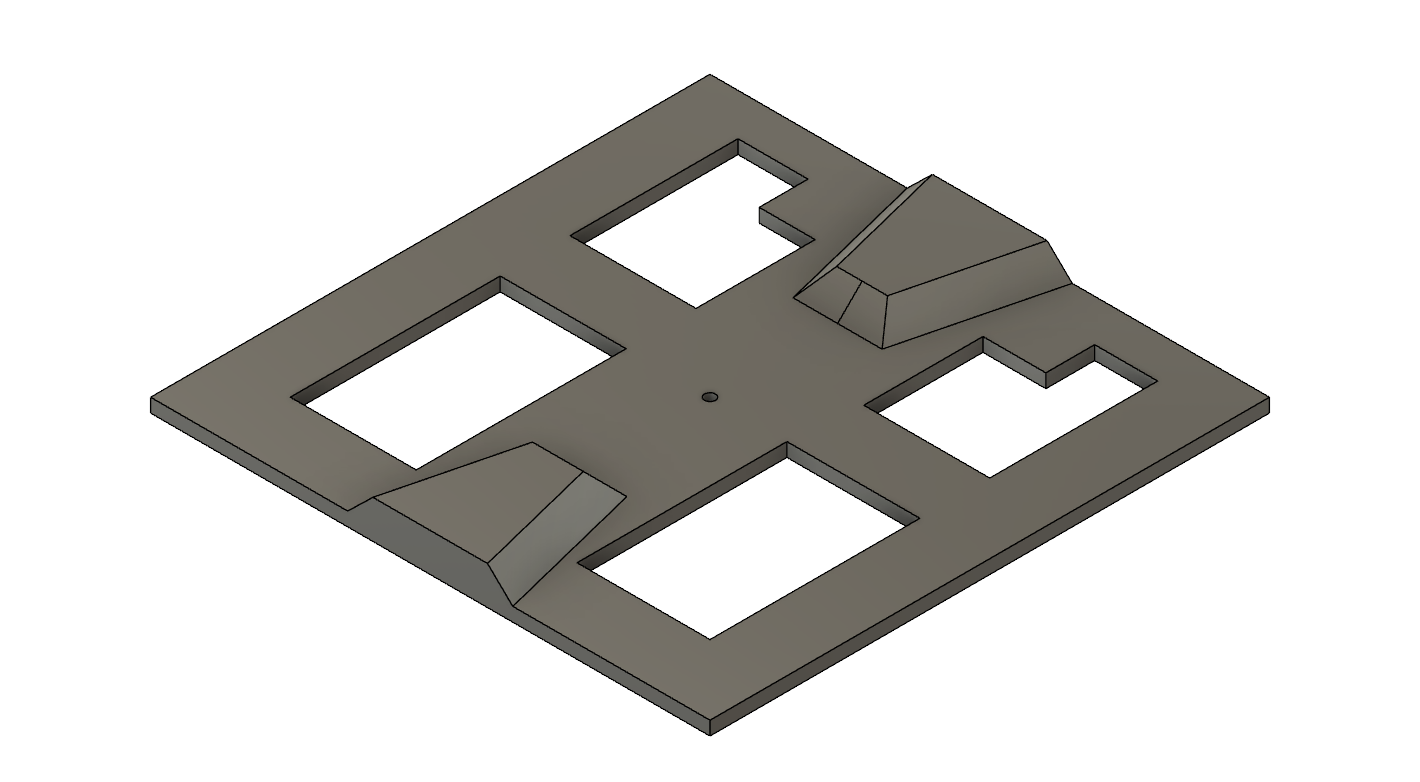


Figure 1: Top Plate

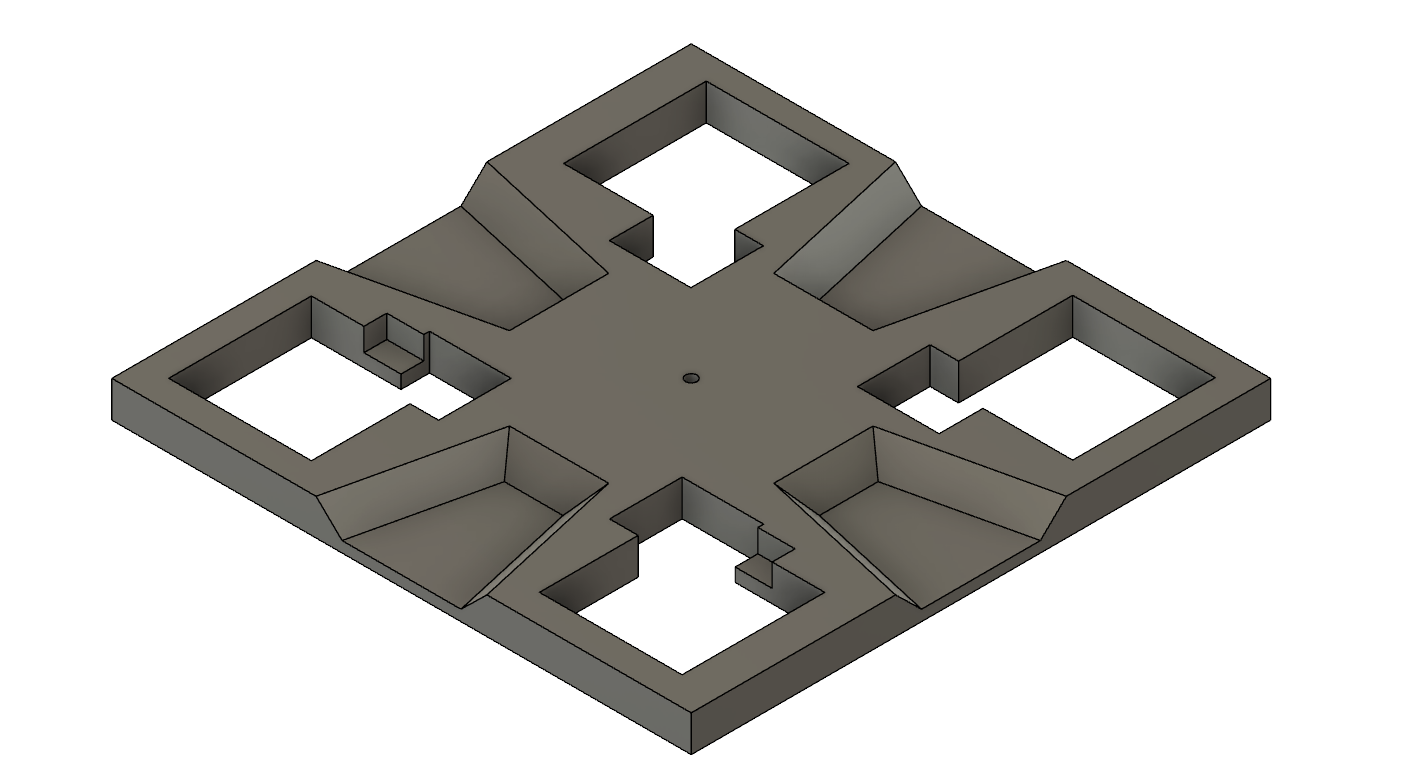


Figure 2: Base Plate

* + 1. **Rotation**: The device has rotating keypads. To accomplish this the keypad has a rotation base plate and top plate shown in figure 1 and 2 respectively. The top plate has contacts that are connected to pull-up resistors which will identify four different rotation positions (up, down, left, right).

## System Components and Interfaces

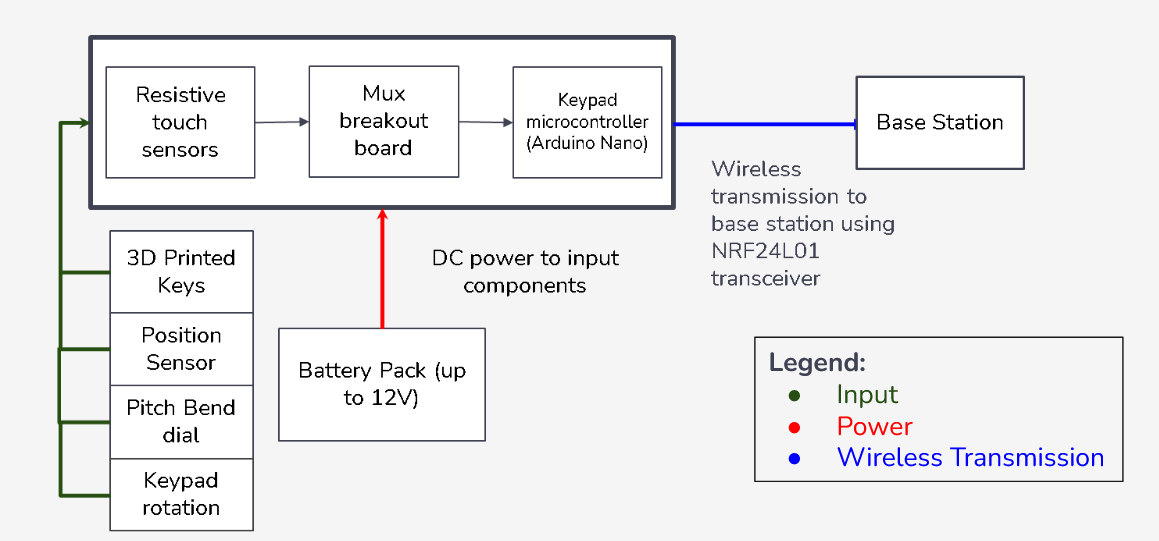


Figure 3: High Level Keypad Diagram

* + 1. **Keypad:** The instrument has two sliding keypads. Each keypad will contain seven force sensitive resistors, one rotary encoder, one NRF2401 wireless transceiver, one analog multiplexer breakout board, one two-axis joystick, and one microcontroller. The microcontroller takes all the inputs and sends them to the base station in a struct. The microcontroller was chosen for its small form factor. The multiplexer breakout board is implemented to allow enough input ports for the keypad components. The seven force sensitive resistors (FSR) are implemented under the seven 3D printed keys. Each FSR provides an analog signal to the microcontroller board, and just as in a traditional keyboard, the harder the key is pressed, the louder the note that is played. The left keypad’s two-axis joystick is used for pitch bend, while the joystick on the right keypad is used to mimic a breath in the woodwind modes. The joysticks are located on the side of the keypad, intended to be positioned under a users’ thumb on each keypad. The NRF24L01 transceiver module is used to wirelessly transmit data. A rotary encoder outputs a value that will increase or decrease to indicate which direction it is turning. The software constantly updates this value starting from a known zero position to accurately track the position of the keypad with this method.

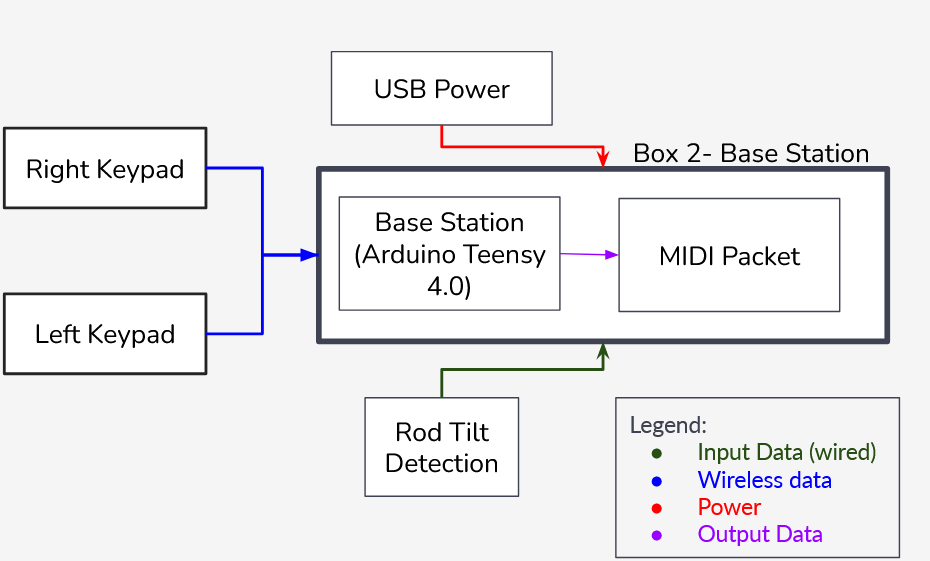


Figure 4: High Level Base Station Keypad Diagram

* + 1. **Base Station:** The instrument has one base station. The base station will use an Arduino Teensy 4.0 microcontroller. The Teensy was chosen for its USB communication capability and its size. The purpose of the base station is to receive wireless data from the keypads. Additionally, the base station has three metal ball tilt sensors that are used to determine the tilting mode of the instrument, whether it be horizontal, vertical or tilted. By combining the tilt motion with the position and rotation of the keypad, this allows the user to change to a number of instruments. Using the MIDIUSB Arduino library, the base station creates a MIDI message which will be sent over USB to the external synthesizer software Element and outputted over the computer speakers.

## Concept of Execution

The target audience uses the MIDI device via interactions with the 2 keypads which rests on the detachable rail. The keypads are ergonomically designed for the left and right hands. The user can determine the orientation of each keypad based on the location of the joystick which is attached to the inside of the keypads. Ideally, the correct orientation of the left keypad for the consumer is one where the joystick is located on the right side of the keypad and the keys are faced in a similar way like a piano. In the instance where the device is in piano mode, this means that the user has allowed the rail to run parallel to the ground with both keypads facing towards him. The device recognizes the switching to this particular instrument mode in part due to the tilt detection system which is made up of tilt switches that turn on and off when the rod is at different angles. The location of the keypad on the rail serves as a crucial aspect in this design. The user can play a variety of notes similar to that of a piano when in piano mode. The inner workings to make this possible involve the transmission of signals produced when the user hits a key. A signal gets generated and travels through an Arduino Nano to a wireless transceiver which then is accepted by a receiver on the base station to a Teensy where the MIDI signal is finally sent to the synthesizer for the correct note to be emitted. This isn’t the only thing that is sent through the transmission line because underneath each key lies a force resistive sensor which collects the pressure exerted by the user on the keys. The range of pressure applied to these keys controls the volume at which the user wants to emit the notes at. The consumer would only need to exert more pressure when wanting to evoke a louder note and exert less pressure for the opposite effect. Moving the joystick in different directions allows the audience to control pitch bend to produce a musical effect of notes sliding into other notes. This information is also sent along the transmission line to be unpackaged and in the Teensy. Since the keypads communicate wirelessly, each keypad will have a rechargeable battery while the base station is powered by the laptop via USB connection. To break this connection and shut off the system, the user only needs to disconnect the laptop from the base station.

## Hardware Design

All 3D models for hardware components were made in Autodesk Fusion 360. This includes the following: white and black keys, a keypad base, rotation mechanism (shown in section 4.1.3) and a slider attachment. While brainstorming, there were multiple keypad designs before the final design shown in Figure XI. The first design was modeled off what would be found on a traditional keyboard, which is shown in figure XII. The lamellophone design was chosen as it would allow the user to be able to play instruments like a bass or a guitar, which require a plucking or strumming motion. This design is not conducive to strumming, and would therefore be extremely impractical and difficult to implement. Implementing this design would also result in a lot of additional hand movement to change the set of notes that were being played. The second design (Figure XIII) consisted of 5 white keys and 4 black keys, where the black keys would be positioned in such a way that they could be used as an analog for the strumming instruments. The black keys would be sufficiently high above the white keys and would require a spring mechanism that could provide a somewhat stringlike motion. While far more practical than the first design, implementing this design would have required a lot of testing to find not only the right kind of spring, but also the necessary actuation force. This would have taken up a lot of valuable time during the spring semester. The current design is a refinement of design 2, adding the lamellophone keys at the bottom of the keypad to imitate a string instrument, as well as a joystick for pitch bend on the side, in reach of the user’s thumb. After coming up with the final keypad design, the models were created for the keypad base, and the black, white and lamellophone keys. There are three different versions of the white keys, one for the left side, one for the right side and one for the two center keys. At the end of the white and black keys, there is a hook, which will be used as an attachment point for the rubber bands to ensure the key returns to its original position after being pressed. Rather than use a spring, rubber bands were used as the actuation force could be easily tuned by simply changing the amount of rubber bands, rather than swapping out the entire spring. As was the case with keypad design 2, this will ultimately save a lot of time while testing. All keys have a post at the front, which allows them to make proper contact with the force sensitive resistors (FSRs) below. As with a traditional piano, the harder you press on the keys, the louder the sound of the note.

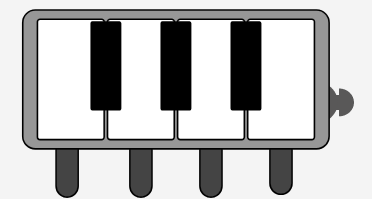


Figure 5: Final Keypad Design, showing the 4 lamellophone keys at the bottom and the location of the pitch bend joystick

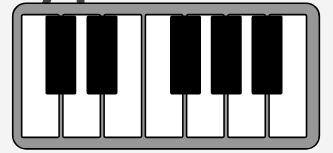


Figure 6: Preliminary Keypad Design, modeled off a traditional keyboard

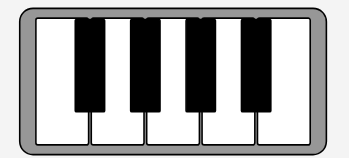


Figure 7: Second Keypad Design

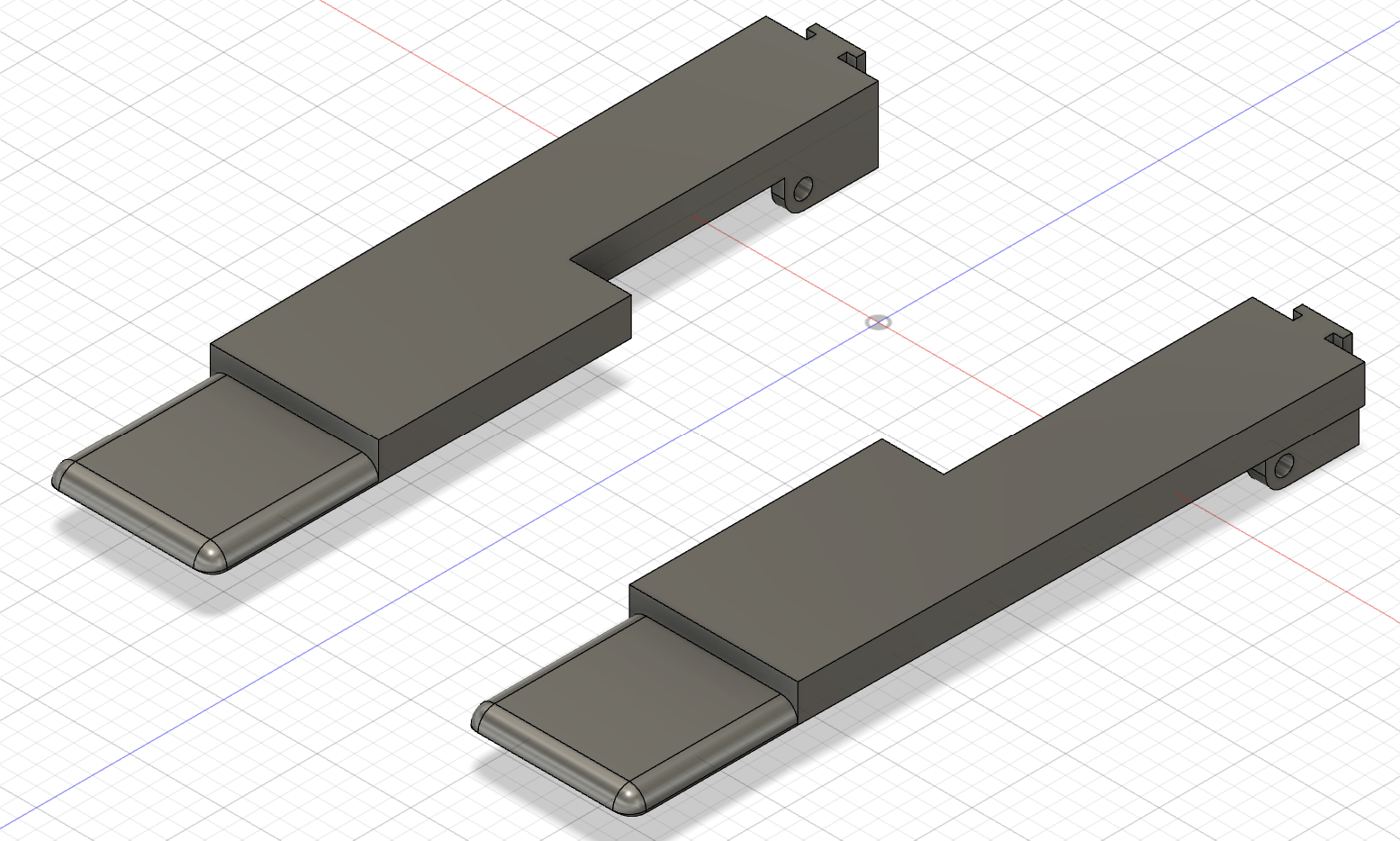


Figure 8: Outermost white keys, including lamellophone key attachments. At the back is a hook to attach a rubber band. There is a corresponding hook on the base plate.

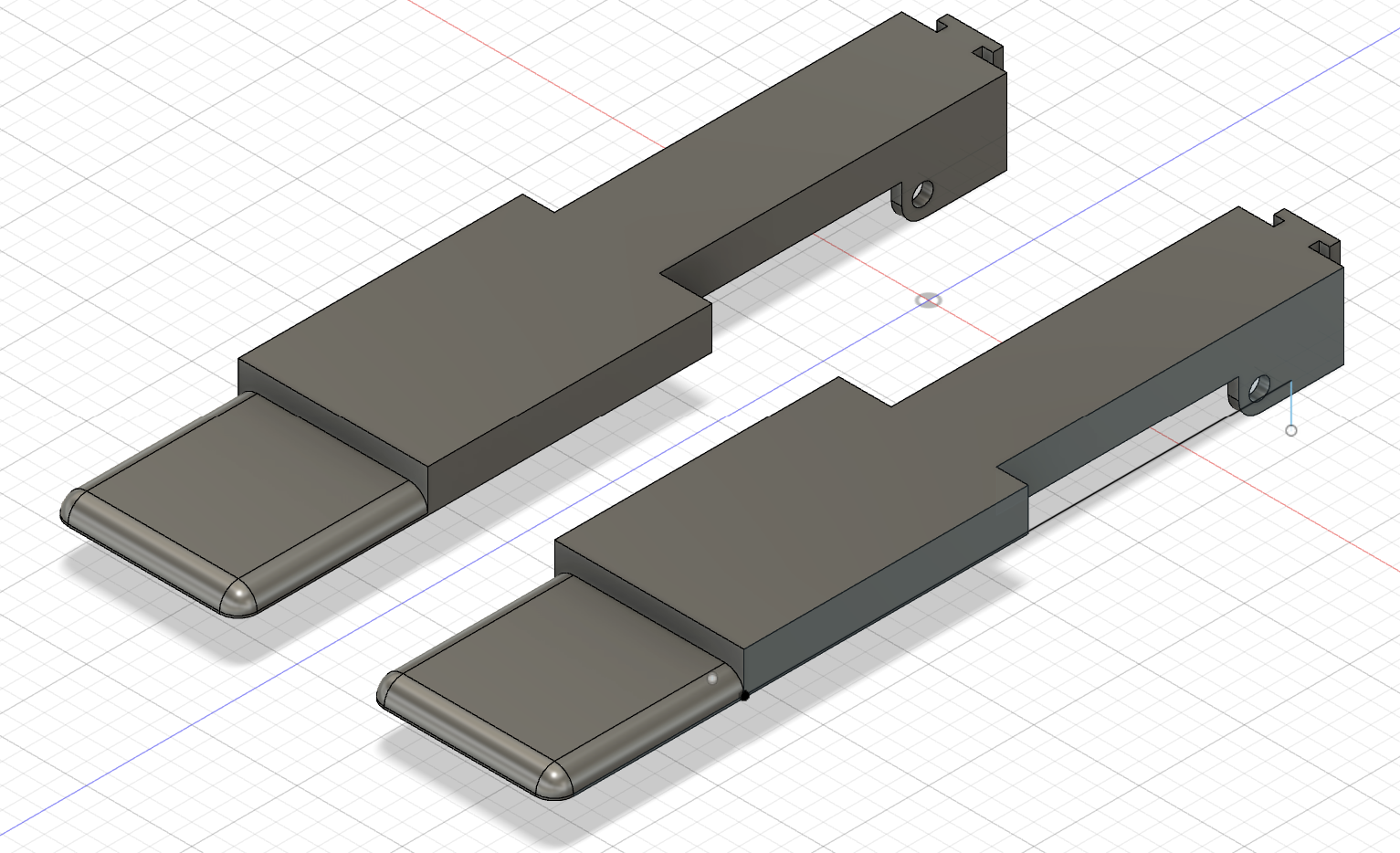


Figure 9: Center white keys. These two keys are a slightly different shape than the outermost keys, allowing them to fit better in the base plate

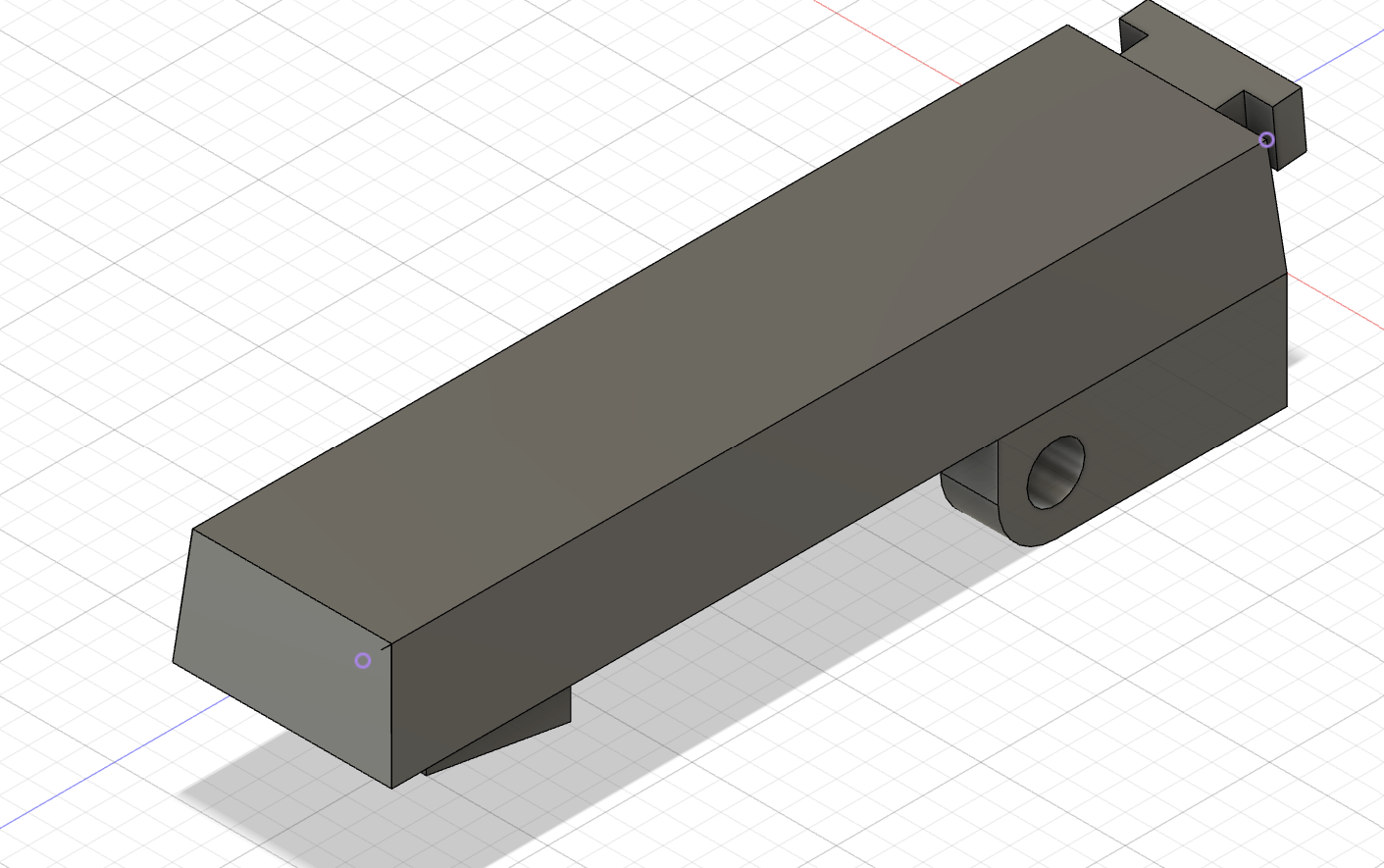


Figure 10: Black key. All keys have a post at the front, which allows them to make proper contact with the force sensitive resistors below

## Software Design

All software that was constructed was created using Arduino IDE. The final programmed files were only created after running multiple demos which involved manipulating a set of test signals to send them to their ultimate destination, the Element synthesizer. Demo code for this project started off with MIDI signals traveling close to the synthesizer relative to the entire transmission line that the signal would eventually have to pass through from start to finish. The program experimented with using a backwards approach by first sending MIDI signals from the base station micro controller to the synthesizer directly through a USB connection. Wireless communication between the base station and keypads were established with two nRF24L01 set to transceiver mode in the keypads and the final nRF24L01 set to receiver mode in the base station. This allowed consistent one directional communication between the keypads and the base station when the system was on. Code was then generated once the Arduino Nano’s on each keypad were connected to the wireless transceivers to send test signals starting from the Nano and traveling through the transmission pipe to the synthesizer. The final step in this was performed through a combination of circuitry knowledge and programming skills, as the test signals no longer needed to be hard coded onto the board. Rather a test user would press on the keys themselves to generate signals. The same was done for the joysticks. During this entire process, all demo programs were consolidated into one location with proper formatting and comments for readability and correct interpretation of program intent.

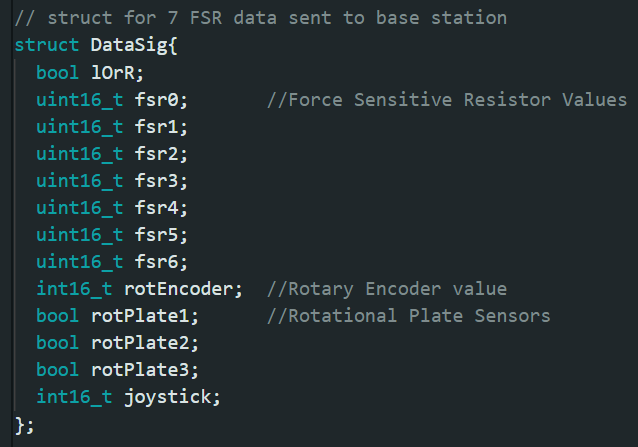


Figure 11: This is the data structure that is received by the base station.

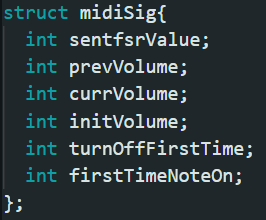


Figure 12: This structure holds variables for each key and determines which notes are going to be sent to the synthesizer.

The final code contains three general programs which had to be uploaded to both keypads and the base station. Please note that the file name demo13\_finalTestTxL contains code for the left keypad while demo13\_finalTestTxR is meant for the right. Since both keypads differ slightly by the soldering of different pins, both keypads must have independent files. Moving on, the file demo13\_finalTestRx is code for the receiver or base station. In summary, the receiver code takes data signals from both transmitters and saves them in separate structures. Data within these structures use the switchInstruments() function to determine which instrument setting the device is on. Depending on the instrument mode, certain functions may be invoked which ultimately determine the instructions to send to the synthesizer. Instructions that may be used include sending notes on or off, which notes to play, what instrument channel to play them with, what volume to play it on and whether there is pitch bend or not.

## Safety Considerations

In the brainstorming process of the novel instrument, ensuring safety was a key consideration in the design discussion. Instead of employing mercury tilt switches for the rod tilt detection, the design opts for metal ball tilt switches. Mercury is classified as a toxic heavy metal. In consideration of health and environmental concerns, the mercury tilt switches were excluded from consideration. All wiring will be discreetly embedded within the instrument housing, eliminating potential environmental hazards with exposed wires.

## Environmental Impact

As mentioned in the safety considerations, the environmental impact was an important consideration when designing the instrument. The metal ball tilt switches do not have the same environmental risk of mercury contamination. The keypads will be powered with a rechargeable battery pack. This choice was made with sustainability in mind, and eliminates the need for battery replacements and minimizes the waste in comparison to single-use disposable batteries. The housings and keypads are created with CAD and printed with ABS plastic, a commonly used thermoplastic in 3D printing. ABS plastic is durable, safe for consumer use, and can be melted and recycled.

# Project Development

## Risk Abatement

Some risks that may affect this project are: integration of many different hardware components, too many inputs into the microcontroller, and total propagation time from note press to sound output. To mitigate these risks, all components are tested and verified separately before integration. All ordered components must also be well-documented. To resolve the potential error of not having enough inputs into the microcontroller, a multiplexer breakout board was procured. To mitigate the propagation time, a low latency wireless standard was selected. The design utilizes low latency components such as the NRF24L01 transceiver.

## Project Schedule

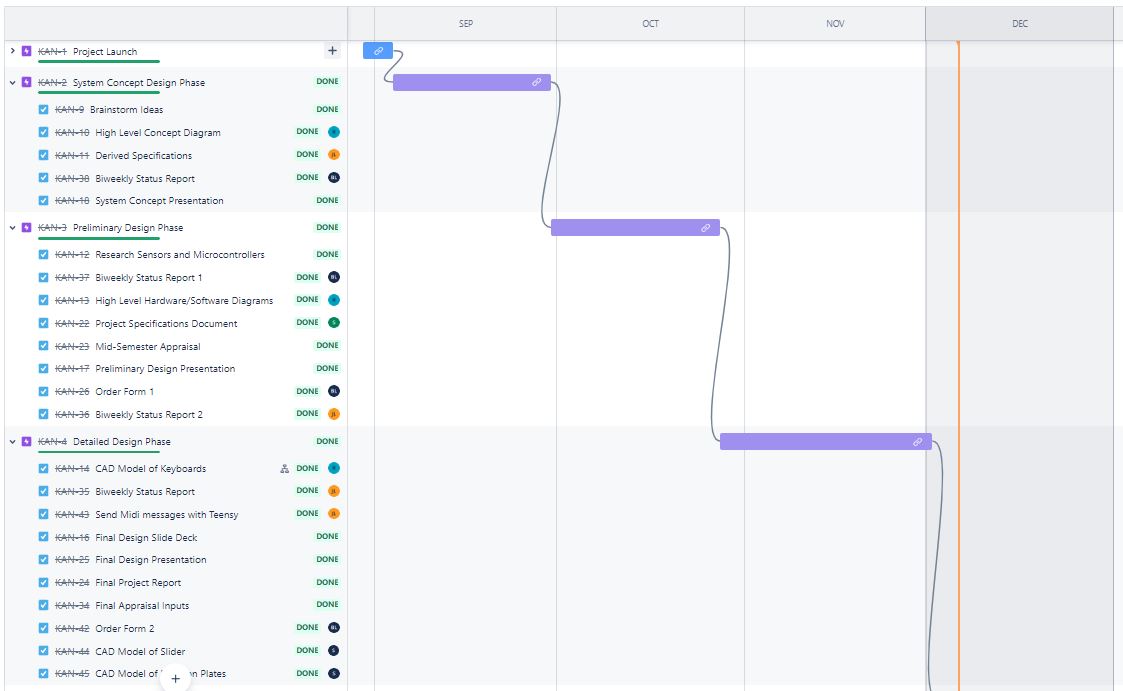


Figure 11: Gantt Chart for the Fall Semester

The project was formally launched on September 1st, 2023. After a brainstorming process, a list was compiled for September.. Research for sensors began in October and early CAD designs for housing components were proposed. The team created high level software and hardware diagrams. The design was finalized in early November, as testing began for some of the sensors and components. Test programs for the MIDI synthesizer and controller were made to successfully generate MIDI signals using the Arduino Teensy and the Element software. Final keys and keypad designs were made and ready to be printed. Throughout the Spring semester, the sensors were integrated into the base station and keypads. After integrating the components together on the microphone stand, thorough testing, calibrating and debugging was done to finalize the prototype.

## Project Finances

A budget of $800 has been allocated to the project’s fulfillment. Electronic equipment and materials were provided by the Seymour Kunis Media Core. A breakdown of the order cost can be seen in Appendix B. As the Novel MIDI Instrument designed for this project was complex, it had surpassed the expected expenditure of $150 - $200 as seen in previous iterations of the project. Due to the high cost of the required sensors and the extensive testing for optimal and compatible components, a total of approximately $400 was spent. As not all of the ordered sensors were used in the final design, the cost of the prototype would be closer to $200.

# System Implementation

The final design contains the 2 keypads with a base station. The electronics of the keypads were assembled with a soldered project board, with some pin extenders that were used to plug in some components so they could be removable. This was done to the Teensy that is on board along with the pins for the rotary encoder, the pins for the NRF24L01 transceiver and battery.

For better use of the rotary encoder, a gear was attached on the knob of the encoder and a track along the rail to ensure that no rotations were missed when the keypad moved along the rod. to attach the keypads top and bottom plate, a screw attached to an U shaped piece of 3d printed plastic which acted as a spring. This ensured that the top plate had enough pressure to attach to the base plate but could be lifted and rotated by the user.

To power the keypads, AA batteries were used. Due to the limited supplies of the Seymour Kunis Media Core, one keypad was powered by 4 AA batteries and the other keypad was powered by 3 AA batteries. The 4 AA batteries worked better due to a longer battery life and better counter weight for the keypad.

For the actual keyboard, a 10-gauge wire was inserted as an axle for all the individual keys to rest on. Orthodontic elastic bands functioned well with the axle to keep the keys in a default position. The bands were also readily available in the Media Core. A spacer was added below the keypad so the bands can be changed after assembly.

The base station and the tilt sensor were combined into one unit for simplicity. The tilt sensor was left on a solderless breadboard due to the position of the metal ball tilt switches being very important. This station was attached to the rod with some trouble sided tape.

# Project Evaluation

## Overview

To evaluate this project, it was demonstrated that each test could be independently performed. After proving successful, the goal was to integrate the systems to test again.

An example of this would be testing wireless communication between two microcontrollers, then performing the test again with an assembled keypad unit and base station. Due to unforeseen integration issues, the verification phase did not end until April 15th, 2024. While many of the tests were met, there were lapses in reliability that would cause the test to fail occasionally. Overall, the major requirements were all met and most of the derived requirements.

## Testing and Results

A list of tests and how they were performed can be found in Appendix F. The instrument was able to reliably detect the three levels of tilt. The rotary encoder could read the rolling along the rod with minimal error. The rotation system was able to detect 4 different positions, however it only worked well on a flat surface. Wireless communication was able to send and receive messages in the allotted time frame.

## Assessment

The instrument was able to meet all of the given requirements. It could accurately and reliably submit MIDI messages to a synthesizer, do variable volume, polyphony and pitch bend. The instruments did meet most of the derived requirements. The instrument is capable of switching between 6 different instruments but could only reliably switch between 4 of the instruments. Wireless communication using an NRF24l01 transceiver between the base station and the keypads was successful. No permanent changes to the microphone stand were required in the prototype’s implementation. Due to time constraints, not all of the stretch goals were met. The lamellophone key stretch goal was met. The design did have some reliability issues that did cause issues when demonstrating the capabilities of the instrument.

## Future Potential

As future groups are expected to design a new novel MIDI instrument, there will be no future modifications. If this design was revisited in the future, potential changes and improvements would involve creating a more stable design, increasing the accuracy of the sensors and integrating more instruments in software.

# Notes

## Acronyms and Abbreviations

ECD ECE Capstone Design

ECE Electrical and Computer Engineering

MIDI Musical Instrument Digital Interface

## Bibliography

Not applicable.

## References

[1] *Teensyduino: Download and Install Teensy support into the Arduino IDE*. (2017). Pjrc.com. <https://www.pjrc.com/teensy/td_download.html>

[2] *Getting Started with the Teensy - SparkFun Learn*. (2018). Sparkfun.com. <https://learn.sparkfun.com/tutorials/getting-started-with-the-teensy/programming>

[3] ada, lady. (2012, July 29). *Tilt Sensor*. Adafruit Learning System. <https://learn.adafruit.com/tilt-sensor/overview>

[4] *RP-C18.3-ST Thin Film Pressure Sensor SKU:SEN0294*. (n.d.). Retrieved December 7, 2023, from <https://mm.digikey.com/Volume0/opasdata/d220001/medias/docus/695/SEN0294_Web.pdf>

# Appendices

## Appendix A - Project Specifications Document

**ECD Project Specification Document**

***ECD423 Novel MIDI Instrument***

**Project Description**

**Summary:**The premise of this project is to design a creative and unique MIDI interface that is able to imitate a variety of musical instruments. The MIDI device must be able to have polyphony, demonstrate variable volume control and be capable of pitch bend. Due to the obscurity of this project, a major portion of the time spent on this proposal will be spent brainstorming potential ideas. Continuous experimentation and trials will take place throughout the entire design process to guarantee the optimal usability of each component in this project.

**Sponsor:** N/A

**Industry Mentor (if applicable):** N/A

**Project Advisor:** Professor Scott Craver

**Team:** Sean MacHaffie, Baron Li, Rohan Giridharan, Jolin Lin

**Requirements**

This document lists all essential project requirements for this project. A requirement is identified by “shall”, a good practice by “should”, permission by “may” or “can”, expected outcome or action by “will”, and descriptive material by “is” or “are” (or another verb form of “to be”).

The following Qualification Method (QM) is to be used:

* Demonstration (D): The operation of the system, or a part of the system, that relies on observable functional operation not requiring the use of instrumentation, special test equipment, or subsequent analysis.
* Test (T): The operation of the system, or a part of the system, that uses instrumentation or other special test equipment to collect data for analysis.
* Analysis (A): The processing of data obtained from another qualification method. For example, reduction, interpolation, or extrapolation of test results.
* Inspection (I): The visual examination of system components, documentation, etc.

The following Requirement Categories (RC) are to be used:

* System Capability Requirements (SC): Requirements pertaining to the functionality and behavior of the system.
* System External Interface Requirements (EI): Requirements based on the external interfaces of the system. Interfaces with input power, user input, or any other outside source
* Project Business Requirements (PB): Requirements pertaining to business objectives set by a sponsor such as installation requirements, requirements pertaining to specific lab access or lab equipment needs etc.
* Other Requirements (O): Safety, Security and Privacy, System Environment concerns etc.

**2.1 Derived Requirement Specification**

| ID | QM | RC | Derived Requirement |
| --- | --- | --- | --- |
| ECD423-R-001 | D | SC | The MIDI instrument shall be able to produce polyphony. |
| ECD423-R-002 | D | SC | The MIDI instrument shall be able to produce pitch bend. |
| ECD423-R-003 | D | SC | The MIDI instrument shall be able to produce variable volume control. |
| ECD423-R-004 | I | O | It shall output accurate MIDI signals to a synthesizer. |
| ECD423-R-005 | I | SC | The device shall not inhibit the functions of an ordinary microphone stand. |
| ECD423-R-006 | D | SC | Both keypads will be capable of rotation and sliding movement on the rod. |
| ECD423-R-007 | D | SC | The device shall be able to switch between at least 6 different instruments based on keypad rotation, keypad location on rod, and rod tilting motion. |
| ECD423-R-008 | I | SC | There shall be wireless communication between the keypads and base station. |
| ECD423-R-009 | A | SC | The device shall be able to detect the position of both keypads on the rod at any given time. |
| ECD423-R-010 | I | SC | The base station shall have the capability to differentiate between 3 positions on the rod: horizontal, vertical, and angled. |
| ECD423-R-011 | T | EI | MIDI signals shall travel from the base station to a synthesizer via USB connection. |

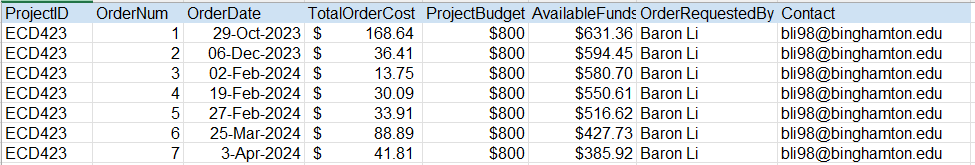
**2.2 Derived Stretch Goals**

| ID | QM | RC | Stretch Goals |
| --- | --- | --- | --- |
| ECD423-G-001 | T | O | The response time between note pressing and note emission from the synthesizer should be less than 10ms. |
| ECD423-G-002 | I | EI | The instrument should be able to synthesize sound directly from the product. |
| ECD423-G-003 | I | SC | Both keypads should have an electronic locking/unlocking mechanism to secure its position on the rod when locked and allow for movement when unlocked. |
| ECD423-G-004 | D | SC | The device should be able to switch between at least 10 different instruments based on keypad rotation, keypad location on rod, and rod tilting motion. |
| ECD423-G-005 | I | SC | The device should have 4 lamellophone style keys which are attached to force sensors and produce sound reminiscent of strumming string instruments. |

**2.3 Original Proposed Requirements**

| QM | RC | Original Requirement |
| --- | --- | --- |
| D | SC | The MIDI instrument shall be able to produce polyphony. |
| D | SC | The MIDI instrument shall be able to produce pitch bend. |
| D | SC | The MIDI instrument shall be able to produce variable volume control. |
| I | EI | It shall output accurate MIDI signals to a synthesizer. |

## Appendix B - Financial Status Table



## Appendix C - Code for Base Station(Receiver)

/\*

Programmer: Jolin Lin

ECD 423 - Novel MIDI Instrument (Senior PRJ)

Date: 04/24/2024

Purpose: This is the program to ran during the Expo(04/26/2024).

The purpose of this file is to be uploaded to the receiver/base station.

It takes in data signals sent from both transmitters/keypads to switch btw

instruments in the synthesizer, have variable volume and polyphony.

Notes:

\*/

#include <SPI.h>

#include <RF24.h>

#include <Control\_Surface.h>

#include <print.h>

#define SIZE 3 //size of volume history

#define TILTSIZE 10 //size of tilt sensor history

#define KEYNUM 7 //size of number of keys on a keypad

#define JOYSTICKUPPERLIMIT 560

#define JOYSTICKLOWERLIMIT 470

RF24 radio(9,10); // pin assignments for nRF CE,CSN

USBMIDI\_Interface midi; //MIDI over USB interface object

//---------------------------------------Setup----------------------------------------------------------

const int sensorVPin = 5;

const int sensorAPin = 19;

const int sensorHPin = 20;

uint8\_t pipe1 = 1; // pipe values may go from 0-6 and represents the individual

uint8\_t pipe2 = 2; // paths available for 1 nRF module to be able to receive data from

//---------------------------------------Global Variables------------------------------------------------

const uint8\_t velocity = 0x7F;

const uint8\_t offvelocity = 0x00;

const uint8\_t ccAddr = MIDI\_CC::Channel\_Volume; //changes control channel to ChannelVolume

int8\_t tiltSensor = 0b000; //global variables associated with tilt sensor noise elimination

int cmpTilt;

int8\_t sameValCount = 0;

int secondTiltVal = 0;

int tiltSensHist[TILTSIZE]; //create an array with size 10 to collect tiltSensor History that updates with every loop

int8\_t iteratetiltHist= -1;

int8\_t i\_collectData = 0;

int8\_t rotPlateHist[2] = {0,0};

int8\_t startingMuxSel = 0;

int16\_t maxRE = 385;

int16\_t maxReRange = maxRE/6;

int16\_t borderValue\_L = 25; //boundary for rotary encoder to switch between clarinet and sax

//---------------------------------------Structs and Initalization----------------------------------------

//Piano and Bass Guitar: Note Addr for each Key on the Keypad

MIDIAddress noteAddr1L; //left most white key

MIDIAddress noteAddr2L;

MIDIAddress noteAddr3L;

MIDIAddress noteAddr4L; //right most white key

MIDIAddress noteAddr5L; //left most black key

MIDIAddress noteAddr6L;

MIDIAddress noteAddr7L; //right most black key

MIDIAddress noteAddr1R; //left most white key

MIDIAddress noteAddr2R;

MIDIAddress noteAddr3R;

MIDIAddress noteAddr4R; //right most white key

MIDIAddress noteAddr5R; //left most black key

MIDIAddress noteAddr6R;

MIDIAddress noteAddr7R; //right most black key

//Sax Mode: Only 1 note addr needed for sax

MIDIAddress saxNoteAddr;

MIDIAddress clarinetNoteAddr;

MIDIAddress fluteNoteAddr;

// struct for 7 FSR data sent to base station

struct DataSig{

bool lOrR;

uint16\_t fsr0; //Force Sensitive Resistor Values

uint16\_t fsr1;

uint16\_t fsr2;

uint16\_t fsr3;

uint16\_t fsr4;

uint16\_t fsr5;

uint16\_t fsr6;

int16\_t rotEncoder; //Rotary Encoder value

bool rotPlate1; //Rotational Plate Sensors

bool rotPlate2;

bool rotPlate3;

int16\_t joystick;

};

//initalize struct for DataSig

struct DataSig dataSigReceivedL;

struct DataSig dataSigReceivedR;

struct DataSig tempSig;

// this struct holds variables for each key and determines which notes

// to send to the synthesizer

struct midiSig{

int sentfsrValue;

int prevVolume;

int currVolume;

int initVolume;

int turnOffFirstTime;

int firstTimeNoteOn;

};

// initialize struct for individual keys

midiSig key1DataL = {0,0,0,0,1,1};

midiSig key2DataL = {0,0,0,0,1,1};

midiSig key3DataL = {0,0,0,0,1,1};

midiSig key4DataL = {0,0,0,0,1,1};

midiSig key5DataL = {0,0,0,0,1,1};

midiSig key6DataL = {0,0,0,0,1,1};

midiSig key7DataL = {0,0,0,0,1,1};

midiSig fsrStructL ={0,0,0,0,1,1}; //temp struct for left keys

midiSig key1DataR = {0,0,0,0,1,1};

midiSig key2DataR = {0,0,0,0,1,1};

midiSig key3DataR = {0,0,0,0,1,1};

midiSig key4DataR = {0,0,0,0,1,1};

midiSig key5DataR = {0,0,0,0,1,1};

midiSig key6DataR = {0,0,0,0,1,1};

midiSig key7DataR = {0,0,0,0,1,1};

midiSig fsrStructR = {0,0,0,0,1,1}; //temp struct for right keys

// a struct to keep track of which note is on at any given time

struct noteOnHis{

bool ith\_Val[KEYNUM];

char whichKeypad[KEYNUM];

};

struct noteOnHis noteOnSig = {{0,0,0,0,0,0,0}, {'N','N','N','N','N','N','N'}};

using namespace MIDI\_Notes; // Define the MIDI note names and their corresponding values

//---------------------------------------Functions Start Here------------------------------------------------

// Piano: Function to produce key fade in piano mode

void fadeVolume(int ith\_Val, char whichKeypad){

if(whichKeypad == 'L'){

if(fsrStructL.initVolume > fsrStructL.currVolume){

//decrease volume very fast

for(int i=fsrStructL.initVolume; i>=0; i-=10)

{

midi.sendControlChange(ccAddr, fsrStructL.initVolume); //changes CC to volume control option

// would be sending the prev. volume

chooseNoteOn(ith\_Val, 'L');

}

}

else{

// decrease volume slowly

for(int i=fsrStructL.initVolume; i>=0; i-=5)

{

midi.sendControlChange(ccAddr, fsrStructL.initVolume); //changes CC to volume control option

// would be sending the prev. volume

chooseNoteOn(ith\_Val, 'L');

}

}

}

else{

if(fsrStructR.initVolume > fsrStructR.currVolume){

//decrease volume very fast

for(int i=fsrStructR.initVolume; i>=0; i-=10)

{

midi.sendControlChange(ccAddr, fsrStructR.initVolume); //changes CC to volume control option

// would be sending the prev. volume

chooseNoteOn(ith\_Val, 'R');

}

}

else{

// decrease volume slowly

for(int i=fsrStructR.initVolume; i>=0; i-=5)

{

midi.sendControlChange(ccAddr, fsrStructR.initVolume); //changes CC to volume control option

// would be sending the prev. volume

chooseNoteOn(ith\_Val, 'R');

}

}

}

//update noteOnHis struct here

noteOnSig.ith\_Val[ith\_Val] = 1;

noteOnSig.whichKeypad[ith\_Val] = whichKeypad;

}

// Piano: Function assigns which noteAddr gets turned on

void chooseNoteOn(int i, char whichKeypad){

if(i == (startingMuxSel)){

if(whichKeypad == 'L'){

midi.sendNoteOn(noteAddr1L, velocity);

}

else{

midi.sendNoteOn(noteAddr1R, velocity);

}

}

else if(i == (startingMuxSel+0b0001)){

if(whichKeypad == 'L'){

midi.sendNoteOn(noteAddr2L, velocity);

}

else{

midi.sendNoteOn(noteAddr2R, velocity);

}

}

else if(i == (startingMuxSel+0b0010)){

if(whichKeypad == 'L'){

midi.sendNoteOn(noteAddr3L, velocity);

}

else{

midi.sendNoteOn(noteAddr3R, velocity);

}

}

else if(i == (startingMuxSel+0b0011)){

if(whichKeypad == 'L'){

midi.sendNoteOn(noteAddr4L, velocity);

}

else{

midi.sendNoteOn(noteAddr4R, velocity);

}

}

else if(i == (startingMuxSel+0b0100)){

if(whichKeypad == 'L'){

midi.sendNoteOn(noteAddr5L, velocity);

}

else{

midi.sendNoteOn(noteAddr5R, velocity);

}

}

else if(i == (startingMuxSel+0b0101)){

if(whichKeypad == 'L'){

midi.sendNoteOn(noteAddr6L, velocity);

}

else{

midi.sendNoteOn(noteAddr6R, velocity);

}

}

else if(i == (startingMuxSel+0b0110)){ //since B sharp dne, do not turn on in even zones

if(whichKeypad == 'L'){

midi.sendNoteOn(noteAddr7L, velocity);

}

else{

midi.sendNoteOn(noteAddr7R, velocity);

}

}

}

// Piano: Function to turn sendNoteOff depending on i

void chooseNoteOff(int i, char whichKeypad){

if(i == startingMuxSel){

if(whichKeypad == 'L'){

midi.sendNoteOff(noteAddr1L, velocity);

}

else{

midi.sendNoteOff(noteAddr1R, velocity);

}

}

else if(i == (startingMuxSel+0b0001)){

if(whichKeypad == 'L'){

midi.sendNoteOff(noteAddr2L, velocity);

}

else{

midi.sendNoteOff(noteAddr2R, velocity);

}

}

else if(i == (startingMuxSel+0b0010)){

if(whichKeypad == 'L'){

midi.sendNoteOff(noteAddr3L, velocity);

}

else{

midi.sendNoteOff(noteAddr3R, velocity);

}

}

else if(i == (startingMuxSel+0b0011)){

if(whichKeypad == 'L'){

midi.sendNoteOff(noteAddr4L, velocity);

}

else{

midi.sendNoteOff(noteAddr4R, velocity);

}

}

else if(i == (startingMuxSel+0b0100)){

if(whichKeypad == 'L'){

midi.sendNoteOff(noteAddr5L, velocity);

}

else{

midi.sendNoteOff(noteAddr5R, velocity);

}

}

else if(i == (startingMuxSel+0b0101)){

if(whichKeypad == 'L'){

midi.sendNoteOff(noteAddr6L, velocity);

}

else{

midi.sendNoteOff(noteAddr6R, velocity);

}

}

else if(i == (startingMuxSel+0b0110)){

if(whichKeypad == 'L'){

midi.sendNoteOff(noteAddr7L, velocity);

}

else{

midi.sendNoteOff(noteAddr7R, velocity);

}

}

}

// Function to eliminate noise on tiltSensor

int tiltSensorNoise(int8\_t tiltSensor){

//Update tiltSensHist with tiltSensor by shifting items to the left

iteratetiltHist += 1;

if(iteratetiltHist >= 9){iteratetiltHist = 0; }

tiltSensHist[iteratetiltHist+1] = tiltSensHist[iteratetiltHist];

tiltSensHist[0] = tiltSensor;

cmpTilt = tiltSensor;

if(cmpTilt == tiltSensHist[iteratetiltHist]){

sameValCount++;

}

else{

secondTiltVal = tiltSensHist[iteratetiltHist];

}

if(sameValCount < 10 && (iteratetiltHist == 0)){ // if identical value is less than 5 update cmpTilt

cmpTilt = secondTiltVal;

}

return cmpTilt;

}

// Function assigns the appropriate struct to the for loop in the main depending on the i value

void pickStruct(int i){

if(i == (startingMuxSel)){

memcpy(&fsrStructL, &key1DataL, sizeof(key1DataL));

memcpy(&fsrStructR, &key1DataR, sizeof(key1DataR));

}

else if(i == (startingMuxSel+0b0001)){

memcpy(&fsrStructL, &key2DataL, sizeof(key2DataL));

memcpy(&fsrStructR, &key2DataR, sizeof(key2DataR));

}

else if(i == (startingMuxSel+0b0010)){

memcpy(&fsrStructL, &key3DataL, sizeof(key3DataL));

memcpy(&fsrStructR, &key3DataR, sizeof(key3DataR));

}

else if(i == (startingMuxSel+0b0011)){

memcpy(&fsrStructL, &key4DataL, sizeof(key4DataL));

memcpy(&fsrStructR, &key4DataR, sizeof(key4DataR));

}

else if(i == (startingMuxSel+0b0100)){

memcpy(&fsrStructL, &key5DataL, sizeof(key5DataL));

memcpy(&fsrStructR, &key5DataR, sizeof(key5DataR));

}

else if(i == (startingMuxSel+0b0101)){

memcpy(&fsrStructL, &key6DataL, sizeof(key6DataL));

memcpy(&fsrStructR, &key6DataR, sizeof(key6DataR));

}

else if(i == (startingMuxSel+0b0110)){

memcpy(&fsrStructL, &key7DataL, sizeof(key7DataL));

memcpy(&fsrStructR, &key7DataR, sizeof(key7DataR));

}

}

// Function copies all values from fsrStruct to key#Data

void copytoStruct(int i){

if(i == (startingMuxSel)){

memcpy(&key1DataL, &fsrStructL, sizeof(fsrStructL));

memcpy(&key1DataR, &fsrStructR, sizeof(fsrStructR));

}

else if(i == (startingMuxSel+0b0001)){

memcpy(&key2DataL, &fsrStructL, sizeof(fsrStructL));

memcpy(&key2DataR, &fsrStructR, sizeof(fsrStructR));

}

else if(i == (startingMuxSel+0b0010)){

memcpy(&key3DataL, &fsrStructL, sizeof(fsrStructL));

memcpy(&key3DataR, &fsrStructR, sizeof(fsrStructR));

}

else if(i == (startingMuxSel+0b0011)){

memcpy(&key4DataL, &fsrStructL, sizeof(fsrStructL));

memcpy(&key4DataR, &fsrStructR, sizeof(fsrStructR));

}

else if(i == (startingMuxSel+0b0100)){

memcpy(&key5DataL, &fsrStructL, sizeof(fsrStructL));

memcpy(&key5DataR, &fsrStructR, sizeof(fsrStructR));

}

else if(i == (startingMuxSel+0b0101)){

memcpy(&key6DataL, &fsrStructL, sizeof(fsrStructL));

memcpy(&key6DataR, &fsrStructR, sizeof(fsrStructR));

}

else if(i == (startingMuxSel+0b0110)){

memcpy(&key7DataL, &fsrStructL, sizeof(fsrStructL));

memcpy(&key7DataR, &fsrStructR, sizeof(fsrStructR));

}

}

// Piano: Function to determine appropriate notes keypad should produce dependent on value of rotary encoder

void setPianoNotes(int8\_t tiltSensor, char whichKeypad){

int16\_t rE\_L = dataSigReceivedL.rotEncoder;

int16\_t rE\_R = dataSigReceivedR.rotEncoder;

if(whichKeypad == 'L'){

if(rE\_L <= maxReRange){

//set note for fsr6 to C and fsr0 to F

noteAddr1L = {MIDI\_Notes::E(2), Channel\_1};

noteAddr2L = {41, Channel\_1};

noteAddr5L = {MIDI\_Notes::G(2), Channel\_1};

noteAddr7L = {MIDI\_Notes::A(2), Channel\_1};

noteAddr3L = {MIDI\_Notes::Eb(2), Channel\_1};

noteAddr4L = {MIDI\_Notes::Gb(2), Channel\_1};

noteAddr6L = {MIDI\_Notes::Ab(2), Channel\_1};

//Serial.println("Zone 1");

}

else if(rE\_L >= maxReRange+1 && rE\_L <= 2\*maxReRange){

//set note for fsr6 to C and fsr0 to F

noteAddr1L = {MIDI\_Notes::B(2), Channel\_1};

noteAddr2L = {MIDI\_Notes::C(3), Channel\_1};

noteAddr5L = {MIDI\_Notes::D(3), Channel\_1};

noteAddr7L = {MIDI\_Notes::E(3), Channel\_1};

noteAddr3L = {MIDI\_Notes::Bb(2), Channel\_1};

noteAddr4L = {MIDI\_Notes::Db(3), Channel\_1};

noteAddr6L = {MIDI\_Notes::Eb(3), Channel\_1};

//Serial.println("Zone 2");

}

else if(rE\_L >= 2\*maxReRange+1 && rE\_L <= 3\*maxReRange){

//set note for fsr6 to C and fsr0 to F

//MIDI\_Notes::note ( int8\_t note,int8\_t numOctave ) ;

noteAddr1L = {53, Channel\_1};

//noteAddr1L = {53, Channel\_1};

noteAddr2L = {MIDI\_Notes::G(3), Channel\_1};

noteAddr5L = {MIDI\_Notes::A(3), Channel\_1};

noteAddr7L = {MIDI\_Notes::B(3), Channel\_1};

noteAddr3L = {MIDI\_Notes::Gb(3), Channel\_1};

noteAddr4L = {MIDI\_Notes::Ab(3), Channel\_1};

noteAddr6L = {MIDI\_Notes::Bb(3), Channel\_1};

//Serial.println("Zone 3");

}

else if(rE\_L >= 3\*maxReRange+1 && rE\_L <= 4\*maxReRange){

//set note for fsr6 to C and fsr0 to F

noteAddr1L = {MIDI\_Notes::C(4), Channel\_1};

noteAddr2L = {MIDI\_Notes::D(4), Channel\_1};

noteAddr5L = {MIDI\_Notes::E(4), Channel\_1};

noteAddr7L = {65, Channel\_1};

noteAddr3L = {MIDI\_Notes::Db(4), Channel\_1};

noteAddr4L = {MIDI\_Notes::Eb(4), Channel\_1};

noteAddr6L = {MIDI\_Notes::Gb(4), Channel\_1};

//Serial.println("Zone 4");

}

else if(rE\_L >= 4\*maxReRange+1 && rE\_L <= 5\*maxReRange){

noteAddr1L = {MIDI\_Notes::G(4), Channel\_1};

noteAddr2L = {MIDI\_Notes::A(4), Channel\_1};

noteAddr5L = {MIDI\_Notes::B(4), Channel\_1};

noteAddr7L = {MIDI\_Notes::C(5), Channel\_1};

noteAddr3L = {MIDI\_Notes::Ab(4), Channel\_1};

noteAddr4L = {MIDI\_Notes::Bb(4), Channel\_1};

noteAddr6L = {MIDI\_Notes::Db(5), Channel\_1};

//Serial.println("Zone 5");

}

else if(rE\_L >= 5\*maxReRange+1){

noteAddr1L = {MIDI\_Notes::D(5), Channel\_1};

noteAddr2L = {MIDI\_Notes::E(5), Channel\_1};

noteAddr5L = {77, Channel\_1};

noteAddr7L = {MIDI\_Notes::G(5), Channel\_1};

noteAddr3L = {MIDI\_Notes::Eb(5), Channel\_1};

noteAddr4L = {MIDI\_Notes::Gb(5), Channel\_1};

noteAddr6L = {MIDI\_Notes::Ab(5), Channel\_1};

//Serial.println("Zone 6");

}

}

else{

if(rE\_R <= maxReRange){

noteAddr1R = {MIDI\_Notes::D(5), Channel\_1};

noteAddr3R = {MIDI\_Notes::E(5), Channel\_1};

noteAddr5R = {77, Channel\_1};

noteAddr7R = {MIDI\_Notes::G(5), Channel\_1};

noteAddr2R = {MIDI\_Notes::Eb(5), Channel\_1};

noteAddr4R = {MIDI\_Notes::Gb(5), Channel\_1};

noteAddr6R = {MIDI\_Notes::Ab(5), Channel\_1};

//Serial.println("R: Zone 6");

}

else if(rE\_R >= maxReRange+1 && rE\_R <= 2\*maxReRange){

noteAddr1R = {MIDI\_Notes::G(4), Channel\_1};

noteAddr3R = {MIDI\_Notes::A(4), Channel\_1};

noteAddr5R = {MIDI\_Notes::B(4), Channel\_1};

noteAddr7R = {MIDI\_Notes::C(5), Channel\_1};

noteAddr2R = {MIDI\_Notes::Ab(4), Channel\_1};

noteAddr4R = {MIDI\_Notes::Bb(4), Channel\_1};

noteAddr6R = {MIDI\_Notes::Db(5), Channel\_1};

//Serial.println("R: Zone 5");

}

else if(rE\_R >= 2\*maxReRange+1 && rE\_R <= 3\*maxReRange){

//set note for fsr6 to C and fsr0 to F

noteAddr1R = {MIDI\_Notes::C(4), Channel\_1};

noteAddr3R = {MIDI\_Notes::D(4), Channel\_1};

noteAddr5R = {MIDI\_Notes::E(4), Channel\_1};

noteAddr7R = {65, Channel\_1};

noteAddr2R = {MIDI\_Notes::Db(4), Channel\_1};

noteAddr4R = {MIDI\_Notes::Eb(4), Channel\_1};

noteAddr6R = {MIDI\_Notes::Gb(4), Channel\_1};

//Serial.println("R: Zone 4");

}

else if(rE\_R >= 3\*maxReRange+1 && rE\_R <= 4\*maxReRange){

noteAddr1R = {53, Channel\_1};

noteAddr3R = {MIDI\_Notes::G(3), Channel\_1};

noteAddr5R = {MIDI\_Notes::A(3), Channel\_1};

noteAddr7R = {MIDI\_Notes::B(3), Channel\_1};

noteAddr2R = {MIDI\_Notes::Gb(3), Channel\_1};

noteAddr4R = {MIDI\_Notes::Ab(3), Channel\_1};

noteAddr6R = {MIDI\_Notes::Bb(3), Channel\_1};

//Serial.println("R: Zone 3");

}

else if(rE\_R >= 4\*maxReRange+1 && rE\_R <= 5\*maxReRange){

//set note for fsr6 to C and fsr0 to F

noteAddr1R = {MIDI\_Notes::B(2), Channel\_1};

noteAddr3R = {MIDI\_Notes::C(3), Channel\_1};

noteAddr5R = {MIDI\_Notes::D(3), Channel\_1};

noteAddr7R = {MIDI\_Notes::E(3), Channel\_1};

noteAddr2R = {MIDI\_Notes::Bb(2), Channel\_1};

noteAddr4R = {MIDI\_Notes::Db(3), Channel\_1};

noteAddr6R = {MIDI\_Notes::Eb(3), Channel\_1};

//Serial.println("R: Zone 2");

}

else if(rE\_R >= 5\*maxReRange+1){

//set note for fsr6 to C and fsr0 to F

noteAddr1R = {MIDI\_Notes::E(2), Channel\_1};

noteAddr3R = {41, Channel\_1};

noteAddr5R = {MIDI\_Notes::G(2), Channel\_1};

noteAddr7R = {MIDI\_Notes::A(2), Channel\_1};

noteAddr2R = {MIDI\_Notes::Eb(2), Channel\_1};

noteAddr4R= {MIDI\_Notes::Gb(2), Channel\_1};

noteAddr6R = {MIDI\_Notes::Ab(2), Channel\_1};

//Serial.println("R: Zone 1");

}

}

}

// Piano: Function turns piano on and off

void pianoMode(int i){

if(fsrStructL.prevVolume <= 24 && fsrStructL.currVolume <= 24 && fsrStructL.turnOffFirstTime == 1)

{

// make sure to only turn the previous note off

if(noteOnSig.ith\_Val[i] == 1 && noteOnSig.whichKeypad[i] == 'L'){

chooseNoteOff(i,'L');

noteOnSig.ith\_Val[i] = 0;

noteOnSig.whichKeypad[i] = 'N';

}

fsrStructL.turnOffFirstTime = 0;

fsrStructL.firstTimeNoteOn = 1;

}

else if(fsrStructL.currVolume >= 25 && fsrStructL.firstTimeNoteOn == 1){ //NOTE SHOULD ONLY GO OFF ONCE WHICH IS AT INITIAL TIME OF PRESS, after that it will never go here

fadeVolume(i, 'L');

fsrStructL.turnOffFirstTime = 1;

fsrStructL.firstTimeNoteOn = 0;

}

else{

//Serial.println("L: Do Nothing");

}

if(fsrStructR.prevVolume <= 24 && fsrStructR.currVolume <= 24 && fsrStructR.turnOffFirstTime == 1)

{

// make sure to only turn the previous note off

if(noteOnSig.ith\_Val[i] == 1 && noteOnSig.whichKeypad[i] == 'R'){

chooseNoteOff(i, 'R');

noteOnSig.ith\_Val[i] = 0;

noteOnSig.whichKeypad[i] = 'N';

}

fsrStructR.turnOffFirstTime = 0;

fsrStructR.firstTimeNoteOn = 1;

}

else if(fsrStructR.currVolume >= 25 && fsrStructR.firstTimeNoteOn == 1){ //NOTE SHOULD ONLY GO OFF ONCE WHICH IS AT INITIAL TIME OF PRESS, after that it will never go here

fadeVolume(i, 'R');

fsrStructR.turnOffFirstTime = 1;

fsrStructR.firstTimeNoteOn = 0;

}

else{

//Serial.println("R: Do Nothing");

}

}

// Piano: Function to set initial volume: if prev volume off, curr Volume on

void initializeVol(int channel){

if(channel == 1 || channel == 6 || channel == 5){

// Set Initial Volume

if(fsrStructL.prevVolume <= 24 && fsrStructL.currVolume >= 25){

fsrStructL.initVolume = fsrStructL.currVolume;

}

if(fsrStructR.prevVolume <= 24 && fsrStructR.currVolume >= 25){

fsrStructR.initVolume = fsrStructR.currVolume;

}

}

else if(channel == 2 || channel == 3 || channel == 4){//dataSigReceivedR.joystick >= 760 || dataSigReceivedR.joystick <= 280

if((fsrStructR.prevVolume <= JOYSTICKUPPERLIMIT && fsrStructR.prevVolume >= JOYSTICKLOWERLIMIT) && (fsrStructR.currVolume >= JOYSTICKUPPERLIMIT || fsrStructR.currVolume <= JOYSTICKLOWERLIMIT)){

fsrStructR.initVolume = fsrStructR.currVolume;

}

else{//initial vol stays exactly the same

}

}

}

// Piano: Function maps the values received to the volume range

int mapFSRtoVolume(int i, char whichKeypad){

//maps fsrValue(0-420) to somewhere in the volume range (0-127)

// constrain ensures that the value derived from map falls within the min and max which is 0 and 127

if(i == (startingMuxSel)){ //maps to mux channel 0(C0)

if(whichKeypad == 'L'){

return constrain(map(dataSigReceivedL.fsr0, 10, 290, 20, 127), 20, 127);

}

else{

return constrain(map(dataSigReceivedR.fsr0, 10, 250, 20, 127), 20, 127);

}

}

else if(i == (startingMuxSel+0b0001)){

if(whichKeypad == 'L'){

return constrain(map(dataSigReceivedL.fsr1, 10, 150, 20, 127), 20, 127);

}

else{

return constrain(map(dataSigReceivedR.fsr1, 10, 150, 20, 127), 20, 127);

}

}

else if(i == (startingMuxSel+0b0010)){

if(whichKeypad == 'L'){

return constrain(map(dataSigReceivedL.fsr2, 10, 200, 20, 127), 20, 127);

}

else{

return constrain(map(dataSigReceivedR.fsr2, 10, 300, 20, 127), 20, 127);

}

}

else if(i == (startingMuxSel+0b0011)){

if(whichKeypad == 'L'){

return constrain(map(dataSigReceivedL.fsr3, 10, 360, 20, 127), 20, 127);

}

else{

return constrain(map(dataSigReceivedR.fsr3, 10, 300, 20, 127), 20, 127);

}

}

else if(i == (startingMuxSel+0b0100)){

if(whichKeypad == 'L'){

return constrain(map(dataSigReceivedL.fsr4, 10, 90, 20, 127), 20, 127);

}

else{

return constrain(map(dataSigReceivedR.fsr4, 10, 200, 20, 127), 20, 127);

}

}

else if(i == (startingMuxSel+0b0101)){

if(whichKeypad == 'L'){

return constrain(map(dataSigReceivedL.fsr5, 10, 120, 20, 127), 20, 127);

}

else{

return constrain(map(dataSigReceivedR.fsr5, 10, 300, 20, 127), 20, 127);

}

}

else if(i == (startingMuxSel+0b0110)){

if(whichKeypad == 'L'){

return constrain(map(dataSigReceivedL.fsr6, 10, 140, 20, 127), 20, 127);

}

else{

return constrain(map(dataSigReceivedR.fsr6, 10, 200, 20, 127), 20, 127);

}

}

return 0;

}

// Sax: Function determines which note to turn on when in sax mode

void saxMode(){

bool keys[14];

keys[0] = (dataSigReceivedR.fsr0 >= 5) ? 1 : 0; //right keypad

keys[1] = (dataSigReceivedR.fsr2 >= 5) ? 1 : 0;

keys[2] = (dataSigReceivedR.fsr4 >= 5) ? 1 : 0;

keys[3] = (dataSigReceivedR.fsr6 >= 5) ? 1 : 0;

keys[4] = (dataSigReceivedR.fsr1 >= 5) ? 1 : 0;

keys[5] = (dataSigReceivedR.fsr3 >= 5) ? 1 : 0;

keys[6] = (dataSigReceivedR.fsr5 >= 5) ? 1 : 0;

keys[7] = (dataSigReceivedL.fsr0 >= 5) ? 1 : 0; //left keypad

keys[8] = (dataSigReceivedL.fsr1 >= 5) ? 1 : 0;

keys[9] = (dataSigReceivedL.fsr4 >= 5) ? 1 : 0;

keys[10] = (dataSigReceivedL.fsr6 >= 5) ? 1 : 0;

keys[11] = (dataSigReceivedL.fsr2 >= 5) ? 1 : 0;

keys[12] = (dataSigReceivedL.fsr3 >= 5) ? 1 : 0;

keys[13] = (dataSigReceivedL.fsr5 >= 5) ? 1 : 0;

Serial.println("Right Keys[]: "+String(keys[0])+","+String(keys[1])+","+String(keys[2])+","+String(keys[3])+","+String(keys[4])+","+String(keys[5])+","+String(keys[6]));

Serial.println("Left Keys[]: "+String(keys[7])+","+String(keys[8])+","+String(keys[9])+","+String(keys[10])+","+String(keys[11])+","+String(keys[12])+","+String(keys[13]));

//Serial.println("if condition result: "+String(keys[3] && keys[2] && keys[1] && keys[7] && keys[8] && keys[9]));

if(keys[3] && keys[2] && keys[1] && keys[7] && keys[8] && keys[9]){

//Bb3 to Eb4, D5 and Eb5

if(keys[13]){

//Bb3 to Db4

if(keys[0]){

saxNoteAddr = {MIDI\_Notes::Bb(3), Channel\_2};

}

else if(keys[4]){

saxNoteAddr = {MIDI\_Notes::Db(4), Channel\_2};

}

else{

saxNoteAddr = {MIDI\_Notes::C(4), Channel\_2};

}

}

else if(keys[10]){

//Eb4, Eb5

if(!keys[6]){

saxNoteAddr = {MIDI\_Notes::Eb(4), Channel\_2};

}

else{

saxNoteAddr = {MIDI\_Notes::Eb(5), Channel\_2};

}

}

else{

//D4,D5

if(!keys[6]){

saxNoteAddr = {MIDI\_Notes::D(4), Channel\_2};

}

else{

saxNoteAddr = {MIDI\_Notes::D(5), Channel\_2};

}

}

}

else if(keys[3] && keys[2] && keys[1] && keys[7] && keys[8] && !keys[9] && !keys[0] && !keys[10]){

//E4,E5

if(!keys[6]){

saxNoteAddr = {MIDI\_Notes::E(4), Channel\_2};

}

else{

saxNoteAddr = {MIDI\_Notes::E(5), Channel\_2};

}

}

else if(keys[3] && keys[2] && keys[1] && keys[7] && !keys[8] && !keys[9] ){

//F4,F5

if(!keys[6]){

saxNoteAddr = {65, Channel\_2};

}

else{

saxNoteAddr = {77, Channel\_2};

}

}

else if(keys[3] && keys[2] && keys[1] && !keys[7] && keys[8] && !keys[9]){

//Gb4,Gb5

if(!keys[6]){

saxNoteAddr = {MIDI\_Notes::Gb(4), Channel\_2};

}

else{

saxNoteAddr = {MIDI\_Notes::Gb(5), Channel\_2};

}

}

else if(keys[3] && keys[2] && keys[1] && !keys[7] && !keys[8] && !keys[9]){

//G4,Ab4,G5,Ab5

if(keys[0]){

if(keys[6]){

saxNoteAddr = {MIDI\_Notes::Ab(5), Channel\_2};

}

else{

saxNoteAddr = {MIDI\_Notes::Ab(4), Channel\_2};

}

}

else{

if(keys[6]){

saxNoteAddr = {MIDI\_Notes::G(5), Channel\_2};

}

else{

saxNoteAddr = {MIDI\_Notes::G(4), Channel\_2};

}

}

}

else if(keys[3] && keys[2] && !keys[1] && !keys[7] && !keys[8] && !keys[9]){

//A4,A5

if(!keys[6]){

saxNoteAddr = {MIDI\_Notes::A(4), Channel\_2};

}

else{

saxNoteAddr = {MIDI\_Notes::A(5), Channel\_2};

}

}

else if(keys[3] && !keys[2] && !keys[1] && !keys[7] && !keys[8] && !keys[9]){

//B4,B5

if(keys[6]){

if(keys[5]){

saxNoteAddr = {MIDI\_Notes::Bb(5), Channel\_2};

}

else{

saxNoteAddr = {MIDI\_Notes::B(5), Channel\_2};

}

}

else{

saxNoteAddr = {MIDI\_Notes::B(4), Channel\_2};

}

}

else if(!keys[3] && keys[2] && !keys[1] && !keys[7] && !keys[8] && !keys[9]){

//C5,C6

if(!keys[6]){

saxNoteAddr = {MIDI\_Notes::C(5), Channel\_2};

}

else{

saxNoteAddr = {MIDI\_Notes::C(6), Channel\_2};

}

}

else if(!keys[3] && !keys[2] && !keys[1] && !keys[7] && !keys[8] && !keys[9]){

//Db5,Db6

if(!keys[6]){

saxNoteAddr = {MIDI\_Notes::Db(5), Channel\_2};

}

else{

saxNoteAddr = {MIDI\_Notes::Db(6), Channel\_2};

}

}

//call function to send notes on/off

saxSendNotes();

}

// Sax: Function signals synthesizer to turn sax note w/ variable volume on when right joystick is pressed

void saxSendNotes(){

bool turnOff = ((fsrStructR.prevVolume <= JOYSTICKUPPERLIMIT && fsrStructR.prevVolume >= JOYSTICKLOWERLIMIT)&&(fsrStructR.currVolume <= JOYSTICKUPPERLIMIT && fsrStructR.currVolume >= JOYSTICKLOWERLIMIT)) ? 1:0;

bool turnOn = (fsrStructR.currVolume >= JOYSTICKUPPERLIMIT || fsrStructR.currVolume <= JOYSTICKLOWERLIMIT) ? 1:0;

if(turnOff && fsrStructR.turnOffFirstTime == 1){

//turn note off

midi.sendNoteOff(saxNoteAddr, velocity);

fsrStructR.turnOffFirstTime = 0;

fsrStructR.firstTimeNoteOn = 1;

}

else if(turnOn && fsrStructR.firstTimeNoteOn == 1){

int16\_t mapped\_joystick = 0;

if(dataSigReceivedR.joystick >= JOYSTICKUPPERLIMIT){

mapped\_joystick = constrain(map(dataSigReceivedR.joystick, JOYSTICKUPPERLIMIT, 1023, 20, 127), 20, 127);

}

else{

mapped\_joystick = constrain(map(dataSigReceivedR.joystick, JOYSTICKLOWERLIMIT, 0, 20, 127), 20, 127);

}

//Serial.println("Mapped Joystick: "+String(mapped\_joystick));

midi.sendControlChange(ccAddr, mapped\_joystick); //control volume then send note on

// Send pitch bend message

int pitchBendValue = map(dataSigReceivedL.joystick, 0, 1023, 0, 16383);

pitchBendValue = constrain(pitchBendValue, 0, 16383); // Ensure the value stays within bounds

midi.sendPitchBend(Channel\_2, pitchBendValue);

midi.sendNoteOn(saxNoteAddr, velocity);

fsrStructR.turnOffFirstTime = 1;

fsrStructR.firstTimeNoteOn = 0;

}

}

// Clarinet: Function determines which note to turn on when in clarinet mode

void clarinetMode(){

bool keys[14];

keys[0] = (dataSigReceivedR.fsr0 >= 5) ? 1 : 0; //right keypad

keys[1] = (dataSigReceivedR.fsr2 >= 5) ? 1 : 0;

keys[2] = (dataSigReceivedR.fsr4 >= 5) ? 1 : 0;

keys[3] = (dataSigReceivedR.fsr6 >= 5) ? 1 : 0;

keys[4] = (dataSigReceivedR.fsr1 >= 5) ? 1 : 0;

keys[5] = (dataSigReceivedR.fsr3 >= 5) ? 1 : 0;

keys[6] = (dataSigReceivedR.fsr5 >= 5) ? 1 : 0;

keys[7] = (dataSigReceivedL.fsr0 >= 5) ? 1 : 0; //left keypad

keys[8] = (dataSigReceivedL.fsr1 >= 5) ? 1 : 0;

keys[9] = (dataSigReceivedL.fsr4 >= 5) ? 1 : 0;

keys[10] = (dataSigReceivedL.fsr6 >= 5) ? 1 : 0;

keys[11] = (dataSigReceivedL.fsr2 >= 5) ? 1 : 0;

keys[12] = (dataSigReceivedL.fsr3 >= 5) ? 1 : 0;

keys[13] = (dataSigReceivedL.fsr5 >= 5) ? 1 : 0;

Serial.println("Right Keys[]: "+String(keys[0])+","+String(keys[1])+","+String(keys[2])+","+String(keys[3])+","+String(keys[4])+","+String(keys[5])+","+String(keys[6]));

Serial.println("Left Keys[]: "+String(keys[7])+","+String(keys[8])+","+String(keys[9])+","+String(keys[10])+","+String(keys[11])+","+String(keys[12])+","+String(keys[13]));

// if statements to change clarinetNoteAddr

if(keys[3] && keys[2] && keys[1] && keys[7] && keys[8] && keys[9]){

if(!keys[10] && !keys[11] && !keys[12] && !keys[13] && !keys[5] && !keys[6] && keys[0]){

//G3

clarinetNoteAddr = {MIDI\_Notes::G(3), Channel\_3};

}

else if(!keys[10] && !keys[11] && keys[12] && !keys[13] && !keys[5] && !keys[6] && keys[0]){

//E3

clarinetNoteAddr = {MIDI\_Notes::E(3), Channel\_3};

}

else if(!keys[10] && !keys[11] && !keys[12] && keys[13] && !keys[5] && !keys[6] && keys[0]){

//F3

clarinetNoteAddr = {53, Channel\_3};

}

else if(!keys[10] && keys[11] && !keys[12] && !keys[13] && !keys[5] && !keys[6] && keys[0]){

//Gb3

clarinetNoteAddr = {MIDI\_Notes::Gb(3), Channel\_3};

}

else if(!keys[10] && !keys[11] && keys[12] && !keys[13] && keys[5] && !keys[6] && keys[0]){

//B4

clarinetNoteAddr = {MIDI\_Notes::B(4), Channel\_3};

}

else if(!keys[10] && !keys[11] && !keys[12] && keys[13] && keys[5] && !keys[6] && keys[0]){

//C5

clarinetNoteAddr = {MIDI\_Notes::C(5), Channel\_3};

}

else if(keys[10] && !keys[11] && !keys[12] && !keys[13] && keys[5] && !keys[6] && keys[0]){

//Db5

clarinetNoteAddr = {MIDI\_Notes::Db(5), Channel\_3};

}

else if(!keys[10] && !keys[11] && !keys[12] && !keys[13] && keys[5] && !keys[6] && keys[0]){

//D5

clarinetNoteAddr = {MIDI\_Notes::D(5), Channel\_3};

}

else if(!keys[10] && keys[11] && !keys[12] && !keys[13] && keys[5] && !keys[6] && keys[0]){

//Eb5

clarinetNoteAddr = {MIDI\_Notes::Eb(5), Channel\_3};

}

}

else if(keys[3] && keys[2] && keys[1] && keys[7] && keys[8] && !keys[9]){

if(!keys[10] && !keys[11] && !keys[12] && !keys[13] && !keys[5] && !keys[6] && keys[0]){

//A3

clarinetNoteAddr = {MIDI\_Notes::A(3), Channel\_3};

}

else if(!keys[10] && !keys[11] && !keys[12] && !keys[13] && keys[5] && !keys[6] && keys[0]){

//E5

clarinetNoteAddr = {MIDI\_Notes::E(5), Channel\_3};

}

}

else if(keys[3] && keys[2] && keys[1] && keys[7] && !keys[8] && !keys[9]){

if(!keys[10] && !keys[11] && !keys[12] && !keys[13] && !keys[5] && !keys[6] && keys[0]){

//Bb3

clarinetNoteAddr = {MIDI\_Notes::Bb(3), Channel\_3};

}

else if(!keys[10] && !keys[11] && !keys[12] && !keys[13] && keys[5] && !keys[6] && keys[0]){

//F5

clarinetNoteAddr = {77, Channel\_3};

}

}

else if(keys[3] && keys[2] && keys[1] && !keys[7] && keys[8] && !keys[9]){

if(!keys[10] && !keys[11] && !keys[12] && !keys[13] && !keys[5] && !keys[6] && keys[0]){

//B3

clarinetNoteAddr = {MIDI\_Notes::B(3), Channel\_3};

}

else if(!keys[10] && !keys[11] && !keys[12] && !keys[13] && keys[5] && !keys[6] && keys[0]){

//Gb5

clarinetNoteAddr = {MIDI\_Notes::Gb(5), Channel\_3};

}

}

else if(keys[3] && keys[2] && keys[1] && !keys[7] && !keys[8] && !keys[9]){

if(!keys[10] && !keys[11] && !keys[12] && !keys[13] && !keys[5] && !keys[6] && keys[0]){

//C4

clarinetNoteAddr = {MIDI\_Notes::C(4), Channel\_3};

}

else if(!keys[10] && !keys[11] && !keys[12] && !keys[13] && keys[5] && !keys[6] && keys[0]){

//G5

clarinetNoteAddr = {MIDI\_Notes::G(5), Channel\_3};

}

}

else if(keys[3] && keys[2] && !keys[1] && !keys[7] && !keys[8] && !keys[9]){

if(!keys[10] && !keys[11] && !keys[12] && !keys[13] && !keys[5] && !keys[6] && keys[0]){

//D4

clarinetNoteAddr = {MIDI\_Notes::D(4), Channel\_3};

}

else if(!keys[10] && !keys[11] && !keys[12] && !keys[13] && keys[5] && !keys[6] && keys[0]){

//A5

clarinetNoteAddr = {MIDI\_Notes::A(5), Channel\_3};

}

}

else if(keys[3] && !keys[2] && !keys[1]){

if(!keys[7] && !keys[8] && !keys[9] && !keys[10] && !keys[11] && !keys[12] && !keys[13] && !keys[5] && !keys[6] && keys[0]){

//E4

clarinetNoteAddr = {MIDI\_Notes::E(4), Channel\_3};

}

else if(keys[7] && !keys[8] && !keys[9] && !keys[10] && !keys[11] && !keys[12] && !keys[13] && !keys[5] && !keys[6] && keys[0]){

//Eb4

clarinetNoteAddr = {MIDI\_Notes::Eb(4), Channel\_3};

}

else if(!keys[7] && !keys[8] && !keys[9] && !keys[10] && !keys[11] && !keys[12] && !keys[13] && !keys[5] && !keys[6] && !keys[0]){

//Gb4

clarinetNoteAddr = {MIDI\_Notes::Gb(4), Channel\_3};

}

else if(keys[7] && !keys[8] && !keys[9] && !keys[10] && !keys[11] && !keys[12] && !keys[13] && keys[5] && !keys[6] && keys[0]){

//Bb5

clarinetNoteAddr = {MIDI\_Notes::Bb(5), Channel\_3};

}

else if(!keys[7] && !keys[8] && !keys[9] && !keys[10] && !keys[11] && !keys[12] && !keys[13] && keys[5] && !keys[6] && keys[0]){

//B5

clarinetNoteAddr = {MIDI\_Notes::B(5), Channel\_3};

}

}

else if(!keys[3] && !keys[2] && !keys[1] && !keys[7] && !keys[8] && !keys[9]){

if(!keys[10] && !keys[11] && !keys[12] && !keys[13] && !keys[5] && keys[6] && !keys[0]){

//A4

clarinetNoteAddr = {MIDI\_Notes::A(4), Channel\_3};

}

else if(!keys[10] && !keys[11] && !keys[12] && !keys[13] && keys[5] && keys[6] && !keys[0]){

//Bb4

clarinetNoteAddr = {MIDI\_Notes::Bb(4), Channel\_3};

}

else if(!keys[10] && !keys[11] && !keys[12] && !keys[13] && !keys[5] && !keys[6] && keys[0]){

//F4

clarinetNoteAddr = {65, Channel\_3};

}

else if(!keys[10] && !keys[11] && !keys[12] && !keys[13] && keys[5] && !keys[6] && keys[0]){

//C6

clarinetNoteAddr = {MIDI\_Notes::C(6), Channel\_3};

}

}

else if(!keys[3] && keys[2] && keys[1] && keys[7] && keys[8] && !keys[9]){

if(!keys[10] && !keys[11] && !keys[12] && !keys[13] && keys[5] && !keys[6] && keys[0]){

//Db6

clarinetNoteAddr = {MIDI\_Notes::Db(6), Channel\_3};

}

}

else if(!keys[3] && keys[2] && keys[1] && keys[7] && !keys[8] && !keys[9]){

//D6

clarinetNoteAddr = {MIDI\_Notes::D(6), Channel\_3};

}

else if(!keys[3] && keys[2] && keys[1] && !keys[7] && !keys[8] && !keys[9]){

//E6

clarinetNoteAddr = {MIDI\_Notes::E(6), Channel\_3};

}

else if(!keys[3] && !keys[2] && !keys[1] && !keys[7] && !keys[8] && !keys[9]){

//G4

clarinetNoteAddr = {MIDI\_Notes::G(4), Channel\_3};

}

//call function to send notes on/off

clarinetSendNotes();

}

// Clarinet: Function signals synthesizer to turn clarinet note w/ variable volume on when right joystick is pressed

void clarinetSendNotes(){

bool turnOff = ((fsrStructR.prevVolume <= JOYSTICKUPPERLIMIT && fsrStructR.prevVolume >= JOYSTICKLOWERLIMIT)&&(fsrStructR.currVolume <= JOYSTICKUPPERLIMIT && fsrStructR.currVolume >= JOYSTICKLOWERLIMIT)) ? 1:0;

bool turnOn = (fsrStructR.currVolume >= JOYSTICKUPPERLIMIT || fsrStructR.currVolume <= JOYSTICKLOWERLIMIT) ? 1:0;

if(turnOff && fsrStructR.turnOffFirstTime == 1){

//turn note off

midi.sendNoteOff(clarinetNoteAddr, velocity);

fsrStructR.turnOffFirstTime = 0;

fsrStructR.firstTimeNoteOn = 1;

}

else if(turnOn && fsrStructR.firstTimeNoteOn == 1){

int16\_t mapped\_joystick = 0;

if(dataSigReceivedR.joystick >= JOYSTICKUPPERLIMIT){

mapped\_joystick = constrain(map(dataSigReceivedR.joystick, JOYSTICKUPPERLIMIT, 1023, 20, 127), 20, 127);

}

else{

mapped\_joystick = constrain(map(dataSigReceivedR.joystick, JOYSTICKLOWERLIMIT, 0, 20, 127), 20, 127);

}

Serial.println("Mapped Joystick: "+String(mapped\_joystick));

midi.sendControlChange(ccAddr, mapped\_joystick); //control volume then send note on

// Send pitch bend message

int pitchBendValue = map(dataSigReceivedL.joystick, 0, 1023, 0, 16383);

pitchBendValue = constrain(pitchBendValue, 0, 16383); // Ensure the value stays within bounds

midi.sendPitchBend(Channel\_3, pitchBendValue);

midi.sendNoteOn(clarinetNoteAddr, velocity);

fsrStructR.turnOffFirstTime = 1;

fsrStructR.firstTimeNoteOn = 0;

}

}

// Flute: Function determines which note to turn on when in flute mode

void fluteMode(){

bool keys[14];

keys[0] = (dataSigReceivedR.fsr0 >= 5) ? 1 : 0; //right keypad

keys[1] = (dataSigReceivedR.fsr2 >= 5) ? 1 : 0;

keys[2] = (dataSigReceivedR.fsr4 >= 5) ? 1 : 0;

keys[3] = (dataSigReceivedR.fsr6 >= 5) ? 1 : 0;

keys[4] = (dataSigReceivedR.fsr1 >= 5) ? 1 : 0;

keys[5] = (dataSigReceivedR.fsr3 >= 5) ? 1 : 0;

keys[6] = (dataSigReceivedR.fsr5 >= 5) ? 1 : 0;

keys[7] = (dataSigReceivedL.fsr0 >= 5) ? 1 : 0; //left keypad

keys[8] = (dataSigReceivedL.fsr1 >= 5) ? 1 : 0;

keys[9] = (dataSigReceivedL.fsr4 >= 5) ? 1 : 0;

keys[10] = (dataSigReceivedL.fsr6 >= 5) ? 1 : 0;

keys[11] = (dataSigReceivedL.fsr2 >= 5) ? 1 : 0;

keys[12] = (dataSigReceivedL.fsr3 >= 5) ? 1 : 0;

keys[13] = (dataSigReceivedL.fsr5 >= 5) ? 1 : 0;

Serial.println("Left Keys[]: "+String(keys[7])+","+String(keys[8])+","+String(keys[9])+","+String(keys[10])+","+String(keys[11])+","+String(keys[12])+","+String(keys[13]));

Serial.println("Right Keys[]: "+String(keys[0])+","+String(keys[1])+","+String(keys[2])+","+String(keys[3])+","+String(keys[4])+","+String(keys[5])+","+String(keys[6]));

// if statements to change fluteNoteAddr

if(keys[3] && keys[2] && keys[1] && keys[7] && keys[8] && keys[9]){

if(!keys[10] && !keys[11] && !keys[12] && !keys[13] && !keys[5] && !keys[6] && keys[0]){

//G3

fluteNoteAddr = {MIDI\_Notes::G(3), Channel\_4};

}

else if(!keys[10] && !keys[11] && keys[12] && !keys[13] && !keys[5] && !keys[6] && keys[0]){

//E3

fluteNoteAddr = {MIDI\_Notes::E(3), Channel\_4};

}

else if(!keys[10] && !keys[11] && !keys[12] && keys[13] && !keys[5] && !keys[6] && keys[0]){

//F3

fluteNoteAddr = {53, Channel\_4};

}

else if(!keys[10] && keys[11] && !keys[12] && !keys[13] && !keys[5] && !keys[6] && keys[0]){

//Gb3

fluteNoteAddr = {MIDI\_Notes::Gb(3), Channel\_4};

}

else if(!keys[10] && !keys[11] && keys[12] && !keys[13] && keys[5] && !keys[6] && keys[0]){

//B4

fluteNoteAddr = {MIDI\_Notes::B(4), Channel\_4};

}

else if(!keys[10] && !keys[11] && !keys[12] && keys[13] && keys[5] && !keys[6] && keys[0]){

//C5

fluteNoteAddr = {MIDI\_Notes::C(5), Channel\_4};

}

else if(keys[10] && !keys[11] && !keys[12] && !keys[13] && keys[5] && !keys[6] && keys[0]){

//Db5

fluteNoteAddr = {MIDI\_Notes::Db(5), Channel\_4};

}

else if(!keys[10] && !keys[11] && !keys[12] && !keys[13] && keys[5] && !keys[6] && keys[0]){

//D5

fluteNoteAddr = {MIDI\_Notes::D(5), Channel\_4};

}

else if(!keys[10] && keys[11] && !keys[12] && !keys[13] && keys[5] && !keys[6] && keys[0]){

//Eb5

fluteNoteAddr = {MIDI\_Notes::Eb(5), Channel\_4};

}

}

else if(keys[3] && keys[2] && keys[1] && keys[7] && keys[8] && !keys[9]){

if(!keys[10] && !keys[11] && !keys[12] && !keys[13] && !keys[5] && !keys[6] && keys[0]){

//A3

fluteNoteAddr = {MIDI\_Notes::A(3), Channel\_4};

}

else if(!keys[10] && !keys[11] && !keys[12] && !keys[13] && keys[5] && !keys[6] && keys[0]){

//E5

fluteNoteAddr = {MIDI\_Notes::E(5), Channel\_4};

}

}

else if(keys[3] && keys[2] && keys[1] && keys[7] && !keys[8] && !keys[9]){

if(!keys[10] && !keys[11] && !keys[12] && !keys[13] && !keys[5] && !keys[6] && keys[0]){

//Bb3

fluteNoteAddr = {MIDI\_Notes::Bb(3), Channel\_4};

}

else if(!keys[10] && !keys[11] && !keys[12] && !keys[13] && keys[5] && !keys[6] && keys[0]){

//F5

fluteNoteAddr = {77, Channel\_4};

}

}

else if(keys[3] && keys[2] && keys[1] && !keys[7] && keys[8] && !keys[9]){

if(!keys[10] && !keys[11] && !keys[12] && !keys[13] && !keys[5] && !keys[6] && keys[0]){

//B3

fluteNoteAddr = {MIDI\_Notes::B(3), Channel\_4};

}

else if(!keys[10] && !keys[11] && !keys[12] && !keys[13] && keys[5] && !keys[6] && keys[0]){

//Gb5

fluteNoteAddr = {MIDI\_Notes::Gb(5), Channel\_4};

}

}

else if(keys[3] && keys[2] && keys[1] && !keys[7] && !keys[8] && !keys[9]){

if(!keys[10] && !keys[11] && !keys[12] && !keys[13] && !keys[5] && !keys[6] && keys[0]){

//C4

fluteNoteAddr = {MIDI\_Notes::C(4), Channel\_4};

}

else if(!keys[10] && !keys[11] && !keys[12] && !keys[13] && keys[5] && !keys[6] && keys[0]){

//G5

fluteNoteAddr = {MIDI\_Notes::G(5), Channel\_4};

}

}

else if(keys[3] && keys[2] && !keys[1] && !keys[7] && !keys[8] && !keys[9]){

if(!keys[10] && !keys[11] && !keys[12] && !keys[13] && !keys[5] && !keys[6] && keys[0]){

//D4

fluteNoteAddr = {MIDI\_Notes::D(4), Channel\_4};

}

else if(!keys[10] && !keys[11] && !keys[12] && !keys[13] && keys[5] && !keys[6] && keys[0]){

//A5

fluteNoteAddr = {MIDI\_Notes::A(5), Channel\_4};

}

}

else if(keys[3] && !keys[2] && !keys[1]){

if(!keys[7] && !keys[8] && !keys[9] && !keys[10] && !keys[11] && !keys[12] && !keys[13] && !keys[5] && !keys[6] && keys[0]){

//E4

fluteNoteAddr = {MIDI\_Notes::E(4), Channel\_4};

}

else if(keys[7] && !keys[8] && !keys[9] && !keys[10] && !keys[11] && !keys[12] && !keys[13] && !keys[5] && !keys[6] && keys[0]){

//Eb4

fluteNoteAddr = {MIDI\_Notes::Eb(4), Channel\_4};

}

else if(!keys[7] && !keys[8] && !keys[9] && !keys[10] && !keys[11] && !keys[12] && !keys[13] && !keys[5] && !keys[6] && !keys[0]){

//Gb4

fluteNoteAddr = {MIDI\_Notes::Gb(4), Channel\_4};

}

else if(keys[7] && !keys[8] && !keys[9] && !keys[10] && !keys[11] && !keys[12] && !keys[13] && keys[5] && !keys[6] && keys[0]){

//Bb5

fluteNoteAddr = {MIDI\_Notes::Bb(5), Channel\_4};

}

else if(!keys[7] && !keys[8] && !keys[9] && !keys[10] && !keys[11] && !keys[12] && !keys[13] && keys[5] && !keys[6] && keys[0]){

//B5

fluteNoteAddr = {MIDI\_Notes::B(5), Channel\_4};

}

}

else if(!keys[3] && !keys[2] && !keys[1] && !keys[7] && !keys[8] && !keys[9]){

if(!keys[10] && !keys[11] && !keys[12] && !keys[13] && !keys[5] && keys[6] && !keys[0]){

//A4

fluteNoteAddr = {MIDI\_Notes::A(4), Channel\_4};

}

else if(!keys[10] && !keys[11] && !keys[12] && !keys[13] && keys[5] && keys[6] && !keys[0]){

//Bb4

fluteNoteAddr = {MIDI\_Notes::Bb(4), Channel\_4};

}

else if(!keys[10] && !keys[11] && !keys[12] && !keys[13] && !keys[5] && !keys[6] && keys[0]){

//F4

fluteNoteAddr = {65, Channel\_4};

}

else if(!keys[10] && !keys[11] && !keys[12] && !keys[13] && keys[5] && !keys[6] && keys[0]){

//C6

fluteNoteAddr = {MIDI\_Notes::C(6), Channel\_4};

}

}

else if(!keys[3] && keys[2] && keys[1] && keys[7] && keys[8] && !keys[9]){

if(!keys[10] && !keys[11] && !keys[12] && !keys[13] && keys[5] && !keys[6] && keys[0]){

//Db6

fluteNoteAddr = {MIDI\_Notes::Db(6), Channel\_4};

}

}

else if(!keys[3] && keys[2] && keys[1] && keys[7] && !keys[8] && !keys[9]){

//D6

fluteNoteAddr = {MIDI\_Notes::D(6), Channel\_4};

}

else if(!keys[3] && keys[2] && keys[1] && !keys[7] && !keys[8] && !keys[9]){

//E6

fluteNoteAddr = {MIDI\_Notes::E(6), Channel\_4};

}

else if(!keys[3] && !keys[2] && !keys[1] && !keys[7] && !keys[8] && !keys[9]){

//G4

fluteNoteAddr = {MIDI\_Notes::G(4), Channel\_4};

}

//call function to send notes on/off

fluteSendNotes();

}

// Flute: Function signals synthesizer to turn flute note w/ variable volume on when right joystick is pressed

void fluteSendNotes(){

bool turnOff = ((fsrStructR.prevVolume <= JOYSTICKUPPERLIMIT && fsrStructR.prevVolume >= JOYSTICKLOWERLIMIT)&&(fsrStructR.currVolume <= JOYSTICKUPPERLIMIT && fsrStructR.currVolume >= JOYSTICKLOWERLIMIT)) ? 1:0;

bool turnOn = (fsrStructR.currVolume >= JOYSTICKUPPERLIMIT || fsrStructR.currVolume <= JOYSTICKLOWERLIMIT) ? 1:0;

if(turnOff && fsrStructR.turnOffFirstTime == 1){

//turn note off

midi.sendNoteOff(fluteNoteAddr, velocity);

fsrStructR.turnOffFirstTime = 0;

fsrStructR.firstTimeNoteOn = 1;

}

else if(turnOn && fsrStructR.firstTimeNoteOn == 1){

int16\_t mapped\_joystick = 0;

if(dataSigReceivedR.joystick >= JOYSTICKUPPERLIMIT){

mapped\_joystick = constrain(map(dataSigReceivedR.joystick, JOYSTICKUPPERLIMIT, 1023, 20, 127), 20, 127);

}

else{

mapped\_joystick = constrain(map(dataSigReceivedR.joystick, JOYSTICKLOWERLIMIT, 0, 20, 127), 20, 127);

}

Serial.println("Mapped Joystick: "+String(mapped\_joystick));

midi.sendControlChange(ccAddr, mapped\_joystick); //control volume then send note on

// Send pitch bend message

int pitchBendValue = map(dataSigReceivedL.joystick, 0, 1023, 0, 16383);

pitchBendValue = constrain(pitchBendValue, 0, 16383); // Ensure the value stays within bounds

midi.sendPitchBend(Channel\_4, pitchBendValue);

midi.sendNoteOn(fluteNoteAddr, velocity);

fsrStructR.turnOffFirstTime = 1;

fsrStructR.firstTimeNoteOn = 0;

}

}

// Bass Guitar: Function determines which note to send

void setBassNotes(){

int16\_t rE\_R = dataSigReceivedR.rotEncoder;

// if zone1 then set notes for right keypad noteAddr1R-noteAddr4R values

bool zone1 = (rE\_R <= maxReRange) ? 1 : 0;

bool zone2 = (rE\_R >= maxReRange+1 && rE\_R <= 2\*maxReRange) ? 1 : 0;

bool zone3 = (rE\_R >= 2\*maxReRange+1 && rE\_R <= 3\*maxReRange) ? 1 : 0;

bool zone4 = (rE\_R >= 3\*maxReRange+1 && rE\_R <= 4\*maxReRange) ? 1 : 0;

if(zone1){ //top of the rod

noteAddr1R = {65, Channel\_6};

noteAddr3R = {MIDI\_Notes::Bb(4), Channel\_6};

noteAddr5R = {MIDI\_Notes::Eb(4), Channel\_6};

noteAddr7R = {MIDI\_Notes::Ab(4), Channel\_6};

}

else if(zone2){

noteAddr1R = {MIDI\_Notes::Gb(4), Channel\_6};

noteAddr3R = {MIDI\_Notes::B(4), Channel\_6};

noteAddr5R = {MIDI\_Notes::E(4), Channel\_6};

noteAddr7R = {MIDI\_Notes::A(4), Channel\_6};

}

else if(zone3){

noteAddr1R = {MIDI\_Notes::G(4), Channel\_6};

noteAddr3R = {MIDI\_Notes::C(4), Channel\_6};

noteAddr5R = {77, Channel\_6};

noteAddr7R = {MIDI\_Notes::Bb(5), Channel\_6};

}

else if(zone4){

noteAddr1R = {MIDI\_Notes::Ab(5), Channel\_6};

noteAddr3R = {MIDI\_Notes::Db(5), Channel\_6};

noteAddr5R = {MIDI\_Notes::Gb(5), Channel\_6};

noteAddr7R = {MIDI\_Notes::B(5), Channel\_6};

}

else{

noteAddr1R = {MIDI\_Notes::A(5), Channel\_6};

noteAddr3R = {MIDI\_Notes::D(5), Channel\_6};

noteAddr5R = {MIDI\_Notes::G(5), Channel\_6};

noteAddr7R = {MIDI\_Notes::C(5), Channel\_6};

}

//Serial.println("Bass Guitar Notes Set");

}

// Bass Guitar

void bassFadeVolume(int ith\_Val){

// Pitch Bend

int pitchBendValue = map(dataSigReceivedL.joystick, 0, 1023, 0, 16383);

pitchBendValue = constrain(pitchBendValue, 0, 16383);

midi.sendPitchBend(Channel\_6, pitchBendValue);

if(fsrStructR.initVolume > fsrStructR.currVolume){

//decrease volume very fast

for(int i=fsrStructR.initVolume; i>=0; i-=10)

{

midi.sendControlChange(ccAddr, fsrStructR.initVolume); //changes CC to volume control option

// would be sending the prev. volume

chooseNoteOn(ith\_Val, 'R');

}

}

else{

// decrease volume slowly

for(int i=fsrStructR.initVolume; i>=0; i-=5)

{

midi.sendControlChange(ccAddr, fsrStructR.initVolume); //changes CC to volume control option

// would be sending the prev. volume

chooseNoteOn(ith\_Val, 'R');

}

}

//update noteOnHis struct here

noteOnSig.ith\_Val[ith\_Val] = 1;

noteOnSig.whichKeypad[ith\_Val] = 'R';

}

// Bass Guitar: Function determines which note is turned on or off

void bassGuitarMode(int i){

if(fsrStructR.prevVolume <= 24 && fsrStructR.currVolume <= 24 && fsrStructR.turnOffFirstTime == 1)

{

//Serial.println("Bass Guitar Note Off");

// make sure to only turn the previous note off

if(noteOnSig.ith\_Val[i] == 1 && noteOnSig.whichKeypad[i] == 'R'){

chooseNoteOff(i,'R');

noteOnSig.ith\_Val[i] = 0;

noteOnSig.whichKeypad[i] = 'N';

}

fsrStructR.turnOffFirstTime = 0;

fsrStructR.firstTimeNoteOn = 1;

}

else if(fsrStructR.currVolume >= 25 && fsrStructR.firstTimeNoteOn == 1){ //NOTE SHOULD ONLY GO OFF ONCE WHICH IS AT INITIAL TIME OF PRESS, after that it will never go here

//Serial.println("Bass Guitar Note On");

bassFadeVolume(i);

fsrStructR.turnOffFirstTime = 1;

fsrStructR.firstTimeNoteOn = 0;

}

else{

//Serial.println("Bass Guitar Skips");

}

}

// Chinese Piano

void setCPianoNotes(int8\_t tiltSensor, char whichKeypad){

int16\_t rE\_L = dataSigReceivedL.rotEncoder;

int16\_t rE\_R = dataSigReceivedR.rotEncoder;

if(whichKeypad == 'L'){

if(rE\_L <= maxReRange){

//set note for fsr6 to C and fsr0 to F

noteAddr1L = {MIDI\_Notes::E(2), Channel\_5};

noteAddr2L = {41, Channel\_5};

noteAddr5L = {MIDI\_Notes::G(2), Channel\_5};

noteAddr7L = {MIDI\_Notes::A(2), Channel\_5};

noteAddr3L = {MIDI\_Notes::Eb(2), Channel\_5};

noteAddr4L = {MIDI\_Notes::Gb(2), Channel\_5};

noteAddr6L = {MIDI\_Notes::Ab(2), Channel\_5};

//Serial.println("Zone 1");

}

else if(rE\_L >= maxReRange+1 && rE\_L <= 2\*maxReRange){

//set note for fsr6 to C and fsr0 to F

noteAddr1L = {MIDI\_Notes::B(2), Channel\_5};

noteAddr2L = {MIDI\_Notes::C(3), Channel\_5};

noteAddr5L = {MIDI\_Notes::D(3), Channel\_5};

noteAddr7L = {MIDI\_Notes::E(3), Channel\_5};

noteAddr3L = {MIDI\_Notes::Bb(2), Channel\_5};

noteAddr4L = {MIDI\_Notes::Db(3), Channel\_5};

noteAddr6L = {MIDI\_Notes::Eb(3), Channel\_5};

//Serial.println("Zone 2");

}

else if(rE\_L >= 2\*maxReRange+1 && rE\_L <= 3\*maxReRange){

//set note for fsr6 to C and fsr0 to F

//MIDI\_Notes::note ( int8\_t note,int8\_t numOctave ) ;

noteAddr1L = {53, Channel\_5};

//noteAddr1L = {53, Channel\_5};

noteAddr2L = {MIDI\_Notes::G(3), Channel\_5};

noteAddr5L = {MIDI\_Notes::A(3), Channel\_5};

noteAddr7L = {MIDI\_Notes::B(3), Channel\_5};

noteAddr3L = {MIDI\_Notes::Gb(3), Channel\_5};

noteAddr4L = {MIDI\_Notes::Ab(3), Channel\_5};

noteAddr6L = {MIDI\_Notes::Bb(3), Channel\_5};

//Serial.println("Zone 3");

}

else if(rE\_L >= 3\*maxReRange+1 && rE\_L <= 4\*maxReRange){

//set note for fsr6 to C and fsr0 to F

noteAddr1L = {MIDI\_Notes::C(4), Channel\_5};

noteAddr2L = {MIDI\_Notes::D(4), Channel\_5};

noteAddr5L = {MIDI\_Notes::E(4), Channel\_5};

noteAddr7L = {65, Channel\_5};

noteAddr3L = {MIDI\_Notes::Db(4), Channel\_5};

noteAddr4L = {MIDI\_Notes::Eb(4), Channel\_5};

noteAddr6L = {MIDI\_Notes::Gb(4), Channel\_5};

//Serial.println("Zone 4");

}

else if(rE\_L >= 4\*maxReRange+1 && rE\_L <= 5\*maxReRange){

noteAddr1L = {MIDI\_Notes::G(4), Channel\_5};

noteAddr2L = {MIDI\_Notes::A(4), Channel\_5};

noteAddr5L = {MIDI\_Notes::B(4), Channel\_5};

noteAddr7L = {MIDI\_Notes::C(5), Channel\_5};

noteAddr3L = {MIDI\_Notes::Ab(4), Channel\_5};

noteAddr4L = {MIDI\_Notes::Bb(4), Channel\_5};

noteAddr6L = {MIDI\_Notes::Db(5), Channel\_5};

//Serial.println("Zone 5");

}

else if(rE\_L >= 5\*maxReRange+1){

noteAddr1L = {MIDI\_Notes::D(5), Channel\_5};

noteAddr2L = {MIDI\_Notes::E(5), Channel\_5};

noteAddr5L = {77, Channel\_5};

noteAddr7L = {MIDI\_Notes::G(5), Channel\_5};

noteAddr3L = {MIDI\_Notes::Eb(5), Channel\_5};

noteAddr4L = {MIDI\_Notes::Gb(5), Channel\_5};

noteAddr6L = {MIDI\_Notes::Ab(5), Channel\_5};

//Serial.println("Zone 6");

}

}

else{

if(rE\_R <= maxReRange){

noteAddr1R = {MIDI\_Notes::D(5), Channel\_5};

noteAddr3R = {MIDI\_Notes::E(5), Channel\_5};

noteAddr5R = {77, Channel\_5};

noteAddr7R = {MIDI\_Notes::G(5), Channel\_5};

noteAddr2R = {MIDI\_Notes::Eb(5), Channel\_5};

noteAddr4R = {MIDI\_Notes::Gb(5), Channel\_5};

noteAddr6R = {MIDI\_Notes::Ab(5), Channel\_5};

//Serial.println("R: Zone 6");

}

else if(rE\_R >= maxReRange+1 && rE\_R <= 2\*maxReRange){

noteAddr1R = {MIDI\_Notes::G(4), Channel\_5};

noteAddr3R = {MIDI\_Notes::A(4), Channel\_5};

noteAddr5R = {MIDI\_Notes::B(4), Channel\_5};

noteAddr7R = {MIDI\_Notes::C(5), Channel\_5};

noteAddr2R = {MIDI\_Notes::Ab(4), Channel\_5};

noteAddr4R = {MIDI\_Notes::Bb(4), Channel\_5};

noteAddr6R = {MIDI\_Notes::Db(5), Channel\_5};

//Serial.println("R: Zone 5");

}

else if(rE\_R >= 2\*maxReRange+1 && rE\_R <= 3\*maxReRange){

//set note for fsr6 to C and fsr0 to F

noteAddr1R = {MIDI\_Notes::C(4), Channel\_5};

noteAddr3R = {MIDI\_Notes::D(4), Channel\_5};

noteAddr5R = {MIDI\_Notes::E(4), Channel\_5};

noteAddr7R = {65, Channel\_5};

noteAddr2R = {MIDI\_Notes::Db(4), Channel\_5};

noteAddr4R = {MIDI\_Notes::Eb(4), Channel\_5};

noteAddr6R = {MIDI\_Notes::Gb(4), Channel\_5};

//Serial.println("R: Zone 4");

}

else if(rE\_R >= 3\*maxReRange+1 && rE\_R <= 4\*maxReRange){

noteAddr1R = {53, Channel\_5};

noteAddr3R = {MIDI\_Notes::G(3), Channel\_5};

noteAddr5R = {MIDI\_Notes::A(3), Channel\_5};

noteAddr7R = {MIDI\_Notes::B(3), Channel\_5};

noteAddr2R = {MIDI\_Notes::Gb(3), Channel\_5};

noteAddr4R = {MIDI\_Notes::Ab(3), Channel\_5};

noteAddr6R = {MIDI\_Notes::Bb(3), Channel\_5};

//Serial.println("R: Zone 3");

}

else if(rE\_R >= 4\*maxReRange+1 && rE\_R <= 5\*maxReRange){

//set note for fsr6 to C and fsr0 to F

noteAddr1R = {MIDI\_Notes::B(2), Channel\_5};

noteAddr3R = {MIDI\_Notes::C(3), Channel\_5};

noteAddr5R = {MIDI\_Notes::D(3), Channel\_5};

noteAddr7R = {MIDI\_Notes::E(3), Channel\_5};

noteAddr2R = {MIDI\_Notes::Bb(2), Channel\_5};

noteAddr4R = {MIDI\_Notes::Db(3), Channel\_5};

noteAddr6R = {MIDI\_Notes::Eb(3), Channel\_5};

//Serial.println("R: Zone 2");

}

else if(rE\_R >= 5\*maxReRange+1){

//set note for fsr6 to C and fsr0 to F

noteAddr1R = {MIDI\_Notes::E(2), Channel\_5};

noteAddr3R = {41, Channel\_5};

noteAddr5R = {MIDI\_Notes::G(2), Channel\_5};

noteAddr7R = {MIDI\_Notes::A(2), Channel\_5};

noteAddr2R = {MIDI\_Notes::Eb(2), Channel\_5};

noteAddr4R= {MIDI\_Notes::Gb(2), Channel\_5};

noteAddr6R = {MIDI\_Notes::Ab(2), Channel\_5};

//Serial.println("R: Zone 1");

}

}

}

// Piano: Function to produce key fade in piano mode

void cPianoFadeVolume(int ith\_Val, char whichKeypad){

// Pitch Bend

int pitchBendValue = map(dataSigReceivedL.joystick, 0, 1023, 0, 16383);

pitchBendValue = constrain(pitchBendValue, 0, 16383);

midi.sendPitchBend(Channel\_5, pitchBendValue);

// manipulate the velocity for this

// decrease faster if the velocity by a lot if the initial pressure is greater than following pressure

// decrease slowly if the initial pressure is equal to, a little bit greater than or less than following pressure

if(whichKeypad == 'L'){

if(fsrStructL.initVolume > fsrStructL.currVolume){

//decrease volume very fast

for(int i=fsrStructL.initVolume; i>=0; i-=10)

{

midi.sendControlChange(ccAddr, fsrStructL.initVolume); //changes CC to volume control option

// would be sending the prev. volume

chooseNoteOn(ith\_Val, 'L');

}

}

else{

// decrease volume slowly

for(int i=fsrStructL.initVolume; i>=0; i-=5)

{

midi.sendControlChange(ccAddr, fsrStructL.initVolume); //changes CC to volume control option

// would be sending the prev. volume

chooseNoteOn(ith\_Val, 'L');

}

}

}

else{

if(fsrStructR.initVolume > fsrStructR.currVolume){

//decrease volume very fast

for(int i=fsrStructR.initVolume; i>=0; i-=10)

{

midi.sendControlChange(ccAddr, fsrStructR.initVolume); //changes CC to volume control option

// would be sending the prev. volume

chooseNoteOn(ith\_Val, 'R');

}

}

else{

// decrease volume slowly

for(int i=fsrStructR.initVolume; i>=0; i-=5)

{

midi.sendControlChange(ccAddr, fsrStructR.initVolume); //changes CC to volume control option

// would be sending the prev. volume

chooseNoteOn(ith\_Val, 'R');

}

}

}

//update noteOnHis struct here

noteOnSig.ith\_Val[ith\_Val] = 1;

noteOnSig.whichKeypad[ith\_Val] = whichKeypad;

}

// Chinese Piano

void chinesePianoMode(int i){

if(fsrStructL.prevVolume <= 24 && fsrStructL.currVolume <= 24 && fsrStructL.turnOffFirstTime == 1)

{

// make sure to only turn the previous note off

if(noteOnSig.ith\_Val[i] == 1 && noteOnSig.whichKeypad[i] == 'L'){

chooseNoteOff(i,'L');

noteOnSig.ith\_Val[i] = 0;

noteOnSig.whichKeypad[i] = 'N';

}

fsrStructL.turnOffFirstTime = 0;

fsrStructL.firstTimeNoteOn = 1;

}

else if(fsrStructL.currVolume >= 25 && fsrStructL.firstTimeNoteOn == 1){ //NOTE SHOULD ONLY GO OFF ONCE WHICH IS AT INITIAL TIME OF PRESS, after that it will never go here

cPianoFadeVolume(i, 'L');

fsrStructL.turnOffFirstTime = 1;

fsrStructL.firstTimeNoteOn = 0;

}

else{

//Serial.println("L: Do Nothing");

}

if(fsrStructR.prevVolume <= 24 && fsrStructR.currVolume <= 24 && fsrStructR.turnOffFirstTime == 1)

{

// make sure to only turn the previous note off

if(noteOnSig.ith\_Val[i] == 1 && noteOnSig.whichKeypad[i] == 'R'){

chooseNoteOff(i, 'R');

noteOnSig.ith\_Val[i] = 0;

noteOnSig.whichKeypad[i] = 'N';

}

fsrStructR.turnOffFirstTime = 0;

fsrStructR.firstTimeNoteOn = 1;

}

else if(fsrStructR.currVolume >= 25 && fsrStructR.firstTimeNoteOn == 1){ //NOTE SHOULD ONLY GO OFF ONCE WHICH IS AT INITIAL TIME OF PRESS, after that it will never go here

cPianoFadeVolume(i, 'R');

fsrStructR.turnOffFirstTime = 1;

fsrStructR.firstTimeNoteOn = 0;

}

else{

//Serial.println("R: Do Nothing");

}

}

// Final Demo Function///////////////////////////////////////////////////////////////////////////////////////////

int switchInstruments(int8\_t tiltSensor, int16\_t rE\_L){

char tilt = 'A';

int8\_t rotPlate\_L = (dataSigReceivedL.rotPlate1<<2) | (dataSigReceivedL.rotPlate2<<1) | dataSigReceivedL.rotPlate3; //left rotation Plate

//update history of rotPlate

if(rotPlate\_L != 0b111){

rotPlateHist[0] = rotPlateHist[1];

rotPlateHist[1] = rotPlate\_L;

}

//Serial.println("s tiltSensor: "+String(tiltSensor));

Serial.println("Left Rotational Plate: "+String(rotPlate\_L));

//determine orientation of tilt sensor

if(tiltSensor == 0b011 || tiltSensor == 0b110 || tiltSensor == 0b001){

tilt = 'V';

}

else if(tiltSensor == 0b101){

tilt = 'A';

}

else{

tilt = 'H';

}

//Serial.println("Tilt Sensor: "+String(tiltSensor));

Serial.println("Rotational Plate: "+String(rotPlateHist[1]));

if(tilt == 'H'){

if(rotPlateHist[1] == 0b101){

//channel 1 to piano

return 1;

}

else if(rotPlateHist[1] == 0b011){

//channel 4 to flute

return 4;

}

else if(rotPlateHist[1] == 0b110){

//channel 5 to Chinese Piano

return 5;

}

}

else if(tilt == 'A'){

if(dataSigReceivedR.rotEncoder < borderValue\_L){

//channel 2 to sax

return 2;

}

else if(dataSigReceivedR.rotEncoder >= borderValue\_L){

//channel 3 to clarinet

return 3;

}

}

else if(tilt == 'V'){

//channel 6 to bass guitar

return 6;

}

return 1;

}

void setup(void){

Serial.begin(9600);

pinMode(sensorHPin, INPUT);

pinMode(sensorVPin, INPUT);

pinMode(sensorAPin, INPUT);

radio.begin(); // Start the NRF24L01

radio.openReadingPipe(1, 0xABCDEF12); //left keypad Tx

radio.openReadingPipe(2, 0xABCDEF23); //right keypad Tx

radio.setPALevel(RF24\_PA\_LOW);

radio.startListening();

midi.begin(); //Initializes MIDI interface

Serial.println("-------------------------Program Start------------------------");

}

void loop(void){

//read in values for tiltSensor

int sensor\_HValue = digitalRead(sensorHPin);

int sensor\_VValue = digitalRead(sensorVPin);

int sensor\_AValue = digitalRead(sensorAPin);

int8\_t tiltSensor = (sensor\_HValue << 2) | (sensor\_VValue << 1) | sensor\_AValue;

tiltSensorNoise(tiltSensor);

Serial.println("Tilt Sensor: "+String(tiltSensor));

if(radio.available()){

radio.read(&tempSig, sizeof(tempSig));

//if left or right is 0, then copy all values from tempSig to dataSigReceivedL, else dataSigReceivedR

if(tempSig.lOrR == 0){

memcpy(&dataSigReceivedL, &tempSig, sizeof(tempSig));

}

else{

memcpy(&dataSigReceivedR, &tempSig, sizeof(tempSig));

}

for(int8\_t i=startingMuxSel; i<=(startingMuxSel + 0b0110); i++){

pickStruct(i); //copy dataSigReceived to temp struct

int16\_t rE\_L = dataSigReceivedL.rotEncoder;

//only needs to be called once since both keypads should be on the same channel

int which\_channel = switchInstruments(tiltSensor, rE\_L);

if(which\_channel == 1){

Serial.println("Piano Mode On");

fsrStructL.prevVolume = fsrStructL.currVolume;

fsrStructR.prevVolume = fsrStructR.currVolume;

fsrStructL.currVolume = mapFSRtoVolume(i, 'L');

fsrStructR.currVolume = mapFSRtoVolume(i, 'R');

// initalize volume: fsrStuct(L/R).initVolume;

initializeVol(which\_channel);

// set notes to keys here and return a zone to prevent B sharp from turning on

setPianoNotes(cmpTilt, 'L');

setPianoNotes(cmpTilt, 'R');

// Pitch Bend

int pitchBendValue = map(dataSigReceivedL.joystick, 0, 1023, 0, 16383);

pitchBendValue = constrain(pitchBendValue, 0, 16383);

midi.sendPitchBend(Channel\_1, pitchBendValue);

// if note is released or just not pressed, then proceed, else turn note on

pianoMode(i);

}

else if(which\_channel == 2){

Serial.println("Saxophone Mode On");

//maintain volume history

fsrStructR.prevVolume = fsrStructR.currVolume;

fsrStructR.currVolume = dataSigReceivedR.joystick;

initializeVol(which\_channel);

//call function for sax...

saxMode();

}

else if(which\_channel == 3){

Serial.println("Clarinet Mode On");

//maintain volume history

fsrStructR.prevVolume = fsrStructR.currVolume;

fsrStructR.currVolume = dataSigReceivedR.joystick;

initializeVol(which\_channel);

//call function for clarinet...

clarinetMode();

}

else if(which\_channel == 4){

Serial.println("Flute Mode On");

//maintain volume history

fsrStructR.prevVolume = fsrStructR.currVolume;

fsrStructR.currVolume = dataSigReceivedR.joystick;

initializeVol(which\_channel);

//call function for flute...

fluteMode();

}

else if(which\_channel == 5){

Serial.println("Chinese Piano Mode On");

//call function for chinese piano...

fsrStructL.prevVolume = fsrStructL.currVolume;

fsrStructR.prevVolume = fsrStructR.currVolume;

fsrStructL.currVolume = mapFSRtoVolume(i, 'L');

fsrStructR.currVolume = mapFSRtoVolume(i, 'R');

// initalize volume: fsrStuct(L/R).initVolume;

initializeVol(which\_channel);

// set notes to keys here and return a zone to prevent B sharp from turning on

setCPianoNotes(cmpTilt, 'L');

setCPianoNotes(cmpTilt, 'R');

// if note is released or just not pressed, then proceed, else turn note on

chinesePianoMode(i);

}

else if(which\_channel == 6){

Serial.println("Bass Guitar Mode On");

//maintain volume history

fsrStructR.prevVolume = fsrStructR.currVolume;

fsrStructR.currVolume = mapFSRtoVolume(i, 'R');

initializeVol(which\_channel);

//call function for bass guitar...

setBassNotes();

bassGuitarMode(i);

}

//call function to copy all values from fsrReading(L/R) to struct

copytoStruct(i);

}

Serial.println("");

Serial.println("dataSigReceivedL: "+String(dataSigReceivedL.fsr0)+", "+String(dataSigReceivedL.fsr1)+", "+String(dataSigReceivedL.fsr2)+", "+String(dataSigReceivedL.fsr3)+", "+String(dataSigReceivedL.fsr4)+", "+String(dataSigReceivedL.fsr5)+", "+String(dataSigReceivedL.fsr6));

Serial.println("dataSigReceivedR: "+String(dataSigReceivedR.fsr0)+", "+String(dataSigReceivedR.fsr1)+", "+String(dataSigReceivedR.fsr2)+", "+String(dataSigReceivedR.fsr3)+", "+String(dataSigReceivedR.fsr4)+", "+String(dataSigReceivedR.fsr5)+", "+String(dataSigReceivedR.fsr6));

Serial.println("Joystick(Left/Right): "+String(dataSigReceivedL.joystick)+","+String(dataSigReceivedR.joystick));

Serial.println("Rotary Encoder(Left/Right): "+String(dataSigReceivedL.rotEncoder)+","+String(dataSigReceivedR.rotEncoder));

}

else

{

Serial.println("No msg received ");

}

midi.update(); // Handle or discard MIDI input

delay(23);

}

## Appendix D - Code for Left Keypad(Transmitter)

/\*

Programmer: Jolin Lin

ECD 423 - Novel MIDI Instrument (Senior PRJ)

Date: 04/24/2024

Purpose: This program is a continuation of demo11\_rotaryEncoder by receiving values from the rotary encoder, tilt sensor and rotational sensor to switch between instruments(channels). Tilt sensors are directly connected to the based station using a wire.

\*/

#include <SPI.h>

#include <RF24.h>

#include <light\_CD74HC4067.h> //library for mux

#include <Encoder.h> //lib for Teensy

RF24 radio(9,10); // nRF: CE,CSN

// Analog and digital pins below

const int MUX\_ENABLE = 8;

const int S0 = 4;

const int S1 = 3;

const int S2 = 2;

const int S3 = 1;

const int ROTENC1 = 6; //Rotary Encoder

const int ROTENC2 = 7;

const int ROTPLATE1 = 17;

const int ROTPLATE2 = 18;

const int ROTPLATE3 = 19;

const int SIG = A6;

const int JOYSTICK = A8;

CD74HC4067 mux(S0,S1,S2,S3); //creates a new object

Encoder myEnc(7,6);

// struct to send all 7 FSR data to the base station

struct DataSig{

bool lOrR;

uint16\_t fsr0;

uint16\_t fsr1;

uint16\_t fsr2;

uint16\_t fsr3;

uint16\_t fsr4;

uint16\_t fsr5;

uint16\_t fsr6;

int16\_t rotEncoder;

bool rotPlate1; //Rotational Plate Sensors

bool rotPlate2;

bool rotPlate3;

int16\_t joystick;

};

//initalize struct

struct DataSig dataSigSend = {0,0,0,0,0,0,0,0,0,0,0};

//Function to assign fsrDataSig to appropriate int within struct dataSigSend

void setStructData(int fsrDataSig, int i){

if(i == 0b0000){

dataSigSend.fsr0 = fsrDataSig;

}

else if(i == 0b0001){

dataSigSend.fsr1 = fsrDataSig;

}

else if(i == 0b0010){

dataSigSend.fsr2 = fsrDataSig;

}

else if(i == 0b0011){

dataSigSend.fsr3 = fsrDataSig;

}

else if(i == 0b0100){

dataSigSend.fsr4 = fsrDataSig;

}

else if(i == 0b0101){

dataSigSend.fsr5 = fsrDataSig;

}

else if(i == 0b0110){

dataSigSend.fsr6 = fsrDataSig;

}

else{

Serial.println("Not updating anything");

}

}

void setup() {

Serial.begin(9600);

pinMode(ROTPLATE1, INPUT\_PULLUP); //Pull up resistors for rotational plate

pinMode(ROTPLATE2, INPUT\_PULLUP);

pinMode(ROTPLATE3, INPUT\_PULLUP);

radio.begin(); // Starts the NRF24L01

radio.openWritingPipe(0xABCDEF12);

radio.setPALevel(RF24\_PA\_MIN);

Serial.println("-------------------------Program Start------------------------");

}

int16\_t iterate = 0;

int16\_t z=0;

void loop() {

int16\_t newPosition = myEnc.read();

if(newPosition != dataSigSend.rotEncoder){

dataSigSend.rotEncoder = newPosition;

}

//read joystick value

dataSigSend.joystick = analogRead(JOYSTICK);

//reading rotational plate sensors

dataSigSend.rotPlate1 = digitalRead(ROTPLATE1);

dataSigSend.rotPlate2 = digitalRead(ROTPLATE2);

dataSigSend.rotPlate3 = digitalRead(ROTPLATE3);

digitalWrite(MUX\_ENABLE, HIGH);

// for each iteration of the main loop, i increases from 0-6 back to 0

mux.channel(iterate);

int val = analogRead(SIG);

//assign val to appropriate fsr# within struct dataSigSend

setStructData(val, iterate);

digitalWrite(MUX\_ENABLE, LOW);

//set LeftOrRight to 0

dataSigSend.lOrR = 0;

//increase value of i

iterate += 1;

if(iterate >= 7){

iterate = 0;

radio.write(&dataSigSend, sizeof(dataSigSend));

Serial.println("Mux Values: " + String(dataSigSend.fsr0) + ", " + String(dataSigSend.fsr1) + ", " + String(dataSigSend.fsr2) + ", " + String(dataSigSend.fsr3) + ", " + String(dataSigSend.fsr4) + ", " + String(dataSigSend.fsr5) + ", " + String(dataSigSend.fsr6));

Serial.println("Rotational Sensors: "+String(dataSigSend.rotPlate1)+", " + String(dataSigSend.rotPlate2) + ", " + String(dataSigSend.rotPlate3));

Serial.println("RotaryEncoder Val: "+String(dataSigSend.rotEncoder));

Serial.println("Joystick Val: "+String(dataSigSend.joystick));

Serial.println("");

}

delay(2); //slight delay to prevent multi-Tx data signals from cancelling each other out

}

## Appendix E - Code for Right Keypad(Transmitter)

/\*

Programmer: Jolin Lin

ECD 423 - Novel MIDI Instrument (Senior PRJ)

Date: 04/24/2024

Purpose: This program is a continuation of demo11\_rotaryEncoder by receiving values from the rotary encoder, tilt sensor and rotational sensor to switch between instruments(channels). Tilt sensors are directly connected to the based station using a wire.

\*/

#include <SPI.h>

#include <RF24.h>

#include <light\_CD74HC4067.h> //library for mux

#include <Encoder.h> //library for Teensy 4.0

RF24 radio(9,10); //nRF: CE,CSN

// Analog and digital pins below

const int MUX\_ENABLE = 8;

const int S0 = 4;

const int S1 = 3;

const int S2 = 2;

const int S3 = 1;

const int8\_t ROTENC1 = 6;

const int8\_t ROTENC2 = 7;

const int ROTPLATE1 = 17;

const int ROTPLATE2 = 18;

const int ROTPLATE3 = 19;

const int SIG = A7;

const int JOYSTICK = A8;

CD74HC4067 mux(S0,S1,S2,S3); //creates a new object

Encoder myEnc(7,6);

// struct to send all 7 FSR data to the base station: total of 21 bytes

struct DataSig{

bool lOrR;

uint16\_t fsr0;

uint16\_t fsr1;

uint16\_t fsr2;

uint16\_t fsr3;

uint16\_t fsr4;

uint16\_t fsr5;

uint16\_t fsr6;

int16\_t rotEncoder;

bool rotPlate1;

bool rotPlate2;

bool rotPlate3;

int16\_t joystick;

};

struct DataSig dataSigSend = {0,0,0,0,0,0,0,0,0,0,0};

//Function to assign fsrDataSig to appropriate int within struct dataSigSend

void setStructData(int fsrDataSig, int i){

if(i == 0b0000){

dataSigSend.fsr0 = fsrDataSig;

}

else if(i == 0b0001){

dataSigSend.fsr1 = fsrDataSig;

}

else if(i == 0b0010){

dataSigSend.fsr2 = fsrDataSig;

}

else if(i == 0b0011){

dataSigSend.fsr3 = fsrDataSig;

}

else if(i == 0b0100){

dataSigSend.fsr4 = fsrDataSig;

}

else if(i == 0b0101){

dataSigSend.fsr5 = fsrDataSig;

}

else if(i == 0b0110){

dataSigSend.fsr6 = fsrDataSig;

}

else{

Serial.println("Not updating anything");

}

}

void setup() {

Serial.begin(9600);

pinMode(ROTPLATE1, INPUT\_PULLUP);

pinMode(ROTPLATE2, INPUT\_PULLUP);

pinMode(ROTPLATE3, INPUT\_PULLUP);

radio.begin(); // Starts the NRF24L01

radio.openWritingPipe(0xABCDEF23);

radio.setPALevel(RF24\_PA\_MIN);

Serial.println("-------------------------Program Start------------------------");

}

int16\_t iterate = 0;

int16\_t z=0;

void loop() {

int16\_t newPosition = myEnc.read();

if(newPosition != dataSigSend.rotEncoder){

dataSigSend.rotEncoder = newPosition/4;

}

//read joystick value

dataSigSend.joystick = analogRead(JOYSTICK);

//reading rotational plate sensors

dataSigSend.rotPlate1 = digitalRead(ROTPLATE1);

dataSigSend.rotPlate2 = digitalRead(ROTPLATE2);

dataSigSend.rotPlate3 = digitalRead(ROTPLATE3);

digitalWrite(MUX\_ENABLE, HIGH);

// for each iteration of the main loop, i increases from 0-6 back to 0

mux.channel(iterate); //opens channel i on mux

int val = analogRead(SIG);

//assign val to appropriate fsr# within struct dataSigSend

setStructData(val, iterate);

digitalWrite(MUX\_ENABLE, LOW);

// set LeftOrRight to 1

dataSigSend.lOrR = 1;

//increase value of i

iterate += 1;

if(iterate >= 7){

iterate = 0;

radio.write(&dataSigSend, sizeof(dataSigSend));

Serial.println("Mux Values: " + String(dataSigSend.fsr0) + ", " + String(dataSigSend.fsr1) + ", " + String(dataSigSend.fsr2) + ", " + String(dataSigSend.fsr3) + ", " + String(dataSigSend.fsr4) + ", " + String(dataSigSend.fsr5) + ", " + String(dataSigSend.fsr6));

Serial.println("Rotational Sensors: "+String(dataSigSend.rotPlate1)+", " + String(dataSigSend.rotPlate2) + ", " + String(dataSigSend.rotPlate3));

Serial.println("RotaryEncoder Val: "+String(dataSigSend.rotEncoder));

Serial.println("Joystick Val: "+String(dataSigSend.joystick));

Serial.println("");

}

}

## Appendix F - INT Plan

These qualification methods will be used in verifying that the project system meets all requirements in its Project Specification.

* Demonstration (D)
* Test (T)
* Analysis (A)
* Inspection (I)

The following Requirement Categories (RC) are used to describe the purpose of each test:

* System Capability Requirements (SC)
* System External Interface Requirements (EI)
* Project Business Requirements (PB)
* Other Requirements (O)

## Verification Overview

The table displayed below shows all of the tests to be performed as proof of meeting all design requirements mentioned in the project specifications document. The name of the test is provided along with their Test ID, details regarding the requirement being fulfilled and test completion date. Please note that test completion dates which display times that have yet to arrive are simply estimated dates and not the actual time of completion.

## Test Coverage

| **Test ID** | **Test Name** | **QM** | **RC** | **Requirements Addressed** | **Test Completion Date** |
| --- | --- | --- | --- | --- | --- |
| ECD423-T-001 | MIDI Test | I,D,T | O,EI,SC | ECD423-R-001  ECD423-R-002  ECD423-R-003  ECD423-R-004  ECD423-R-011 | 2023-11-13 |
| ECD423-T-002 | Wireless Communication Distance Test | I | O,SC | ECD423-R-004  ECD423-R-008 | 2024-02-09 |
| ECD423-T-003 | Multiple Transceivers Communication Test | I,T | O, SC | ECD423-R-008  ECD423-G-001 | 2024-03-01 |
| ECD423-T-004 | Tilt Sensor Test | A,I | SC | ECD423-R-009  ECD423-R-010 | 2024-03-08 |
| ECD423-T-005 | Rotary Encoder Test | D | SC | ECD423-R-006  ECD423-R-007  ECD423-R-009  ECD423-G-004 | 2024-03-08 |
| ECD423-T-006 | Keypad Rotation Test | I,D,A | SC | ECD423-R-005  ECD423-R-006  ECD423-R-007  ECD423-R-009  ECD423-G-004 | 2024-03-15 |
| ECD423-T-007 | Instrument Transition Test | D | SC | ECD423-R-007  ECD423-G-004 | 2024-03-22 |
| ECD423-T-008 | Variable Volume Test | D,I | SC, O | ECD423-R-003  ECD423-R-004 | 2024-03-01 |
| ECD423-T-009 | Lamellophone Key Test | D,I | SC | ECD423-R-003  ECD423-G-005 | 2024-03-08 |
| ECD423-T-010 | Joystick Test | D | SC | ECD423-R-002 | 2024-03-15 |
| ECD423-T-011 | Final Test | D | SC, EI | All Requirements | 2024-04-15 |

**6 System Verification Tests**

## 

### *Communications Test Group Overview*

This test group includes all wireless communication tests and the outputting of MIDI messages.

### *Test Procedures*

#### *ECD423-T-001 MIDI Test*

This test is implemented with the help of the Teensy 4.0 device which is connected to a computer via USB connection. The purpose of this evaluation is to demonstrate the ability to send MIDI signals which emit notes with varying variable volume control and pitch bend. Additionally, the program should switch between different channels. This is done by programming the microcontroller with code from Arduino IDE. This code utilizes built in functions from the Control Surface Library to create for loops which demonstrate scaling of the notes ranging from 0-127. An additional two for loops are used to show variable volume ranging from values 0-127 and pitch bend. The results of this program are shown through the monitor on the synthesizer program, Element.

| **#** | **Step-by-Step Operations** | **Expected Results** | Actual Results |
| --- | --- | --- | --- |
| 1 | Connect Teensy 4.0 to the computer via USB connection. | No results expected as this is setup |  |
| 2 | Open, verify and upload the program used to demonstrate this test. | The Arduino IDE Serial Monitor should display which channels are currently on/off at any given time in addition to their volume, and note being sent. |  |
| 3 | Open the synthesizer program, Element. Go to File>Preferences>MIDI and turn “Active MIDI inputs” on. | All current results are maintained with no added changes made. |  |
| 4 | Turn your volume to the appropriate range and start observing the MIDI monitor. | There should be a total of 3 MIDI monitors displaying the current note, pitch bend, and volume control at any given time on the synthesizer. |  |

#### *ECD423-T-002 Wireless Communication Distance Test*

This test will ensure that both keypads can send wireless messages to the base station reliably without transmission errors. Additionally, the wireless modules must be able to maintain communication of up to 50 inches. This test consists of two Arduino Nano’s/Teensy with nRF24L01 transceivers transmitting data and 1 Teensy 4.0 with a nRF24L01 transceiver receiving the data. The transmitters will slowly be moved away from the source until the receiver can no longer reliably receive data. An evaluator will stop and record the distance between modules once the connection has been broken.

| **#** | **Step-by-Step Operations** | **Expected Results** | Actual Results |
| --- | --- | --- | --- |
| 1 | Connect the 2 Nano’s/Teensy’s and 1 Teensy 4.0 to their respective nRF24L01 modules. | No results expected as this is setup |  |
| 2 | Connect the Teensy 4.0 to one laptop and both Nano’s/Teensy’s to another. Note, for this experiment the Nano’s/Teensy’s and Teensy must not be using the same laptop unless you have cables that can extend at least 50 inches in width. | No results expected as this is setup |  |
| 3 | Open, verify and upload the code for the transceivers to the Nano’s/Teensy’s and the code for the receiver to the Teensy. | Serial Monitor should display that the program has finished set up. |  |
| 4 | Have person A look at the Serial Monitor on the Arduino IDE and person B to move the Nano’s/Teensy’s away from the Teensy at a steady pace. When the connection is broken, person A will tell person B to stop. Person B will measure the distance from the Teensy to the Nano/Teensy. | Person A should see on the Serial Monitor for the receiver code, the message being received. When the message to be received is displaying “No msg received” for an extended period of time. Person B should record the distance between the wireless modules and check to see that the distance is at least 50 inches apart. | Transceivers were able to read past 80 inches before testing stopped. |

#### *ECD423-T-003 Multiple Transceivers Communication Test*

This is a test performed with the sole purpose of connecting two transceiver modules to a single receiver module for constant uni-directional communication. Each transceiver module will be attached to each keypad respectively and the receiver module will be located at the base station. The strength of transmission signals will be recorded based on the frequency of failed transmission. The average time to transmit each signal will be taken into consideration in regards to evaluating the quality of this communication.

| **#** | **Step-by-Step Operations** | **Expected Results** | Actual Results |
| --- | --- | --- | --- |
| 1 | Connect the 2 Nano’s and 1 Teensy 4.0 to their respective nRF24L01 modules. | No results expected as this is setup |  |
| 2 | Connect the Teensy 4.0 to one laptop and both Nano’s to another. Note, for this experiment the Nano’s and Teensy must not be using the same laptop unless you have cables that can extend at least 50 inches in width. | No results expected as this is setup |  |
| 3 | Open, verify and upload the code for the transceivers to the Nano’s and the code for the receiver to the Teensy.  Make sure to type “R” into the Serial Monitor for the receiver and different node addresses, from 0-5, for each transceiver. | The Serial Monitor for the receiver should recognize it is in receiving mode and that of the transceiver should start sending transmission signal’s with payload ID. | Transceivers are in the correct mode. |
| 4 | Observe the Serial Monitor for both transceivers and determine the average time to transmit signals for each. Take note of the max value observed from the Serial Monitors. | The transceiver should be taking on average 580-600us to transmit signals. single transmission time should not exceed 10ms. | Transmission time averages 600us with some spikes to 5 ms. |
| 5 | Have one person continuously read the payload ID on the receiver side to determine if there are any missed transmission signals. | The receiver should be receiving almost all of the messages being sent. This can be observed since the payload ID’s for each transmission signal increases by a factor of 1. | Payload successfully increases by 1 on each id reliably. |

## 

### *Position Test Group Overview*

This test group handles the tests that require physical adjustments to the instrument.

#### *ECD423-T-004 Tilt Sensor Test*

The test will be performed by attaching the tilt sensor to the microphone stand and then tilting the rod of the stand to desired tilt level and seeing if the correct value is read. Light shaking will be applied to the rod also at each tilt position to make sure no false readings occur. The tilt sensor will be attached to a Teensy 4.0 which will read the sensor. The results of this test will determine how accurate the tilt sensor can be to detect current position.

| **#** | **Step-by-Step Operations** | **Expected Results** | Actual Results |
| --- | --- | --- | --- |
| 1 | Connect the tilt sensors to the Teensy 4.0 base station’s digital inputs, 3.3V output and ground. | No results expected as this is setup. |  |
| 2 | Tilt the microphone stands in a horizontal position. | Teensy outputs that the microphone stand is horizontal to the serial monitor. | Successful serial print |
| 3 | Lightly shake the microphone stand in the horizontal position. | Teensy outputs that the microphone stand is horizontal to the serial monitor. | Successful serial print |
| 4 | Tilt the microphone stand to the tilted position. | Teensy outputs that the microphone stand is tilted to the serial monitor. | Successful serial print |
| 5 | Lightly shake the microphone stand in the tilted position | Teensy outputs that the microphone stand is tilted to the serial monitor. | Successful serial print |
| 6 | Tilt the microphone stand to the vertical position. | Teensy outputs that the microphone stand is vertical to the serial monitor. | Successful serial print |
| 7 | Lightly shake the microphone stand in the vertical position. | Teensy outputs that the microphone stand is vertical to the serial monitor. | Successful serial print |

***ECD423-T-005 Rotary Encoder Test***

This test will test the accuracy of the rotary encoder. The rotary encoder will start from a known position and then have a counter on the microcontroller count if the rotary encoder turns left or right.

| **#** | **Step-by-Step Operations** | **Expected Results** | Actual Results |
| --- | --- | --- | --- |
| 1 | Connect the rotary encoder to 2 digital pins on the keypad microcontroller. | No results expected as this is setup. |  |
| 2 | Start the rotary encoder at one end of the rod and set the internal count inside the microcontroller to 0. | No results expected as this is setup. |  |
| 3 | Move the keypad to the other end of the rod. | The value on the internal counter should increase. | Rotary encoder value increased |
| 4 | Move the keypad back to its initial position | The value on the internal counter should decrease. | Rotary encoder value decreased |
| 5 | Evaluate the counter at the initial position. | The value on the internal counter should read 0. | Rotary encoder value read 0 |

#### *ECD423-T-006 Keypad Rotation Test*

This test will have contacts placed in each of the four positions on the baseplate and a reader of the contacts will be on the top plate. The reader will be connected to an Arduino Nano/Teensy to read the results. The results of this test will ensure that the rotation system is working correctly.

| **#** | **Step-by-Step Operations** | **Expected Results** | Actual Results |
| --- | --- | --- | --- |
| 1 | Connect rotation clutch contacts to 4 digital inputs on the Arduino Nano/Teensy. | No results expected as this is setup. |  |
| 2 | Rotate the clutch so that it is in the 1st position. | Arduino Nano/Teensy outputs that it is in the 1st position on the Serial Monitor. | Correct serial print |
| 3 | Rotate the clutch so that it is in the 2nd position. | Arduino Nano/Teensy outputs that it is in the 2nd position on the Serial Monitor. | Correct serial print |
| 4 | Rotate the clutch so that it is in the 3rd position. | Arduino Nano/Teensy outputs that it is in the 3rd position on the Serial Monitor. | Correct serial print |
| 5 | Rotate the clutch so that it is in the 4th position. | Arduino Nano/Teensy outputs that it is in the 4th position on the Serial Monitor. | Correct serial print |

#### *ECD423-T-007 Instrument Transition Test*

The test will utilize the established tilt sensors, rotary encoder, and keypad rotation mechanisms. The software will use the detected rod tilt, rod position, and keypad rotation to determine the correct instrument setting. The results of this test will ensure that the MIDI instrument is transitioning to the correct instrument.

| **#** | **Step-by-Step Operations** | **Expected Results** | Actual Results |
| --- | --- | --- | --- |
| 1 | Connect tilt sensor to Teensy 4.0 base station’s digital inputs, 3.3-volt output and ground and connect Rotation clutch contacts to 4 digital inputs on the Arduino Nano/Teensy. Maintain the keypads of the instrument in its horizontal position. | The default position of the instrument is the piano setting, and the instrument “Piano” will output on the serial monitor and the Element software will use the correct MIDI packages. | Correct serial print and MIDI channel on element |
| 2 | Invert the left-hand keypad and maintain the horizontal position of the rod. | The instrument name “Flute” will output on the serial monitor and the Element software will use the Flute MIDI packages. | Correct serial print and MIDI channel on element |
| 3 | Turn the rod vertically and lock the bottom keypad in its upright position. The top keypad will be free-sliding and upside-down. | The instrument name “Bass” will output on the serial monitor and the Element software will use the Bass MIDI packages. | Correct serial print and MIDI channel on element |
| 4 | The rod will maintain the vertical position. Rotate the upper keypad facing left and the bottom keypad facing right. Lock the keypads in place. | The instrument name “Bassoon” will output on the serial monitor and the Element software will use the Bassoon MIDI packages. | Did not implement a bassoon in the final design |
| 5 | Position the rod of the instrument at a 45 degree angle. The keypads will be close together in the middle of the rod. Rotate the left keypad to its default rotation, and invert the right keypad. | The instrument name “Clarinet” will be output on the serial monitor and the Element software will use the Clarinet MIDI packages. | Correct serial print and MIDI channel on element |
| 6 | Move the keypads further apart from the clarinet test’s position | The instrument name “Saxophone” will output on the serial monitor and the Element software will use the Saxophone MIDI packages. | Correct serial print and MIDI channel on element |

## 

### *Inputs Test Group Overview*

This test group handles the tests for all user inputs.

#### *ECD423-T-008 Variable Volume Test*

The test will have force sensitive resistors positioned under the keys. The force sensitive resistors are connected to the analog inputs of the Arduino Nano/Teensy, utilizing the multiplexer breakout boards for additional inputs.The force sensitive resistors will output a numerical value to the serial monitor. This numerical value will be divided into ten ranges and used by the Element software to control the variable volume. The results of this test will ensure that the MIDI instrument is capable of changing volume based on the force applied to the input keys.

| **#** | **Step-by-Step Operations** | **Expected Results** | Actual Results |
| --- | --- | --- | --- |
| 1 | Connect the keypads and force sensitive resistors to the Arduino Nano’s/Teensy’s 3.3-volt output, ground, and analog inputs. | The serial monitor will output 0 for each of the force sensitive resistors, as no force is being applied. | Correct serial print for all FSRs |
| 2 | Apply varying levels of force to each of the keys. | The serial monitor will output a value in the ten ranges from 0 to 1023 for the resistor being pressed. The instrument will play in the corresponding volume level. | As the FSRs were calibrated with different resistors than suggested by the manufacturer, this value generally ranged from 0 to 400 rather than 0 to 1023. The function remained the same. |

#### *ECD423-T-009 Lamellophone Key Test*

The test will use the lamellophone keys that extrude out of the piano keys. The lamellophone keys allow for a strumming and plucking motion for instruments that do not use a simple press. The lamellophone uses the same force sensitive resistors as the connected piano keys. The results of this test will ensure that the lamellophone keys can operate a force sensitive resistor.

| **#** | **Step-by-Step Operations** | **Expected Results** | Actual Results |
| --- | --- | --- | --- |
| 1 | Connect the keypad and force sensitive resistors to the Arduino Nano’s/Teensy’s 3.3-volt output, ground, and analog inputs. | The serial monitor will output 0 for each of the force sensitive resistors, as the lamellophone keys are in a neutral position. | Correct serial print |
| 2 | Configure into an instrument position that utilizes the lamellophone keys. | The serial monitor will output 0 for each of the force sensitive resistors, as the lamellophone keys are in a neutral position. | Correct serial print |
| 3 | Pluck the lamellophone keys with varying levels of force. | The serial monitor will output a value in the ten ranges from 0 to 1023 for the resistor being pressed. The instrument will play in the corresponding volume level. | As the FSRs were calibrated with different resistors than suggested by the manufacturer, this value generally ranged from 0 to 400 rather than 0 to 1023. The function remained the same. |

#### *ECD423-T-010 Joystick Test*

The test will use the two-axis joystick in its vertical axis. The joysticks are attached on the side of the keypads, positioned under the user's thumbs. The joysticks will be connected to the Arduino Nano/Teensy and the value of the position will be used by the Element software to control pitch bend. The results of this test will ensure that the MIDI instrument is capable of pitch bend.

| **#** | **Step-by-Step Operations** | **Expected Results** | Actual Results |
| --- | --- | --- | --- |
| 1 | Connect the joysticks to the Arduino Nano’s/Teensy’s 3.3-volt output, ground, and analog inputs. | The serial monitor will output approximately 512 at the resting neutral position. | Due to minor noise, the value was not always exactly 512, but within ±10 |
| 2 | Steadily move the joystick upwards. | The serial monitor will gradually output a higher number, reaching the maximum value of 1023 at the highest point. The pitch will bend up. | Successful pitch bend with correct serial print |
| 3 | Return to the neutral position. Steadily move the joystick downwards. | The serial monitor output will gradually decrease, reaching the minimum value of 0 at the lowest point. The pitch will bend down. | Successful pitch bend with correct serial print |

#### *ECD423-T-011 Final Test*

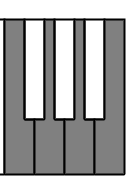
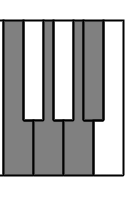
This evaluation is to demonstrate the functionality of the final product as a MIDI instrument capable of pitch bend, variable volume and polyphony. The user will perform a simple tune/song and alternate between at least 6 different instruments.

| **#** | **Step-by-Step Operations** | **Expected Results** | Actual Results |
| --- | --- | --- | --- |
| 1 | Connect the base station to a laptop with the synthesizer program, Element, already on. | No results expected as this is setup. |  |
| 2 | Attach the keypads and tilt detection system onto the rod and then attach the rod onto the mic stand. Adjust the height of the mic stand appropriately. | No results expected as this is setup. |  |
| 3 | Change between MIDI instruments by changing the orientation of the rod, positioning of the keypad and keypad rotation. | Pressing on the keys should emit a sound that is reminiscent of the particular instrument the device is set to. | Transition was successful. |
| 4 | Play a tune. | The device should sound similar to the actual instrument. | As a combination of previous tests, this was successful. |

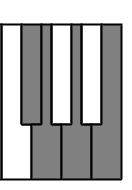
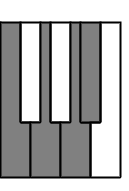
## Appendix G - Demonstration Saxophone Fingering Chart

**Saxophone**

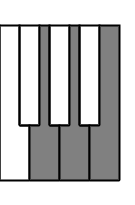
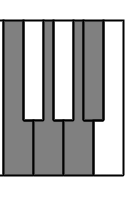
A#2



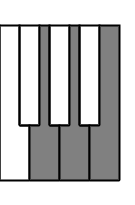
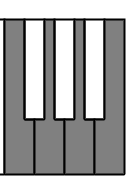
C#3



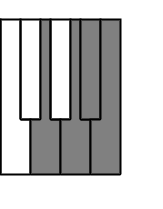
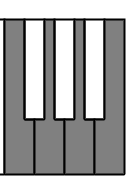
C3



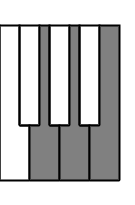
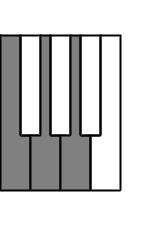
D#3



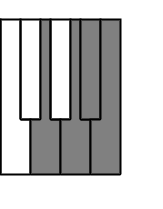
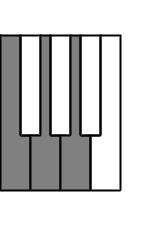
123 789 10 6 D#4



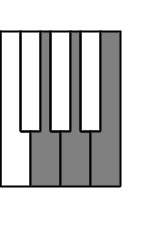
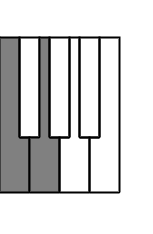
123 789 - > D3



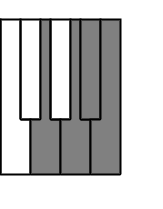
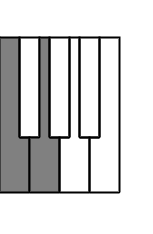
123 789 6 D4



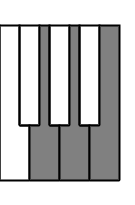
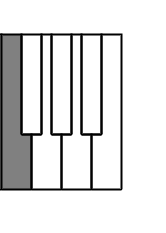
123 78 E3



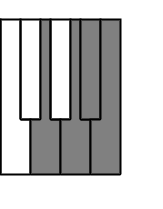
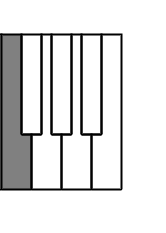
123 6 78 -> E4



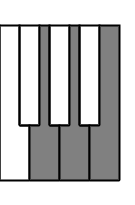
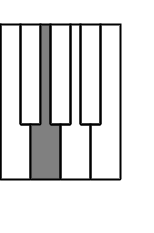
123 7 F3



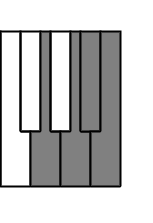
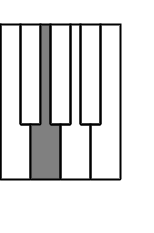
123 6 7 F4



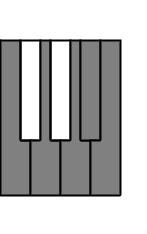
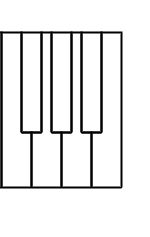
123 8 F#3



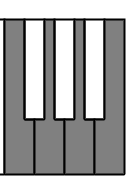
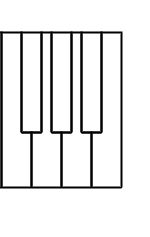
123 6 8 -> F#4



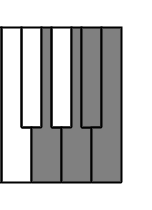
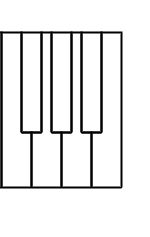
123 0 6 -> G#4



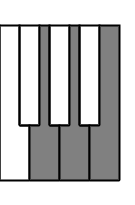
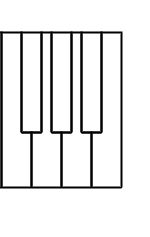
123 0 -> G#3



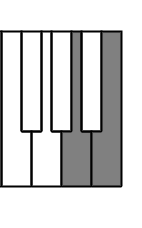
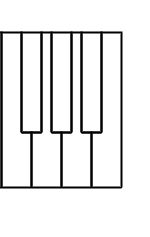
123 6 -> G4



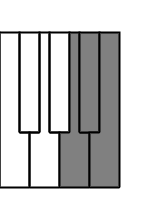
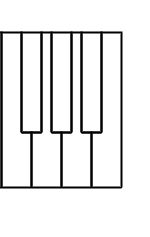
123 -> G3



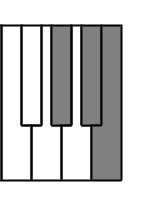
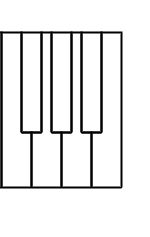
23 !6 -> A3



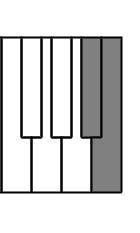
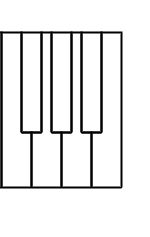
23 6 -> A4



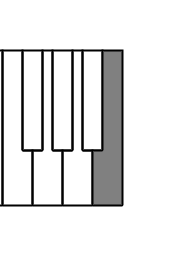
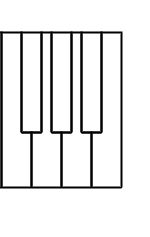
3 6 5 -> A#4



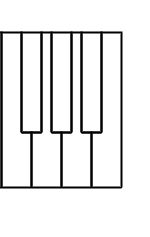
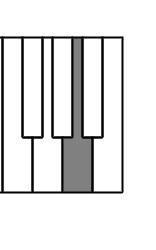
3 6 -> B4



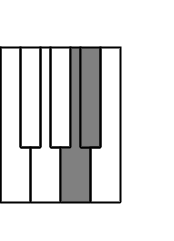
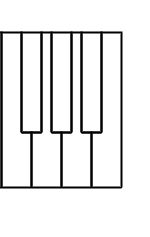
3 -> B3



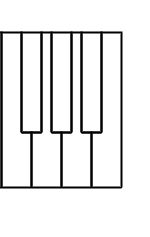
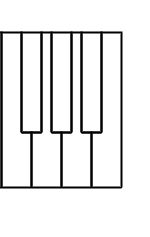
2 !6 -> C4

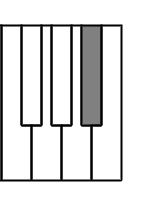
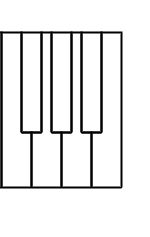
2 6 -> C5



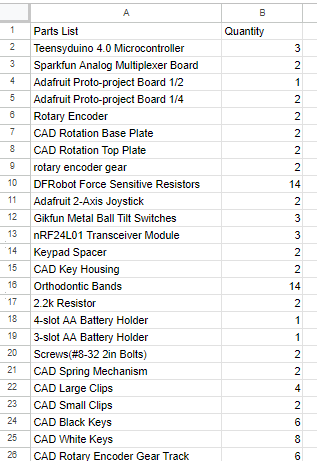
open C#4



6 C#5



## Appendix H - Parts List



## Appendix I - Project Standards

Introduction:

Adhering to Industry Standards for software and hardware projects allows for better readability, effective troubleshooting and reliability among other factors. For one, professionals with limited knowledge of the particular project will have an easier time understanding your code. With a better understanding, they can contribute relevant advice for your design. Abiding by industry standards lowers the possibility of warning messages and errors which may inhibit the progression of your project. Coding in a manner that follows programming language standards allows for the continuous improvement of the software in such a way that the project can still be relevant many years later. Ultimately, applying industry standards to our design process will only be beneficial and not harmful in the long run.

Hardware Engineering Standards:

We apply the IEEE 830 standard to all official documents in correlation with this project. These include the Integration and Test Plan, Project Specification Document as well as Final Report. According to this standard, all requirements are unambiguous and testable. The Final Report is divided into sections which include an Introduction, Problem Definition, System Design, Project Development, System Implementation, Project Evaluation, Notes and Appendices. Each test, requirement and stretch goal has a specific address associated with it for consistency. Feedback has been gathered in the form of advisor demonstrations, final demonstrations, final design presentations and more.

Wireless Communication Protocol:

The particular wireless communication protocol used is the nRF24L01 Specification which is utilized to ensure proper communication between the transmitter and receiver modules. Error handling was performed with built in functions to check the status of data signals with each step of the wireless communication protocol. This ultimately helped ensure reliable transmission of data. During the testing process, we were able to determine whether the transceiver was successfully transmitting, whether the receiver was successfully receiving as well as the contents of the data signal. By being able to do this, it was much easier to narrow down which particular component was causing an issue and how to solve it.

Programming Language Standards:

The Programming Language used for this project was ANSI C since Arduino IDE uses a variant of the C++ programming language. This standard was executed primarily to ensure that the proper data signals were sent through the network path to be able to manipulate data signals to control polyphony, variable volume, and pitch bend in the synthesizer. Through this standard we were able to ensure the effectiveness of debugging and keeping track of all variables used in the code. Functions were created with as much efficiency as possible. Team members were able to understand each other's code due to the structured method of documentation for each coding file.

## Appendix J - Requirements Impact Assessment

