# Fluid MIDI Ribbon Guitar

Noah Baker

Advisor: Wayne Pilkington

Senior Project

Electrical Engineering Department

California Polytechnic State University

San Luis Obispo, CA

2015

## Table of Contents



## Tables And Figures

**Abstract:**

This project was developed to create a new electronic instrument based off of a guitar. The design will give the user more chord combinations than are available on a typical guitar, the ability to retune each "string", and customizable MIDI controller outputs, while trying to retain a similar playing style. The initial design for this project included implementing an onboard synthesizer (analog or digital). This idea was scrapped for time and cost considerations and replaced with MIDI output which will yield user customizable sound through virtual instruments, such as Massive.

**Introduction:**

The purpose of this project is to develop a new affordable, customizable, intuitive guitar based instrument that outputs MIDI protocol. Through my initial market research, I found that no single MIDI instruments encompassed all the requirements I intended mine to have. Listed in Table 1 are all of the similar products that I could find information about.

Table 1: Similar Products

| Company | Product | Type | Cost |
|---|---|---|---|
| Sonuus [1] | G2M | Analog to Midi Converter | $99.00 |
| [2] | B2M | Analog to Midi Converter | $99.00 |
| Roland [3] | GK-2A | Pickup add-on | $219 |
| YouRockGuitar [4] | YRG-1000 (GEN2) | Hybrid | $249.00 |
| Yamaha [5] | EZ-AG | Button Fretboard | $200 (Discontinued) |
| [6] | EZ-EG | Button Fretboard | $300 (Discontinued) |
| MISA [7] | Kitara | Button Fretboard,<br>Touch LCD stum detector | $789.00 (Discontinued) |
| [8] | tri-bass | Capacitive Fretboard,<br>Touch LCD stum detector | $649.00 |
| Starr Labs' [9],[10] | Ztar (various styles) | Button Fretboard | $2500 to $5000 |

Misa, and Starr Labs' are the 2 leaders in MIDI Guitar market. Star Labs' is the most similar to my design, but costs much more than what most people can afford. My product is expected to retail for less than a third of the price.

The MIDI Guitar market is fairly small, but fluctuates quite a bit. Recently, there has been a surge in subgenres of electronic music. I expect this market to continue to expand over the coming years. Genres such as Chiptune, Electro Pop, Electronic New Wave, and Synthpop would benefit from access to more types of instruments. Artists in these Genres, and other electronic music genres, would benefit from alternative MIDI instruments, which could hasten and expand their song writing while improving their stage presence.

My company seeks to market to smaller bands and musicians who can't afford Starr Labs' equipment, but want a modern and flexible alternative to using keyboard MIDI Controllers.

My product will have the customizability of the Ztar, at a price slightly below the tri-bass, and uses a new type of fretboard technology, pioneered by David Levi's Magnetovore, and expanded upon on my test bench. The fretboard can detect the precise position of the users fingers, and pressure which the user is applying. All of the analog and digital inputs can be customized to control Velocity, Aftertouch, or Control Changes. Because it is being developed on the Arduino, the code will be released, so users can further tweak the functionality to suit their application.

The sooner I can build the prototype and get to market the better; the window of opportunity is open and does not seem to be closing very fast. Some other companies have been reportedly working on similar products, but I have not been able to find any hard information. I hope to have a final design by spring of next year, and begin marketing then.

To enter this market, my company needs to build a reliable prototype, and demo it to several interested parties to gain an initial investment for the first run of instruments. In addition to this, we would need to build a website, and create marketing material that would include pictures, and demonstrational videos. One possible way to get press and initial investments would be to run a Kickstarter, or Indiegogo campaign. To build an initial run of 25 instruments, we would need an investment of around $7500 ($300 to build each instrument). This would include the cost of components, and labor.

I have talked with David Levi on several occasions, and mutually expressed interest in partnering up in our Instrument creation ventures. David already has experience in the industry and created a company around his invention, the Magnetovore.

I have several friends who play guitar, as well as make electronic music. I will be working closely with them during the testing and feedback phase of my design. Their feedback will decide what increased functionality will be added to the final design, and what functions can be removed.

The current solutions are either too expensive, lack functionality, are cheaply built, or are discontinued.

The neck of instrument will have 4 ‘strings’, each composed of 2 variable resistors; one to detect location, and one to detect pressure, or force. The body of the instrument shall have both a capacitive beat pad, and a strum detector for multiple types of note strikes. The guitar will also have an character LCD to display current string and chord settings, as well as indicator LED’s.

The advantages of my product include increased chord complexity (when compared to a standard guitar, pickups, analog to midi converters, or hybrid MIDI instruments), logarithmic continuous fretboard (in place of buttons), and endless customizability through buttons and switches on the instrument, as well as access to the source code. My instrument offers doubled simultaneous MIDI voices at an entry level cost.

**Requirements and Specifications:**

Table 2: Requirements and Specifications

| Market Requirements | Engineering Specification | Justification |
|---|---|---|
| 1. | The final design must cost less than $300, including labor, when put into production. | This will ensure that the product will be affordable to most musicians. |
| 2. | The device must be compatible with all MIDI 1.0 protocol. [11] [12] | Reports of incompatible devices will scare away prospective customers. |
| 3. | The linear resistors must be able to detect the locations of the users fingers within 3% accuracy without excess force (greater than 1N). [13] [16] | This will ensure that the instrument is not difficult to play, or susceptible to excess force which could cause damages. |
| 4. | The force sensitive resistor must have at least 400 levels of usable pressure. [15] | This will ensure continuity of the MIDI AfterTouch, or control change (depending on user settings) |
| 5. | The strings and keypad must be reprogrammable on the fly. | Retuning will allow users to quickly make presets and find new chords for their music. |
| 6. | The instrument must have not more than 40ms of latency. | Low latency will make the instrument more playable and feel more natural. |

Market Requirements:

1. Low cost compared to competitors ($500-$750 selling price)
2. MIDI 1.0 Compliant
3. Accurate Fret Spacing
4. Continuous Linear Control Signals
5. Retunable on the fly
6. Low Latency (>40ms)

I expect to spend at least 100 hours programming the microcontroller and adding features. At $20 an hour, there will be an additional initial cost of $2000 on top of hardware. Updating the firmware could also add to the cost, but I don't anticipate lengthy changes to the code after launch. The bulk cost of components is also much less. Many components are nearly half the price when purchased in bulk.

**Met Requirements:**

1. Low cost compared to competitors ($500-$750 selling price)

   The final prototype cost only $269.57 for all the components and required hardware. This cost will be further reduced and labor streamlined if put into production.

2. MIDI 1.0 Compliant

   The MIDI functionality which I implemented is able to handle nearly all MIDI messages. The only signals it currently does not support are fine adjustment control signal changes which require a single extra byte (4 instead of 3). This is a relatively simple function to write but has not been implemented yet and isn't supported by all controllers.

3. Accurate Fret Spacing

   The Fret spacing on my instrument was measured from a Fender Stratocaster, a very widely played electric guitar. The lookup table is accurate to 0.5mm at 10 Bit resolution.

4. Continuous Linear Control Signals

   The instrument can be set to operate in fretless mode, but this only works with some VSTI's.

5. Retunable on the fly

   By pressing both buttons at the same time, the user can retune the last string played to anything by spinning the control channel knob.

6. Low Latency (>40ms)

The latency between pressing a capacitive pad and playing a not is 15ms, as measured in software and with an Oscilloscope by probing the discharge pin and the MIDI Buffer.

Table 3: Prototyping Costs

| **Required Devices** | **Initial Device Selection** | **#** | **Each** | **$ Total** |
|---|---|---|---|---|
| Location Sensor | 500mm ThinPot (Spectra Symbol) | 4 | 16.95 | 67.8 |
| Pressure Sensor | 24” ForceSense Resistor (Interlink) | 4 | 17.95 | 71.8 |
| Strum Sensor | 100mm SoftPot | 1 | 6.95 | 6.95 |
| PCB | For Resistors, and Capacitive Pads | 1 | 19.95 | 19.95 |
| Mode Selection Buttons | Pushbutton (momentary) | 4 | 2 | 8 |
| Note Selection Buttons | Pushbutton (momentary) | 8 | 2 | 16 |
| Processing Unit | Arduino Due | 1 | 50 | 50 |
| Guitar Body/Neck | Handmade | 1 | 100 | 100 |
| Labor | | 100 | 20 | 2000 |
| **Total Cost** | **-** | **-** | **-** | **$ 2340.50** |

Table 3: Final Design Costs

| **Required Devices** | **Final Selection** | **#** | **Each** | **$ Total** |
|---|---|---|---|---|
| Location Sensor | 500mm ThinPot (Spectra Symbol) | 4 | 13.95 | 55.80 |
| Pressure Sensor | 24” ForceSense Resistor (Interlink) | 4 | 14.70 | 58.80 |
| Driver Chip | CD4069UBE | 1 | 0.37 | 0.37 |
| End Pins | Planet Wave Elliptical | 1 | 7.59 | 7.59 |
| Screws | Universal | 12 | .05 | 0.60 |
| PCB 1 | For Resistors, and Capacitive Pads | 1 | 31.66 | 31.66 |
| PCB 2 | Capacitive Touchpad | 1 | 31.66 | 31.66 |
| PCB Standoffs | Generic Plastic | 1 | 4.95 | 4.95 |

| | | | | |
|---|---|---|---|---|
| MIDI Din Connector | 7 pin Female Chassis Mt. Din Connector | 1 | 2.40 | 2.40 |
| Wires | JST Jumper 5 Wire Assembly | 7 | 1.50 | 10.50 |
| Pickguard Material | Musiclily 4Ply Pearl Green 11x17 | 1/2 | 16.97 | 8.46 |
| Mode Selection Buttons | Pushbutton (momentary) | 2 | 1.95 | 3.90 |
| Resistors | (various) | 30 | 0.05 | 1.50 |
| Capacitor (debounce) | 100nF | 3 | 0.05 | 0.15 |
| Output Driver | CD4069UBE | 1 | 0.52 | 0.52 |
| Potentiometer | 10K Linear | 2 | 1.95 | 3.90 |
| Processing Unit | Arduino Due | 1 | 20.95 | 20.95 |
| Guitar Neck | Handmade .75”x2.5”x3’ | 1 | 5.50 | 5.50 |
| Guitar Body | Handmade .75”x5.5”x1’ | 2 | 7.23 | 14.46 |
| Body Coating | Aerosol Lacquer | 1 | 6.79 | 6.79 |
| **Material Cost** | - | - | - | **$ 269.90** |
| Labor | - | 8 | 20 | 160.00 |
| **Total Cost** | **-** | **-** | **-** | **$ 429.90** |

**Preliminary Design and Alternative Discussion:**

The Initial design for the neck of the instrument uses 4 linear resistors (ThinPot) and 4 force sensitive resistors (FSR). The number of linear resistors was chosen based on the width of the force sensitive resistors; If the initial tests of the early prototype revealed no or limited user desire for force sensing capabilities, 5 or 6 linear resistors can be placed on the neck while retaining a reasonable wide, However, the FSR’s proved to be useful for not only chanel pressure, and control channel changes, but also glitch mitigation. This change would also reduce component component cost as well as manufacturing complexibility. This would save at least $25 per instrument.

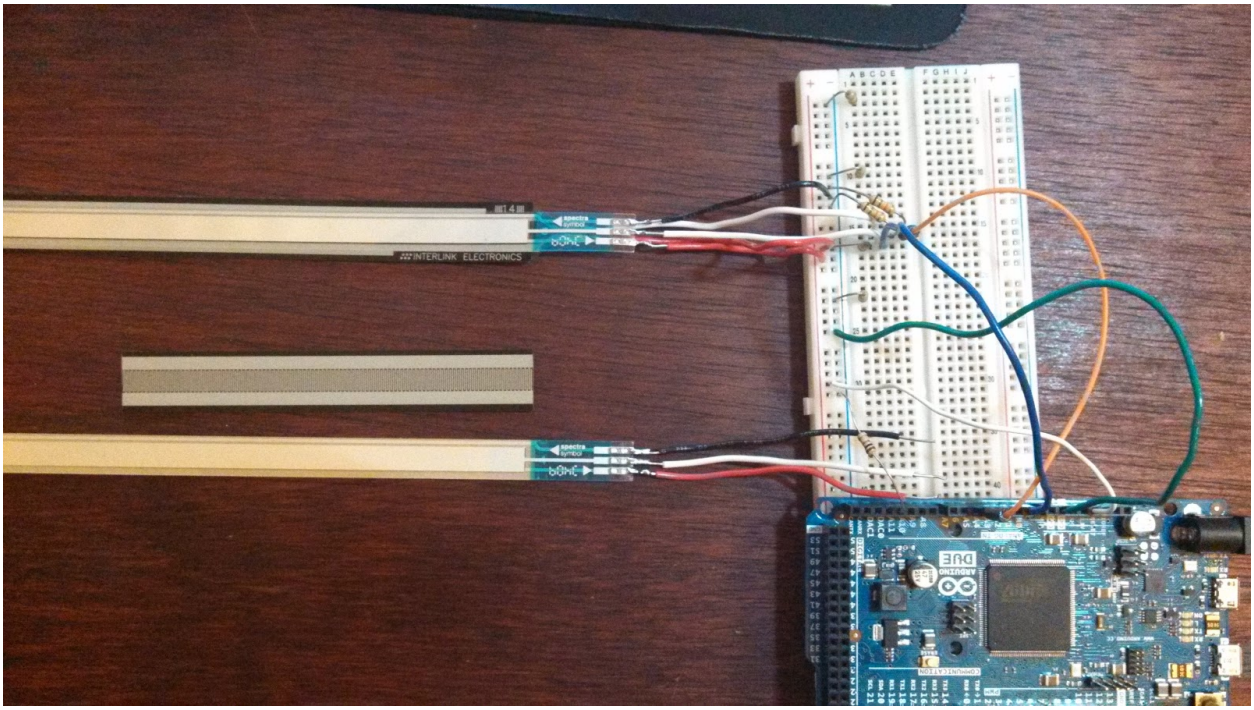



Figure 1: Testing of Linear and Force Sense Ribbons

Initial testing revealed numerous problems with the MPR121 Capacitive Keypad. One of the major problems is the lack of a complete $I^2C$/TWI Library for the Arduino Due. The Arduino Library lacks a Repeated Start Bit condition, which hampers access to registers on the MPR121. This lack of Repeated Start Bit also means that multiple devices can not be used on the TWI, restricting the use of multiple MPR121's, and $I^2C$ LCD's. To properly integrate the MPR121, i would need to write my own $I^2C$ Library which would take quite a bit of time due to limited documentation of the SAM3x TWI. The size and layout of the buttons on the keypad is also not ideal for my application. The keys are slightly too small, too close together and not ergonomic for the user. Alternative designs are shown in Figure 2 below.

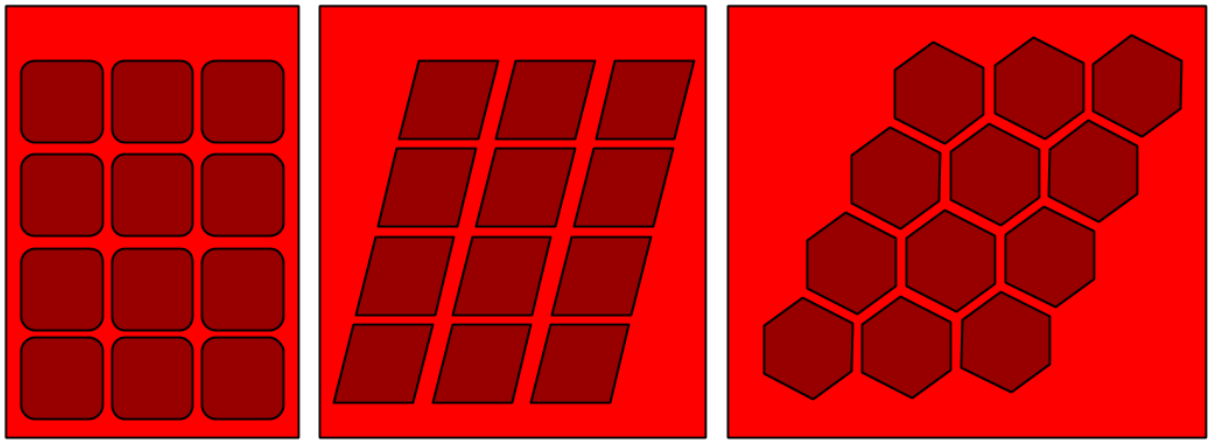



Figure2: Alternative Button Layouts

After some research, I found that instead of using the MPR121 Capacitive touch sensor, it is possible to use digital pins on the Arduino DUE to directly sense capacitive touch inputs. There is also an open source library which has already been written to quickly implement this feature. This design alternative could vastly reduce the cost and complexity of the device and expand customizability. Testers also noted that they would like to have an additional Capacitive button set.

The final design for the button layout is shown in Figure 3. This layout is 50% larger than the original layout, and features a 4th row of capacitive pads, bringing the total number of capacitive buttons up to 16. The method for reading these capacitive pads is as follows; Charge the pad up with one digital pin, begin discharging the pad with another digital pin through a sufficiently large resistor, and see how long it takes to fully discharge. The way I have implemented this uses 2 pins for each sensor, a total of 32 pins, although, if these are needed for other interfacing, one pin could be used as the discharge pin for all of the capacitive pads.

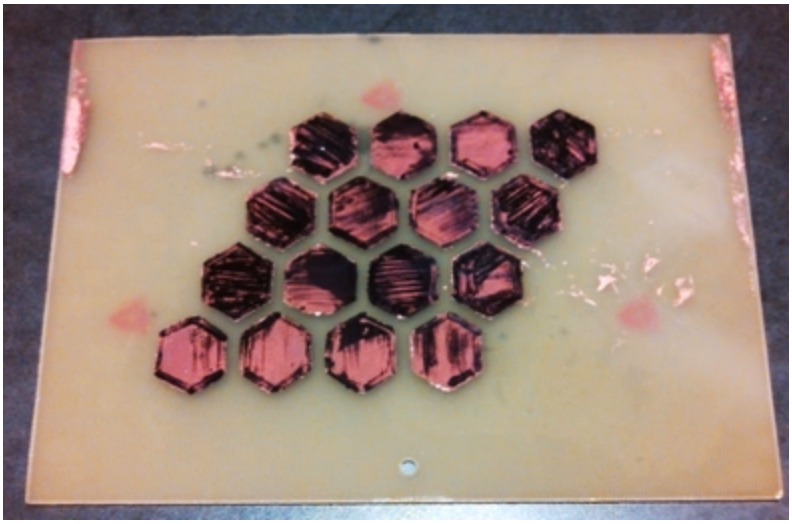


Figure 3: Capacitive Pad Prototype

The initial body design features crisp straight lines and a sharp figure as shown in Figure 4. It was built using a router and 2 jigs, which could be reused for a small production run. Initial testers had mixed reviews about the style of the body; some felt it was “pretty cool”, while others wanted a more traditional curvy shape, like a Fender Stratocaster. Further information will be gathered before a final design is decided upon and jig made.

Figure 4: Initial Prototype

The initial neck was cut from a 0.75” thick piece of wood. Two 0.5” 45$^{o}$ chamfers were cut on either side, however, upon playing on it for a while most users commented that they would like to see a more curved neck, and possibly thinner. The next design will have a rounded neck much closer to that of a standard guitar.

The Arduino Due was initially selected for its speed, number of analog inputs and ease of use. It, However, is one of the more expensive microcontrollers. If the FSR’s are removed, only 8 analog inputs will be needed, which could expand the options of possible microcontrollers. Below, in Table 4, alternative microcontrollers are shown with the relevant specifications.

Table 4: Alternative Microcontrollers

| Microcontroller | Price $ (lowest) | Analog Pins | Digital Pins |
|---|---|---|---|
| Arduino Due | 20.50 | 12 | 54 |
| Arduino Mega2560 | 10.08 | 16 | 54 |

While testing the prototype over USB, it occurred to me that many people will be converting the MIDI DIN to USB for use anyway. USB MIDI can use a much higher baud rate and drastically reduce latency of the controller. Being able to use DIN or USB would be inexpensive to implement and enhance user experience. The prototype is currently set up to send serial data from the programing port at a high baud rate, as well as out of its DIN jack at the standard 31250 baud rate for MIDI instruments.

After implementing the MIDI interface, I discovered that the MIDI to usb converter I was using would not recognize any MIDI signals if i was not connected to USB power. I found that the MIDI specification is reliant on current and the arduino due’s 3.3V digital pin did not supply the proper 5mA. To solve this issue, I used a pair of inverters from a hex inverter out of an old lab kit. The solution to this in the final design will be a single buffer.

By working on a side project, I familiarized myself with the V-USB Library. This library can be used to create virtual USB devices on Atmel AVR processors. While this library is not currently compatible with the due, it may be a future step in the device development to further expand compatibility, and features of the controller.

**Final Prototype:**

The biggest difficulty to overcome playing the initial prototype was the bad ergonomic design of the original neck. The second revision of the neck was initially cut with a router in a similar method to the first design, but was then sanded with a random orbit sander to create a smooth ergonomic grip allowing the user to reach each ribbon without strain. The revised neck is much easier to play on. The design was reviewed by 2 guitar players throughout its manufacturing process to ensure an ergonomic design

Figure 5: Initial Neck Design (Left) Final Neck Design (Right)

When building the final body design, I wanted to increase the simplicity of manufacturing and reduce weight and cost. The final version of the body was much easier to make than the original. The final square body design, which measures 1.5”x5.5”x1’, is a third of the weight of the original design.

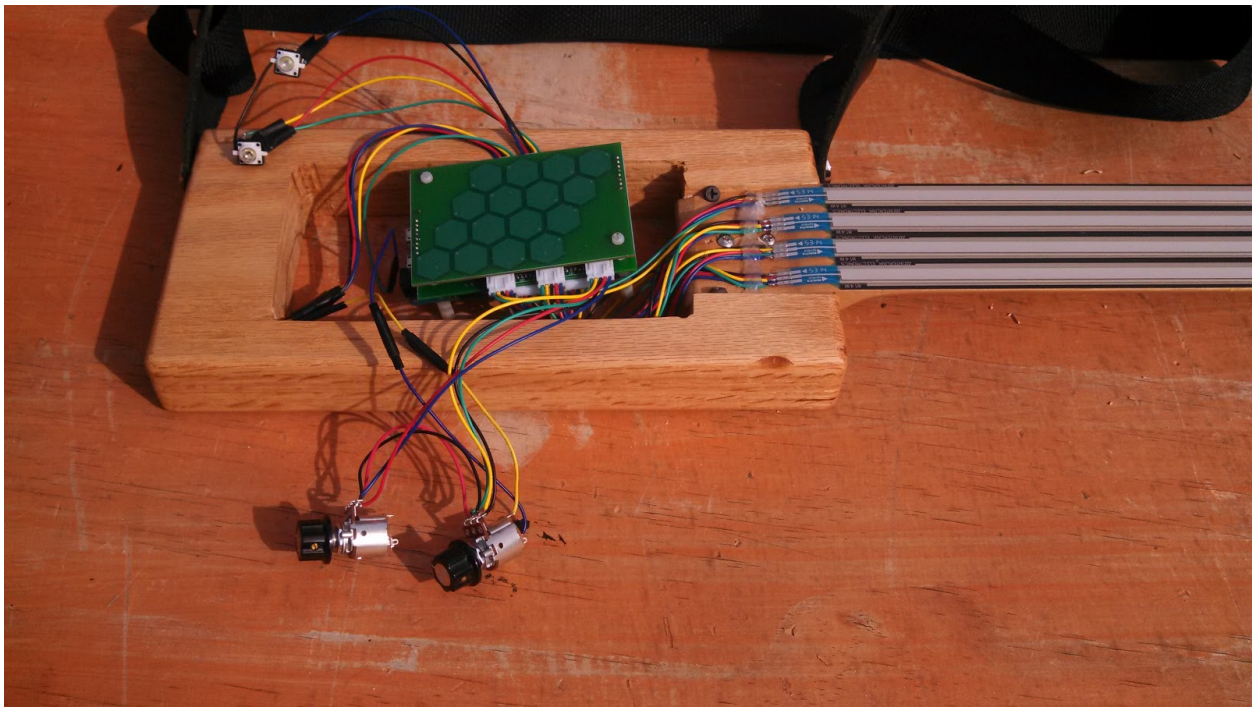


Figure 6: Final Prototype Body and Internals Before Assembly

I decided to continue to use most of the same hardware when building the final prototype , with the addition of another button and 2 rotary potentiometers for controlling volume and control channel changes. To mount and hold the components I selected a green pearl pickguard to be on the front and back of the instrument.  This makes assembly quick, and looks sleek once assembled.

Figure 7: Final Assembled Prototype (Front and Back)

In addition to these extra components, I traced and manufactured 2 PCB's; One shield for the Arduino Due, and one set of capacitive buttons. Both of these boards stack and include holes for plastic standoffs to add structural stability.

After receiving the boards, I noticed several problems. There were a few mistakes on first printing of PCB. One of the headers on the Arduino Due was spaced non uniformly compared to the rest, which I overlooked, and was 0.025" off when I received the board. The header pin through holes were also slightly too small for the header I was using.

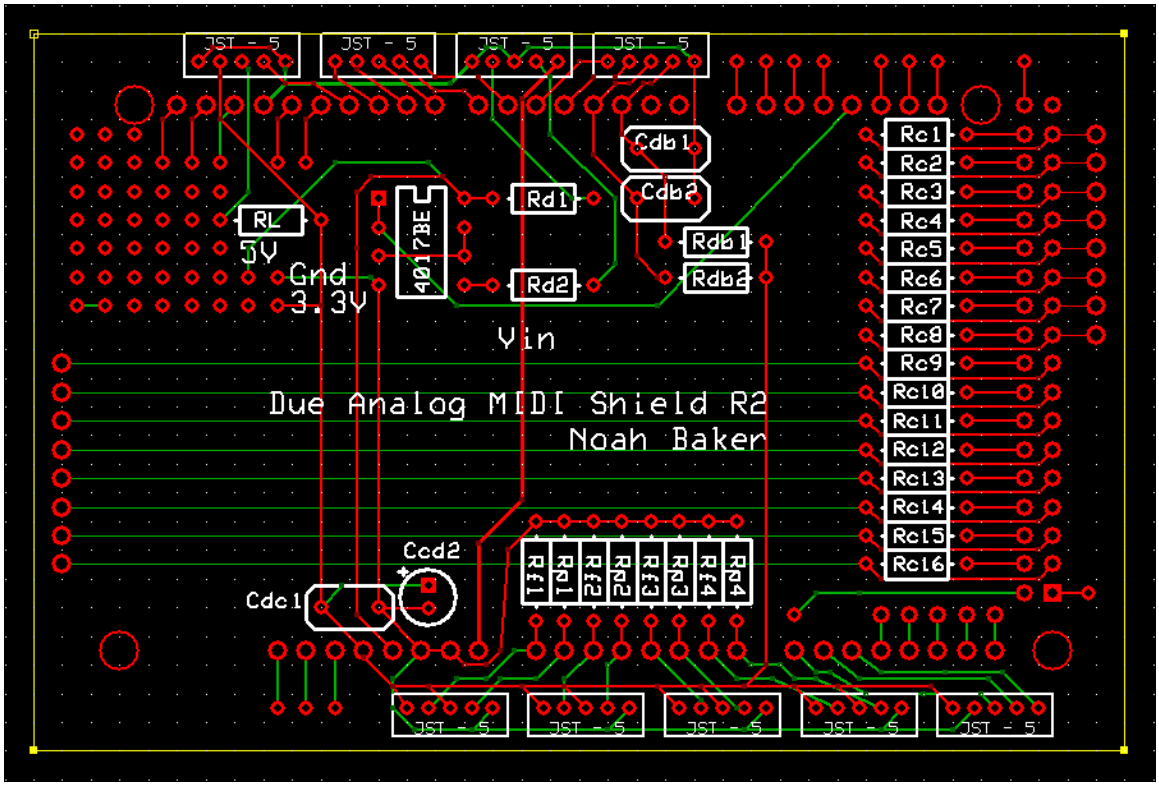



Figure 8: Bottom Shield

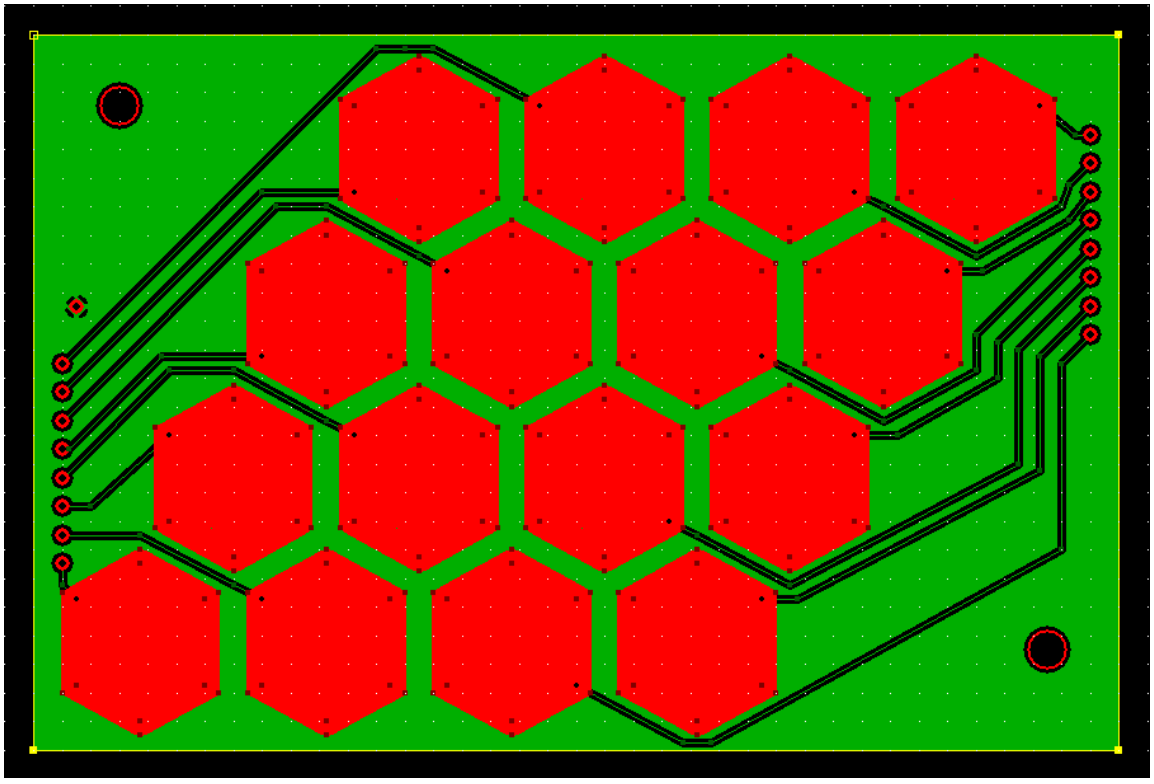


Figure 9: Top Shield with Capacitive Buttons

I was able to work around these issues and use the boards, but corrected the issues on the new design are shown above.

When selecting the driver chip to buffer the midi signals, I used mousers search function and selected a CD40107BE (comprised of 2 NAND Buffers) After installing and testing the chip, however, i found that no transitions were being made. Upon reviewing the datasheet again I noticed that $V_{OL}$ was 3.5V, not 3.0V, as I thought it was. To remedy this, i switched back to using a CD4069UBE, which I had used in the first prototype.

**Functional Decomposition:**

The level 0 block diagram, in Figure 10, shows the flow of analog, digital, and supply inputs, as well as the MIDI, LCD, and indicator LED outputs. Table 5 Describes each input and output in detail.

Figure 10: Level 0 Block Diagram


Table 5: Level 0 Block Diagram Table

| | Name | Description |
|---|---|---|
| Input | 9V Power Supply | DC power will be supplied from a powered MIDI hub over the 2 extra pins on the DIN connection. |
| | Analog Ribbons, Potentiometers | These Analog inputs come from the ForceSense, ThinPot variable resistors and Potentiometers. |

| | | |
|---|---|---|
| | Buttons, Capacitive Pads | These digital inputs come from the momentary push buttons, and the capacitive keypad. [17] |
| Output | 5 pin MIDI DIN Cable | Digital MIDI data sent according to protocol over a DIN 5 cable |
| | USB Serial (Debug) | This allows for debugging of the hardware and testing of new functionality via the serial monitor. |
| | Indicator LED's | Onboard LED's indicating various String and keypad modes. |

Figure 11 shows the level 1 block diagram of the instrument. This diagram shows the basic components on the Arduino Due, the required external components, and their data flow.

Figure 11: Level 1 Block Diagram

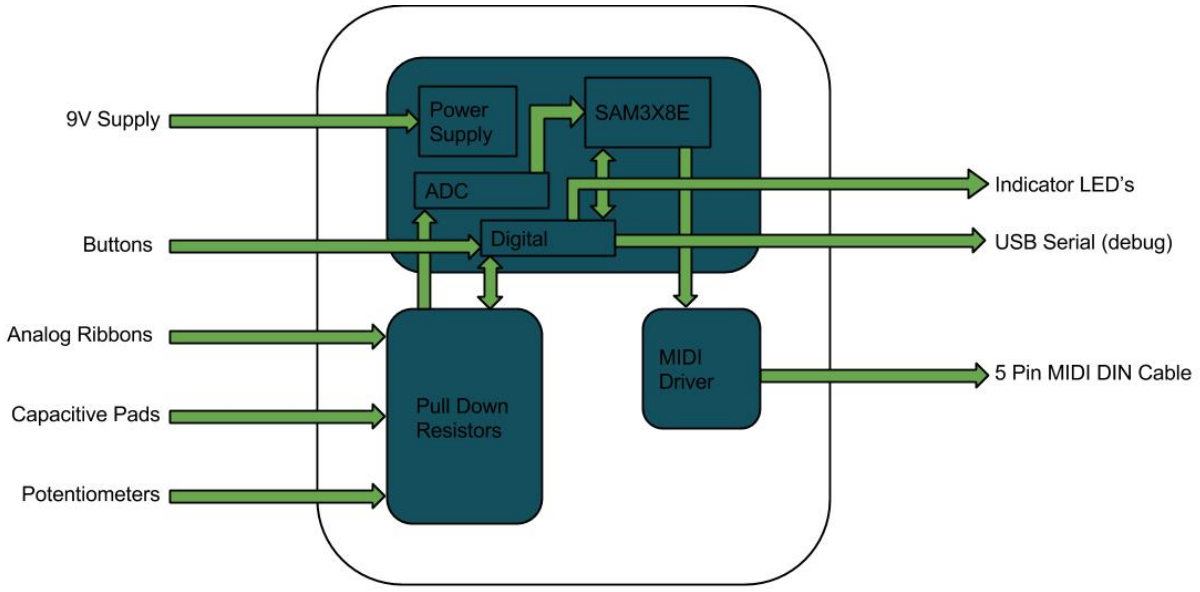
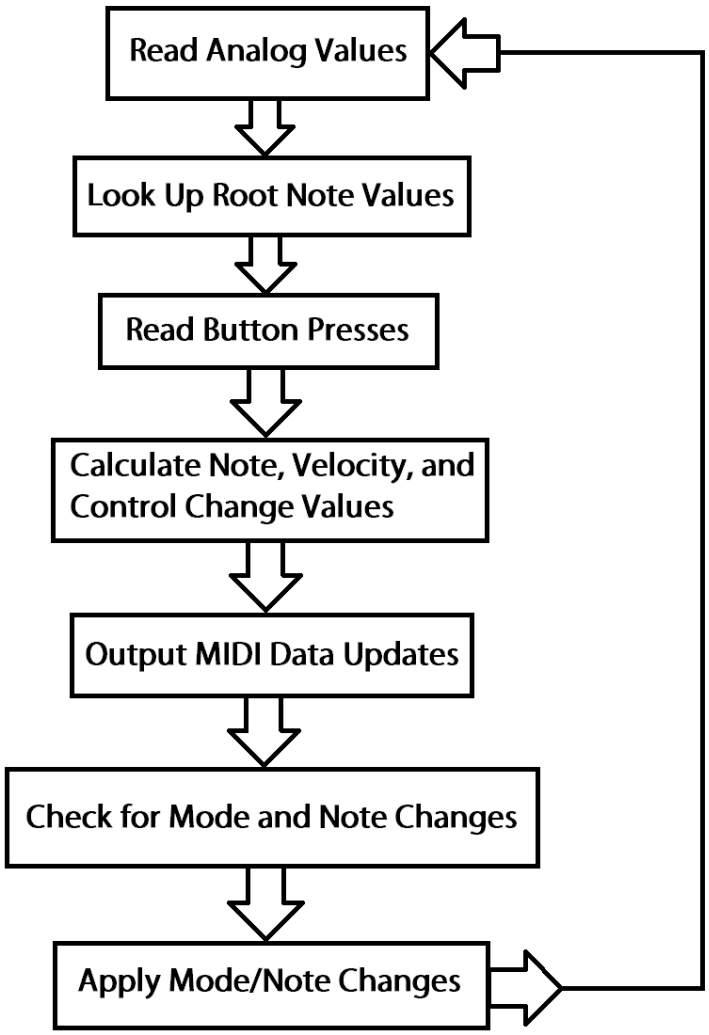



Table 6: Level 1 Block Diagram Table

| | Name | Description |
|---|---|---|
| Components | Arduino Due | The Arduino Due was selected as the microcontroller for its speed, number analog inputs, and number of digital outputs. The Due will regulate power for the digital pins, convert the analog values to digital, Process the |

| | | |
|---|---|---|
| | | information for user inputs and send the MIDI signals. |
| | ADC | The built in ADC on the Due will convert all of the analog values to digital. |
| | Digital | The digital pins on the Due will receive button presses, key presses, and send data to the LCD and MIDI output. |
| | ARM Processor | The ARM processor on the due will process all of the user inputs and compute the MIDI data to be sent out. |
| | Pull-Up/Pull-Down Resistors | These resistors will debounce digital inputs, regulate current on the MIDI output, and assist in the reading of analog voltage values. |

Figure 12: System Software Flowchart



Testing and verification of Protocol was done with several MIDI compliant devices, such as controllers, and MIDI to USB hubs. Latency was be measured with an oscilloscope monitoring the MIDI data pin, and output of the Capacitive keypad, as well as in software. Currently, the controller has been tested on 4 different USB to MIDI interface devices.

Automatic Chord Generation:

The initial chord generation algorithm developed to use with my MIDI controller builds triads based off of the root note of the first string (Default tuning is E). Currently, the programed scales include: Major, Natural Minor, Harmonic Minor, Melodic Minor, and Octave. For example, if the user is in Standard tuning, holding a G major chord, with the controller in Natural minor mode, a G Major, B Minor, D Major and G Major +1 chord will be generated as shown in Table 7.

Table 7: Valid Automatic Chord

| Automatic Triad Generation | | | | | |
|---|---|---|---|---|---|
| E Natural Minor | Natural Minor | root | | | |
| Em tonic | 0 | **0** | **3** | **7** | Minor |
| F#dim supertonic | 2 | **2** | **5** | **8** | Dim |
| GM mediant | 3 | **3** | **7** | **10** | Major |
| Am subdominant | 5 | **5** | **8** | **12** | Minor |
| Bm dominant | 7 | **7** | **10** | **14** | Minor |
| CM submediant | 8 | **8** | **12** | **15** | Major |
| DM subtonic | 10 | **10** | **14** | **17** | Major |
| | | | | | |
| | Generated Notes | | | | |
| G Major Chord in E Natural Minor | Root | | | user defined | |
| (3) 3 | **3** | **7** | **10** | x | **G Major** |
| (2) 7 | **7** | **10** | **14** | x | **B Minor** |
| (0) 10 | **10** | **14** | **17** | x | **D Major** |

| | | | | | |
|---|---|---|---|---|---|
| (0) 15 | **15** | **19** | **22** | x | **G Major (+1)** |
| | **G Major** | **B Minor** | **D Major** | | |

This initial implementation ignores generating chords for notes not in the scale, but could later on be changed to generate a user defined chord for these accidentals. The downfall of the current automatic triad generation is shown in Table 8. As can be seen here, this chord is not in the key signature and lacks many automatically generated chords. Furthermore, misplayed chords from incorrect finger position will also result in undesired results, unless automatic proximity assisting is implemented.

Table 8: Invalid Automatic Chord

| Automatic Triad Generation | | | | | |
|---|---|---|---|---|---|
| E Melodic Minor | Melodic Minor | root | | | |
| Em tonic | 0 | **0** | **3** | **7** | Minor |
| F#m supertonic | 2 | **2** | **5** | **9** | Minor |
| Gaug mediant | 3 | **3** | **7** | **11** | Aug |
| AM subdominant | 5 | **5** | **9** | **12** | Major |
| BM dominant | 7 | **7** | **11** | **14** | Major |
| C#dim submediant | 9 | **9** | **12** | **15** | Dim |
| D#dim subtonic | 11 | **11** | **14** | **17** | Dim |
| | | | | | |
| | Generated Notes | | | | |
| G Major Chord in E Melodic Minor | Root | | | user defined | |
| (3) 3 | **3** | **7** | **11** | x | **G Aug** |
| (2) 7 | **7** | **11** | **14** | x | **B Major** |
| (0) 10 | **10** | **10** | **10** | x | **D** |
| (0) 15 | **15** | **19** | **23** | x | **G Aug +1** |
| | **G Major** | | | | |

User Defined Chord Generation:

One proposed solution to create a seamless automatic chord generator was suggested by a fellow Engineer, Michael Twardochleb. His proposed method lets the user select which types of chords they want to generate based off the combination of buttons that they press. This will greatly expand the possible number of chords that could be played without adjusting settings. This mode has not been fully developed and is not functional at the time of writing this report.

Smart Chord Generation (Following):

The final advanced generation function I made for this project is a smart generation which keeps track of the most often recently played notes. Based off of this it can currently build common triads from the 7 most common notes. If the user switches his chord progression, It will quickly change within the time the user plays a few chords.

Noise and Glitch Mitigation:

There were 2 main causes of glitches in the initial implementation. The first was that when a string is being pressed, the voltage level on the analog pin rises to meet the value, and is not instantaneous. This caused intermediate notes to trigger when pressing higher up on the neck. To account for this, 3 samples are taken for each ribbon. If the samples don't closely match, then it resamples the analog values. There is also a minimum force required to initiate the “note on” command which removes the remainder of the glitches. The second, more expected, glitch occurs when the user places his/her finger directly on the transition (fret) of a note. This will cause the controller to bounce back and forth between the two and cause stuttering. To mitigate this, a ‘fret width’ was added which acts as a schmitt trigger, requiring the user to move past the area to initiate the next note.

Seamless Pitch Bend:

The MIDI protocol only allows for 2 semitones (notes) of pitch bend. It is also limited to bending the entire channel, and can not bend individual notes. In order to implement a fretless mode, multiple things must be calculated and taken into account. Firstly, the notes decrease in size in a

logarithmic fashion, however, because of the number of notes, bending linearly based on the relative position of the users finger on a note yields a pseudo-logarithmic bend, like classical stringed instruments. Each string must also output their data to separate channels because of the lack of functionality in the protocol. Lastly, the fret width of the glitch mitigation causes problems in calculating the pitch bend, creating a very complex function required to have a continuous sweep.

**Code:**

Below is the code as it is on June 1st 2015. More commenting is needed, and individual functions need to be made for each calculation in order to simplify the understanding for users wishing to modify the functionality of their instrument. A header file for constants will also be made once all glitches have been removed and the functionality will not change drastically. The new LCD/user interface code was removed to be rewritten, and is not yet done.

```
/*Fluid MIDI Ribbon Guitar
 *Noah Baker
 *Senior Project Spring 2015
 */

#include <CapacitiveSensor.h>
#include <stdlib.h>

//array of capacitive pads
CapacitiveSensor pad[4][4] = {
      {CapacitiveSensor(22,23),CapacitiveSensor(24,25),CapacitiveSensor(38,39),CapacitiveSensor(40,41)},
      {CapacitiveSensor(26,27),CapacitiveSensor(28,29),CapacitiveSensor(42,43),CapacitiveSensor(44,45)},
      {CapacitiveSensor(30,31),CapacitiveSensor(32,33),CapacitiveSensor(46,47),CapacitiveSensor(48,49)},
      {CapacitiveSensor(34,35),CapacitiveSensor(36,37),CapacitiveSensor(50,51),CapacitiveSensor(52,14)}
};

//analog resistance values (frets)
int markers [23] = {11,78,152,222,288,350,409,464,517,566,612,656,698,737,774,809,842,873,903,930,956,981,1004};
int softpot [4] = {A1,A3,A5,A7}; //softpot analog values
int FSR [4] = {A0,A2,A4,A6}; //fsr analog values

char root [4] = {0x28,0x2D,0x32,0x37}; //E2 as root note, to be set later by button input, tuned as bass
char softpot_previous_note [4] = {0x28,0x2D,0x32,0x37};
int FretlessEnabled = 0; //Changes weather or not fretless mode is enabled
int OffFirst = 0;        //changes weather or not MIDI notes are turned off before turned on when sliding
int FretWidth =5;        //Width of the frets
int total=0;             //Number of buttons pressed
int lastpressed = 0; //last string pressed

char softpot_note [4] = {0,0,0,0};
int FretlessPitch[4] = {0,0,0,0}; //pitch shifting for fretless mode
int RawPadData [4][4] = {
```

```
      {0, 0, 0, 0},
      {0, 0, 0, 0},
      {0, 0, 0, 0},
      {0, 0, 0, 0}
};
char MIDI_notes [4][4] = {
      {0, 0, 0, 0},
      {0, 0, 0, 0},
      {0, 0, 0, 0},
      {0, 0, 0, 0}
};//stores the midi values for the notes
char MIDI_notesp [4][4] = {
      {0, 0, 0, 0},
      {0, 0, 0, 0},
      {0, 0, 0, 0},
      {0, 0, 0, 0}
}; //stores the previous midi values for the notes
int PadButtonsPrevious [4][4] = {
      {0, 0, 0, 0},
      {0, 0, 0, 0},
      {0, 0, 0, 0},
      {0, 0, 0, 0}
}; //previous values on capacitive buttons
int PadButtonsCurrent [4][4]  = {
      {0, 0, 0, 0},
      {0, 0, 0, 0},
      {0, 0, 0, 0},
      {0, 0, 0, 0}
};
int PadNoteMask [4][4] = {
      {0, 0, 0, 0},
      {0, 0, 0, 0},
      {0, 0, 0, 0},
      {0, 0, 0, 0}
};//initially 0
int softpot_reading [4][4] = {  //softpot reading, #of reading, or final value (3)
      {0, 0, 0, 0},
      {0, 0, 0, 0},
      {0, 0, 0, 0},
      {0, 0, 0, 0}
};

int RecentNotes[12] = {0,0,0,0,0,0,0,0,0,0,0,0};
int RecentNoteIndex = 0;
int MinMax = -1;
int MinMaxIndex = -1;
int ScaleNoteIndex = 0;
int ScaleNotes[7] = {0,0,0,0,0,0,0};

int MajorNoteMask [4] = {0,4,7,11};//initially major changes based off pad settings
int MinorNoteMask [4] = {0,3,7,10};
int AugNoteMask [4] = {0,4,8,10};
int DimNoteMask [4] = {0,3,6,9};
int OctaveMask [4] = {0,12,24,36};

char NoteOff [4] = {0x80,0x81,0x82,0x83};
char NoteOn [4] = {0x90,0x91,0x92,0x93};
```

```
char NoteForce [4] = {0xA0,0xA1,0xA2,0xA3}; //aftertouch, individual pressure value
char ControlChange [4] = {0xB0,0xB1,0xB2,0xB3}; //for other sliders etc.
char NoteForceAvg [4] = {0xD0,0xD1,0xD2,0xD3}; //aftertouch, avg pressure value
char PitchBend [4] = {0xE0,0xE1,0xE2,0xE3};

int FSR_reading [4] = {0, 0, 0, 0}; //FSR reading, #of reading, or final value (3)

int scale_mode = 5;
int scale_modep = 4;
int ControlDial = 0;
int ControlDialP = 0;
volatile unsigned long microspsm;
volatile unsigned long microspfm;
int irqPin = 22;  // D22

void UpdateNote(char Status, char Note, char Velocity); //for NoteOff, NoteOn, NoteForce

/*Setup Function*/
void setup(){
	Serial1.begin(31250);
	Serial.begin(115200);
	//set up buttons for scale types
	pinMode(5, OUTPUT);
	pinMode(2, INPUT_PULLUP);
	attachInterrupt(2,scale_modebutton,FALLING); //attaches pin 2 to mode button interrupt
	//set up button for engaging Fretless mode
	pinMode(4, OUTPUT);
	pinMode(3, INPUT_PULLUP);
	attachInterrupt(3,Fretless_modebutton,FALLING); //attaches pin 50 to mode button interrupt
	//enable interrupts
	interrupts();
	SetupPads();
}

/*Main Loop*/
void loop(){
	int ndx;
	int pad_column;
	int string;
	int checknote;

	//gets analog values
	ControlDialP = ControlDial;
	ControlDial = (analogRead(A9) >>3);
	int Volume = (analogRead(A8) >>3);
	for(string=0;string<4;string++){
		do{
			analogRead(A10); //clears mux voltage
			softpot_reading[string][0] = analogRead(softpot[string]);
			analogRead(A10); //clears mux voltage
			softpot_reading[string][1] = analogRead(softpot[string]);
			analogRead(A10); //clears mux voltage
			softpot_reading[string][2] = analogRead(softpot[string]);
			softpot_reading[string][3]=((softpot_reading[string][0]+softpot_reading[string][1]+softpot_reading[string][2])/3);
		}while((abs(softpot_reading[string][3] - softpot_reading[string][1]) >2) & (abs(softpot_reading[string][3] -
		softpot_reading[string][0]) >2) & (abs(softpot_reading[string][3] - softpot_reading[string][2]) >2));
		//clears mux voltage
```

```
        analogRead(A10);
        //reads FSR allows for a maximum pitch bend of 1 semitone (12288) at max pressure
        FSR_reading[string]  = ((analogRead(FSR[string])*analogRead(FSR[string]))/90);
        if(FSR_reading[string] < 8192){FSR_reading[string] =8192;}//if lower than neutral position, then set to neutral

        analogRead(A10);//clears mux voltage
}

//determines the current frets being played
for(string=0;string<4;string++){
        noInterrupts();
        softpot_note[string] = root[string];
        ndx = 0;
        while ( ndx <23 && (softpot_reading[string][3] > markers[ndx])){ //adds fret number to root note
                ndx++;
                softpot_note[string]++;
        }
        //ignores press if user is not pressing hard enough, this greatly reduces glitches when changing notes
        if(FSR_reading[string] < 3000){

        }
        //accounts for analog noise, acts a a Schmitt trigger when user is on a fret
        if((softpot_note[string] == softpot_previous_note[string] - 1)&&(softpot_reading[string][3] > markers[ndx]-FretWidth)){
        //falling
                softpot_note[string] = softpot_previous_note[string];
                FretlessPitch[string] = ((markers[ndx+1] - softpot_reading[string][3])*4096 / (markers[ndx+1] - markers[ndx]));
        }
        else if((softpot_note[string]==softpot_previous_note[string]+1)&&(softpot_reading[string][3]<markers[ndx-1]-FretWidth)){
        //rising
                softpot_note[string] = softpot_previous_note[string];
                FretlessPitch[string] = ((markers[ndx-1] - softpot_reading[string][3])*4096 / (markers[ndx] - markers[ndx-1]));
        }
        else if(FSR_reading[string] >= 3472 && softpot_note[string] != root[string]){
                FretlessPitch[string] = ((markers[ndx] - softpot_reading[string][3])*4096 / (markers[ndx] - markers[ndx-1]));
        }
        else{
                FretlessPitch[string] = 0;
        }
        //calculates a % of a semitone to shift the note below the fret by to give a fretless effect
        //FretlessPitch[string] = ((markers[ndx] - softpot_reading[string][3])*4096 / (markers[ndx] - markers[ndx-1]));
        //records the value of the note played
        softpot_previous_note[string] = softpot_note[string];
        interrupts();
}



//Processes the notes for each string and sends them
for(string=0;string<4;string++){
        //updates modulation if it has changed
        if(ControlDial != ControlDialP){
                UpdateNote(ControlChange[string], 0x01, ControlDial);
        }

        //gets touch inputs from keypad
        readTouchInputs(string);
```

```
//determines notes being held, relative to 1 (root note)
checknote = softpot_note[string] - root[0] ; //major based on root note of first string
while(checknote > 11){
        checknote -= 12;
}

//checks to see if scale_mode has changed, and sets new PadNoteMask if it has

pad_column=0;
//noInterrupts();
while(pad_column <4){
        switch(scale_mode){
        case 0:
                switch(checknote){ //Major scale root note chord generation
                        case 0:
                                PadNoteMask[string][pad_column] = MajorNoteMask[pad_column];
                                break;
                        case 1:
                                PadNoteMask[string][pad_column] = OctaveMask[pad_column];
                                break;
                        case 2:
                                PadNoteMask[string][pad_column] = MinorNoteMask[pad_column];
                                break;
                        case 3:
                                PadNoteMask[string][pad_column] = OctaveMask[pad_column];
                                break;
                        case 4:
                                PadNoteMask[string][pad_column] = MinorNoteMask[pad_column];
                                break;
                        case 5:
                                PadNoteMask[string][pad_column] = MajorNoteMask[pad_column];
                                break;
                        case 6:
                                PadNoteMask[string][pad_column] = OctaveMask[pad_column];
                                break;
                        case 7:
                                PadNoteMask[string][pad_column] = MajorNoteMask[pad_column];
                                break;
                        case 8:
                                PadNoteMask[string][pad_column] = OctaveMask[pad_column];
                                break;
                        case 9:
                                PadNoteMask[string][pad_column] = MinorNoteMask[pad_column];
                                break;
                        case 10:
                                PadNoteMask[string][pad_column] = OctaveMask[pad_column];
                                break;
                        case 11:
                                PadNoteMask[string][pad_column] = DimNoteMask[pad_column];
                                break;
                }
        break;
        case 1:
                switch(checknote){ //Natural Minor scale root note chord generation
                        case 0:
                                PadNoteMask[string][pad_column] = MinorNoteMask[pad_column];
                                break;
```

```
                case 1:
                        PadNoteMask[string][pad_column] = OctaveMask[pad_column];
                        break;
                case 2:
                        PadNoteMask[string][pad_column] = DimNoteMask[pad_column];
                        break;
                case 3:
                        PadNoteMask[string][pad_column] = MajorNoteMask[pad_column];
                        break;
                case 4:
                        PadNoteMask[string][pad_column] = OctaveMask[pad_column];
                        break;
                case 5:
                        PadNoteMask[string][pad_column] = MinorNoteMask[pad_column];
                        break;
                case 6:
                        PadNoteMask[string][pad_column] = OctaveMask[pad_column];
                        break;
                case 7:
                        PadNoteMask[string][pad_column] = MinorNoteMask[pad_column];
                        break;
                case 8:
                        PadNoteMask[string][pad_column] = MajorNoteMask[pad_column];
                        break;
                case 9:
                        PadNoteMask[string][pad_column] = OctaveMask[pad_column];
                        break;
                case 10:
                        PadNoteMask[string][pad_column] = MajorNoteMask[pad_column];
                        break;
                case 11:
                        PadNoteMask[string][pad_column] = OctaveMask[pad_column];
                        break;
        }
break;
case 2:
        switch(checknote){ //Harmonic Minor scale root note chord generation
                case 0:
                        PadNoteMask[string][pad_column] = MinorNoteMask[pad_column];
                        break;
                case 1:
                        PadNoteMask[string][pad_column] = OctaveMask[pad_column];
                        break;
                case 2:
                        PadNoteMask[string][pad_column] = DimNoteMask[pad_column];
                        break;
                case 3:
                        PadNoteMask[string][pad_column] = AugNoteMask[pad_column];
                        break;
                case 4:
                        PadNoteMask[string][pad_column] = OctaveMask[pad_column];
                        break;
                case 5:
                        PadNoteMask[string][pad_column] = MinorNoteMask[pad_column];
                        break;
                case 6:
                        PadNoteMask[string][pad_column] = OctaveMask[pad_column];
```

```
                        break;
                case 7:
                        PadNoteMask[string][pad_column] = MajorNoteMask[pad_column];
                        break;
                case 8:
                        PadNoteMask[string][pad_column] = MajorNoteMask[pad_column];
                        break;
                case 9:
                        PadNoteMask[string][pad_column] = OctaveMask[pad_column];
                        break;
                case 10:
                        PadNoteMask[string][pad_column] = OctaveMask[pad_column];
                        break;
                case 11:
                        PadNoteMask[string][pad_column] = DimNoteMask[pad_column];
                        break;
        }
break;
case 3:
        switch(checknote){ //Melodic Minor scale root note chord generation
                case 0:
                        PadNoteMask[string][pad_column] = MinorNoteMask[pad_column];
                        break;
                case 1:
                        PadNoteMask[string][pad_column] = OctaveMask[pad_column];
                        break;
                case 2:
                        PadNoteMask[string][pad_column] = MinorNoteMask[pad_column];
                        break;
                case 3:
                        PadNoteMask[string][pad_column] = AugNoteMask[pad_column];
                        break;
                case 4:
                        PadNoteMask[string][pad_column] = OctaveMask[pad_column];
                        break;
                case 5:
                        PadNoteMask[string][pad_column] = MajorNoteMask[pad_column];
                        break;
                case 6:
                        PadNoteMask[string][pad_column] = OctaveMask[pad_column];
                        break;
                case 7:
                        PadNoteMask[string][pad_column] = MajorNoteMask[pad_column];
                        break;
                case 8:
                        PadNoteMask[string][pad_column] = OctaveMask[pad_column];
                        break;
                case 9:
                        PadNoteMask[string][pad_column] = DimNoteMask[pad_column];
                        break;
                case 10:
                        PadNoteMask[string][pad_column] = OctaveMask[pad_column];
                        break;
                case 11:
                        PadNoteMask[string][pad_column] = DimNoteMask[pad_column];
                        break;
        }
```

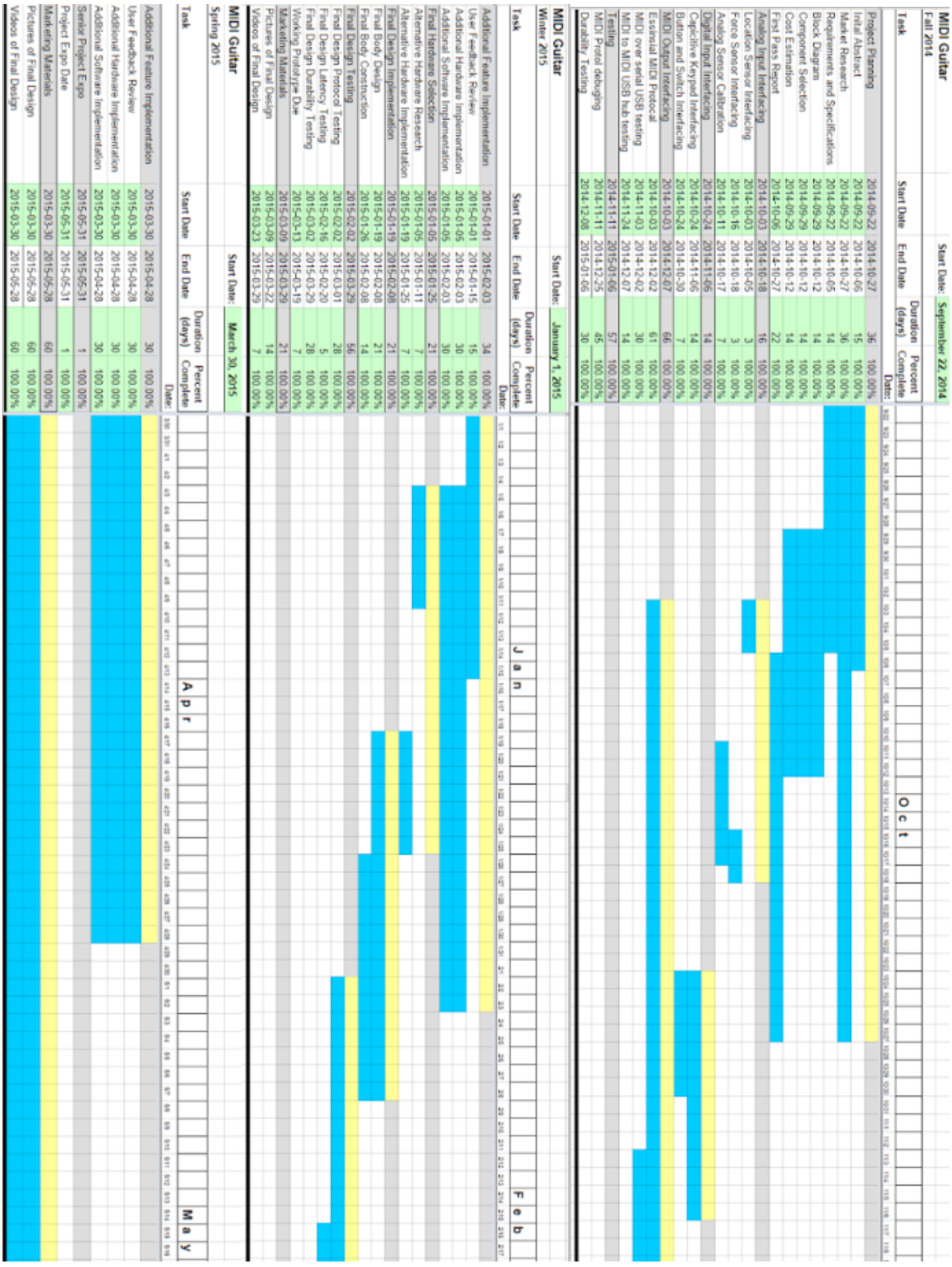
```
break;
case 4: //octave mode
        PadNoteMask[string][pad_column] = OctaveMask[pad_column];
break;
case 5: //learning mode
        //finds most commonly played 7 notes from the array of recent notes once per ribbon
        if(PadButtonsCurrent[string][0] == 1 && pad_column == 0){
                RecentNotes[checknote]++;
                ScaleNoteIndex = 0;
                MinMax = 20;
                for(RecentNoteIndex = 0; RecentNoteIndex < 12; RecentNoteIndex++){

                        //fills up the possible notes with first 7 and finds the min
                        if(ScaleNoteIndex < 7){
                                ScaleNotes[ScaleNoteIndex] = RecentNoteIndex;
                                if(RecentNotes[RecentNoteIndex] < MinMax){
                                        MinMax = RecentNotes[RecentNoteIndex];
                                        MinMaxIndex = ScaleNoteIndex;
                                }
                                ScaleNoteIndex++;
                        }

                        //looks through the remaining values to find greater values than minmax
                        else if(RecentNotes[RecentNoteIndex] > MinMax){
                                //pulls the smallest value out and shifts the rest
                                ScaleNoteIndex = MinMaxIndex;
                                while(ScaleNoteIndex < 6){
                                        ScaleNotes[ScaleNoteIndex] = ScaleNotes[ScaleNoteIndex+1];
                                        ScaleNoteIndex++;
                                }
                                //saves the most recently found note to the scale
                                ScaleNotes[ScaleNoteIndex] = RecentNoteIndex;
                                ScaleNoteIndex++;
                                //find new minmax
                                MinMax = RecentNotes[ScaleNotes[0]];
                                MinMaxIndex = 0;
                                for(ScaleNoteIndex = 1; ScaleNoteIndex <7; ScaleNoteIndex++){
                                        if(RecentNotes[(ScaleNotes[ScaleNoteIndex])] < MinMax){
                                                MinMax = RecentNotes[(ScaleNotes[ScaleNoteIndex])];
                                                MinMaxIndex = ScaleNoteIndex;
                                        }
                                }
                        }
                }

                //checks the smallest value to see it its over 100 and divides the recent note index if it is
                if(RecentNotes[(ScaleNotes[MinMaxIndex])] > 100){
                for(RecentNoteIndex = 0; RecentNoteIndex <12; RecentNoteIndex++){
                RecentNotes[RecentNoteIndex] /=2;
                }
                }

        }
        //update the pad note mask every cycle
        for(ScaleNoteIndex = 0; ScaleNoteIndex <7; ScaleNoteIndex++){
                if(checknote == ScaleNotes[ScaleNoteIndex]){
                        PadNoteMask[string][pad_column] = ScaleNotes[(pad_column * 2)];
```

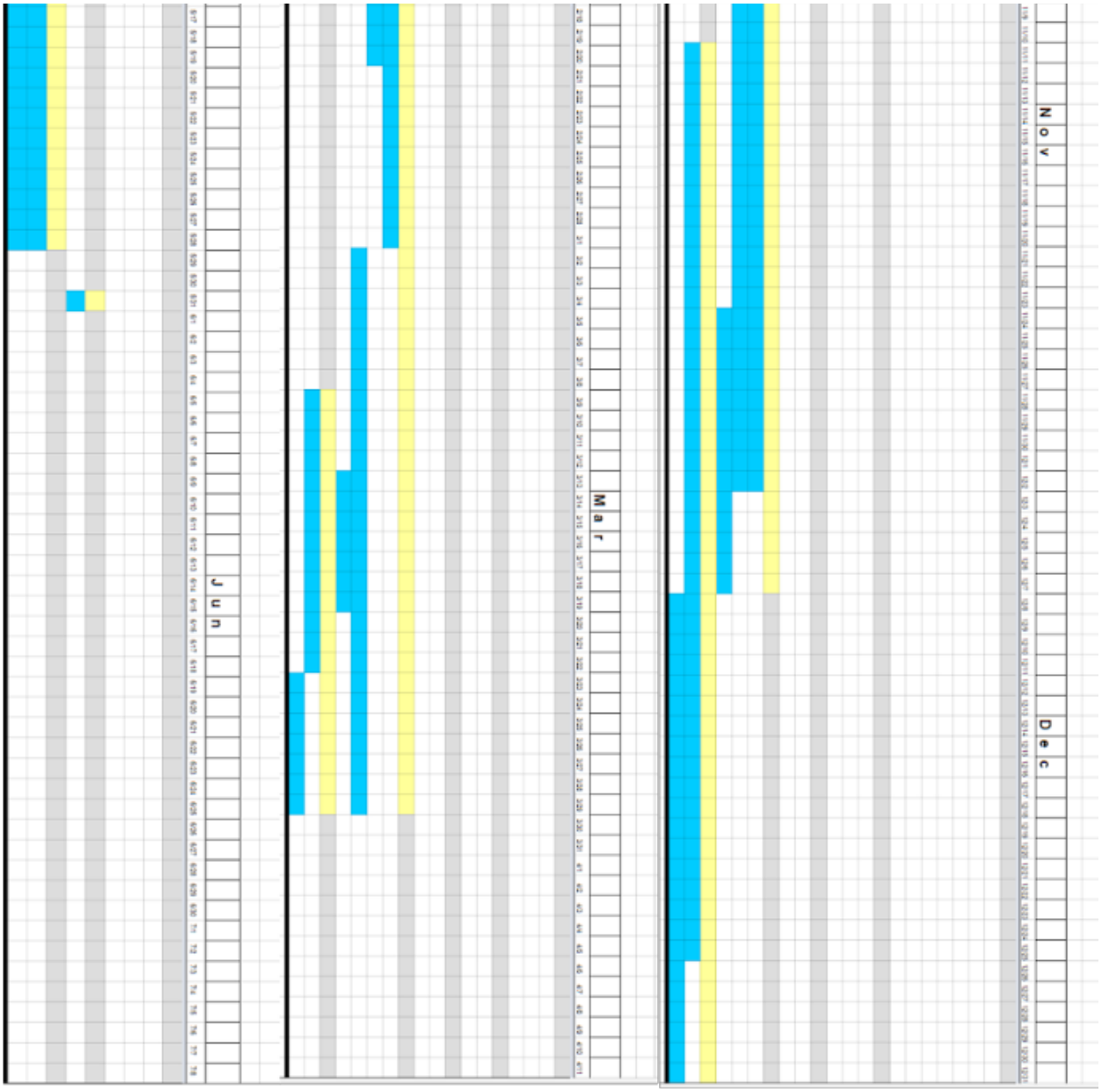
```
                                        break;
                                }
                                else{
                                        PadNoteMask[string][pad_column] = 0;
                                }
                        }
                break;
                case 6: //Standard Mode
                        PadNoteMask[string][0] = 0;
                        PadButtonsCurrent[string][0] |= PadButtonsCurrent[string][pad_column];
                break;
                }
                pad_column++;

        }
        //interrupts();
        //if lower than neutral position, then set to neutral
        if(FSR_reading[string] < 8192){FSR_reading[string] =8192;}
        // if fretless pitch is enabled, the string is pitch shifted down by a % of the semitone
        if(FretlessEnabled == 1){
                FSR_reading[string] -= FretlessPitch[string];
                FretWidth =7;
        }

        //sets MIDI notes based on string and note mask, if note has changed turns off previous note
        pad_column=0;
        while(pad_column <4){
                MIDI_notes[string][pad_column] = softpot_note[string] + PadNoteMask[string][pad_column];
                if((MIDI_notesp[string][pad_column] != MIDI_notes[string][pad_column])){
                        if(OffFirst == 1){UpdateNote(NoteOff[string],MIDI_notesp[string][pad_column],Volume);}
                }
                pad_column++;
        }

        //sends MIDI Note data
        pad_column=0;
        while(pad_column <4){
                //turns a note on if new button is pressed, or button has remained pressed but string changed
                if((PadButtonsCurrent[string][pad_column] > PadButtonsPrevious[string][pad_column]) ||
                ((MIDI_notesp[string][pad_column]!=MIDI_notes[string][pad_column])&&(PadButtonsCurrent[string][pad_column]>0))){
                        UpdateNote(NoteOn[string],MIDI_notes[string][pad_column],Volume);
                        lastpressed = string; //saves the last string pressed
                        if((MIDI_notesp[string][pad_column] != MIDI_notes[string][pad_column])){
                                if(OffFirst == 0){UpdateNote(NoteOff[string],MIDI_notesp[string][pad_column],Volume);}
                        }
                        UpdateNote(PitchBend[string], (FSR_reading[string] & 0x7F),((FSR_reading[string]>>7)& 0x7F));
                }
                else if(PadButtonsPrevious[string][pad_column] > PadButtonsCurrent[string][pad_column]){
                        UpdateNote(NoteOff[string],MIDI_notes[string][pad_column],Volume);
                        UpdateNote(PitchBend[string], (FSR_reading[string] & 0x7F),((FSR_reading[string]>>7)& 0x7F));
                }
                else if(PadButtonsCurrent[string][pad_column] >0){
                        //UpdateNote(NoteForce[string],MIDI_notes[string][pad_column],0x50);
                        //sends pitch bend update
                        UpdateNote(PitchBend[string], (FSR_reading[string] & 0x7F),((FSR_reading[string]>>7)& 0x7F));
                }
                PadButtonsPrevious[string][pad_column] = PadButtonsCurrent[string][pad_column];
```

```
                        MIDI_notesp[string][pad_column] = MIDI_notes[string][pad_column];
                        pad_column++;
                        //if in single mode only go through once
                        if(scale_mode == 6){pad_column =4;}
                }
        }
}


//checks if capacitive buttons have been pressed
void readTouchInputs(int string){
 int pad_column = 0;
 for(pad_column=0;pad_column<4;pad_column++){
   RawPadData[string][pad_column] = pad[string][pad_column].capacitiveSensorRaw(3);
   if(RawPadData[string][pad_column] == -2){
     PadButtonsCurrent[string][pad_column] = 1;
   }
   else{
     PadButtonsCurrent[string][pad_column] = 0;
     total++;
   }
 }
 if (total == 16){
   //solves not off not sent glitches
   UpdateNote(ControlChange[string], 0x7B, 0x00);
 }
 if(string == 3){total = 0;}
}

//sets up the capacitive buttons with unique time-outs
void SetupPads(){
       int string = 0;
       int pad_column = 0;
       for(string=0;string<4;string++){
                for(pad_column=0;pad_column<4;pad_column++){
                        pad[string][pad_column].set_CS_Timeout_Millis((((double)
pad[string][pad_column].capacitiveSensorRaw(3))/1200.0));
                }
       }
}

//updates midi information
void UpdateNote(char Status, char Note, char Velocity){
       digitalWrite(3, HIGH);
       Serial1.write(Status);
       Serial1.write(Note);
       Serial1.write(Velocity);
       Serial.write(Status);
       Serial.write(Note);
       Serial.write(Velocity);
       digitalWrite(3, LOW);
}

//changes the chord generation function, mode indicated by the led
void scale_modebutton(){
       noInterrupts();
       if((long)(micros() - microspsm) >= 600000 && digitalRead(3) == HIGH){
```

```
                scale_mode++;
                if(scale_mode > 6){
                        scale_mode = 0;
                }
                microspsm = micros();
                for(int i = 0; i <= scale_mode; i++){
                        digitalWrite(4, LOW);
                        delayMicroseconds(100000);
                        digitalWrite(4, HIGH);
                        delayMicroseconds(100000);
                }
        }

        //if both buttons are pressed lets the user set the root note of the last string played
        else if(digitalRead(3) == LOW){
                while(digitalRead(3) == LOW){
                        ControlDial = ((analogRead(A9) + analogRead(A9)) >>4);
                        UpdateNote(NoteOff[lastpressed],root[lastpressed],0x7F);
                        root[lastpressed] = ControlDial;
                        UpdateNote(NoteOn[lastpressed],root[lastpressed],0x7F);
                        softpot_previous_note[lastpressed] = ControlDial;
                        delayMicroseconds(500000);
                }
        }
        interrupts();
}

//activates or deactivates fretless mode, indicated by the LED
void Fretless_modebutton(){
        noInterrupts();
        if((long)(micros() - microspfm) >= 600000){
                if(FretlessEnabled == 0){
                        FretlessEnabled =1;
                        digitalWrite(5, HIGH);
                }
        else{
                FretlessEnabled =0;
                digitalWrite(5, LOW);
        }
                microspfm = micros();
        }
        interrupts();
}
```

## Schedule:


MIDI Guitar
Fall 2014
Start Date: September 22, 2014
Task
Start Date
End Date
Duration (days)
Percent Complete
Project Planning 2014-09-22 2014-10-27 36 100.00%
Initial Abstract 2014-09-22 2014-10-06 15 100.00%
Market Research 2014-09-22 2014-10-27 36 100.00%
Requirements and Specifications 2014-09-22 2014-10-05 14 100.00%
Block Diagram 2014-09-29 2014-10-12 14 100.00%
Component Selection 2014-09-29 2014-10-12 14 100.00%
Cost Estimation 2014-09-29 2014-10-12 14 100.00%
First Pass Report 2014-10-06 2014-10-27 22 100.00%
Analog Input Interfacing 2014-10-03 2014-10-18 16 100.00%
Location Sensor Interfacing 2014-10-03 2014-10-05 3 100.00%
Force Sensor Interfacing 2014-10-16 2014-10-18 3 100.00%
Analog Sensor Calibration 2014-10-11 2014-10-17 7 100.00%
Digital Input Interfacing 2014-10-24 2014-11-06 14 100.00%
Capicitive Keypad Interfacing 2014-10-24 2014-11-06 14 100.00%
Button and Switch Interfacing 2014-10-24 2014-10-30 7 100.00%
MIDI Output Interfacing 2014-10-03 2014-12-07 66 100.00%
Essintial MIDI Protocal 2014-10-03 2014-12-02 61 100.00%
MIDI over serial USB testing 2014-11-03 2014-12-02 30 100.00%
MIDI to MIDI USB hub testing 2014-11-24 2014-12-07 14 100.00%
Testing 2014-11-11 2015-01-06 57 100.00%
MIDI Protol debugging 2014-11-11 2014-12-25 45 100.00%
Durability Testing 2014-12-08 2015-01-06 30 100.00%
MIDI Guitar
Winter 2015
Start Date: January 1, 2015
Task
Start Date
End Date
Duration (days)
Percent Complete
Additional Feature Implementation 2015-01-01 2015-02-03 34 100.00%
User Feedback Review 2015-01-01 2015-01-15 15 100.00%
Additional Hardware Implementation 2015-01-05 2015-02-03 30 100.00%
Additional Software Implementation 2015-01-05 2015-02-03 30 100.00%
Final Hardware Selection 2015-01-05 2015-01-25 21 100.00%
Alternative Hardware Research 2015-01-05 2015-01-11 7 100.00%
Alternative Hardware Implementation 2015-01-19 2015-01-25 7 100.00%
Final Design Implementation 2015-01-19 2015-02-08 21 100.00%
Final Body Design 2015-01-19 2015-02-08 21 100.00%
Final Body Construction 2015-01-26 2015-02-08 14 100.00%
Final Design Testing 2015-02-02 2015-03-29 56 100.00%
Final Design Protocol Testing 2015-02-02 2015-03-01 28 100.00%
Final Design Latency Testing 2015-02-16 2015-02-20 5 100.00%
Final Design Durability Testing 2015-03-02 2015-03-29 28 100.00%
Working Prototype Due 2015-03-13 2015-03-19 7 100.00%
Marketing Materials 2015-03-09 2015-03-29 21 100.00%
Pictures of Final Design 2015-03-09 2015-03-22 14 100.00%
Videos of Final Design 2015-03-23 2015-03-29 7 100.00%
MIDI Guitar
Spring 2015
Start Date: March 30, 2015
Task
Start Date
End Date
Duration (days)
Percent Complete
Additional Feature Implementation 2015-03-30 2015-04-28 30 100.00%
User Feedback Review 2015-03-30 2015-04-28 30 100.00%
Additional Hardware Implementation 2015-03-30 2015-04-28 30 100.00%
Additional Software Implementation 2015-03-30 2015-04-28 30 100.00%
Senior Project Expo 2015-05-31 2015-05-31 1 100.00%
Project Expo Date 2015-05-31 2015-05-31 1 100.00%
Marketing Materials 2015-03-30 2015-05-28 60 100.00%
Pictures of Final Design 2015-03-30 2015-05-28 60 100.00%
Videos of Final Design 2015-03-30 2015-05-28 60 100.00%
O c t
J a n
A p r
F e b
M a y


Jun
Mar
Nov
Dec

**References:**

Market Research

[1] Sonuus, “G2M—Universal Guitar to MIDI Converter,” Sonuus - Music Products Designed in the UK, 2014. [Online]. Avilable: http://www.sonuus.com/products_g2m.html [Accessed: Oct. 1, 2014].

This sources was used for information about pricing and operation of the G2M. It was chosen because it is hosted on the manufactures site, and contains accurate information about the product.

[2] Sonuus, “B2M—Universal Bass to MIDI Converter,” Sonuus - Music Products Designed in the UK, 2014. [Online]. Avilable: http://www.sonuus.com/products_g2m.html [Accessed: Oct. 1, 2014].

This sources was used for information about pricing and operation of the B2M. It was chosen because it is hosted on the manufactures site, and contains accurate information about the product.

[3] Roland, “GK-2A Divided Pickup,” GK-2A :: Products :: Roland, 2014. [Online]. Avilable: http://www.roland.com/products/en/GK-2A/ [Accessed: Oct. 3, 2014].

This source was used to obtain information and pricing about the GK-2A, and add-on pickup manufactured by Roland.

[4] You Rock Guitar, “YRG Gen2,” You Rock Guitar YRG Gen2 - You Rock Guitar, 2014. [Online]. Avilable: http://yourockguitar.com/yrg-gen2/ [Accessed: Oct. 5, 2014].

This source was used to obtain information about the YRG including technical specifications, and pricing.

[5] Yamaha, “EZ-AG Guitar,” EZ-AG - EG Series, 2014. [Online]. Avilable: http://usa.yamaha.com/products/musical-instruments/entertainment/lighted_key_fret_instruments/ez_series/ez-ag/ [Accessed: Oct. 8, 2014].

This source was used to determine pricing and availability of Yamaha’s EZ-AG Guitar. This source was chosen because it is hosted on the manufactures site, and contains accurate information about the product.

[6] Yamaha, “EZ-EG Guitar,” EZ-EG - EG Series, 2014. [Online]. Avilable: http://usa.yamaha.com/products/musical-instruments/entertainment/lighted_key_fret_instruments/ez_series/ez-eg/ [Accessed: Oct. 8, 2014].

This source was used to determine pricing and availability of Yamaha’s EZ-EG Guitar. This source was chosen because it is hosted on the manufactures site, and contains accurate information about the product.

[7] Misa, “kitata,” misa kitara - Misa Digital, 2014. [Online]. Avilable: https://misa-digital.myshopify.com/products/kitara [Accessed: Oct. 7, 2014].

This source was chosen to obtain information about the experimental misa kitara. The kitara has been discontinued, and the tri bass has taken its place.

[8] Misa, “tri-bass,” misa tri-bass - Misa Digital, 2014. [Online]. Avilable: https://misa-digital.myshopify.com/products/tri-bass [Accessed: Oct. 7, 2014].

This source was used to find information about the mis tri-bass including, its functionality and cost.

[9] D. Lockwood, “Starr Labs Ztar Z7s,” Sound On Sound, 2011. [Online]. Avilable: http://www.soundonsound.com/sos/feb11/articles/starr-systems-ztar-z7s.htm [Accessed: Oct. 10, 2014].

This source was used because it contains quite a bit of information about the Starr Labs Ztar Z7s, one of the top end MIDI guitars from Starr Labs. This review of the instrument explains how several settings on the controller operate.

[10] Starr Labs, “Price List,” Starr Labs, 2011. [Online]. Avilable: http://starrlabs.com/menu/price_list.php [Accessed: Oct. 10, 2014].

This source was used to determine pricing on Starr Labs’ Ztar’s. Their prices are not directly listed on their site, likely because they are quite high.

Interfacing Guides

[11] GweepNET, “The MIDI Specification,” MIDI Specification [Online]. Avilable: http://www.gweep.net/~prefect/eng/reference/protocol/midispec.html [Accessed: Oct. 15, 2014].

This source was used to gain understanding of the MIDI protocol. While there are other more ‘official’ pages, this single page contains all of the information I need to successfully implement the MIDI protocol. If problems arise, more sources will be used.

[12] Wavosaur, “HEX to MIDI note chart,” Midi note to Hexadecimal chart, 2014. [Online]. Avilable: http://www.wavosaur.com/download/midi-note-hex.php [Accessed: Oct. 15, 2014].

This source was used as a quick reference for programing in note values with MIDI.

[13] R. Smith, “Interfacing a Softpot Membrane Potentiometer,” QQRS, 2013. [Online]. Avilable: https://qqrs.github.io/blog/2013/04/22/interfacing-a-softpot-sensor-to-an-adc/ [Accessed: Oct. 15, 2014].

This source was used to determine the range of the value of the pull-down resistors for the ThinPot potentiometer. Smith tested a near identical linear resistor from the same study and plotted the linearity results of various sizes of pull down resistors.

[14] Sparkfun, “MPR121 Hookup Guide,” MPR121 Hookup Guide - learn.sparkfun.com, [Online]. Avilable: https://learn.sparkfun.com/tutorials/mpr121-hookup-guide/communicating-with-the-keypad [Accessed: Oct. 22, 2014].

This Source was used as a general interfacing guide for the capacitive keypad. It has many hints for setting up the keypad to work smoothly with the arduino. It, however, does not explain how to use all of the functionality of the device, so the datasheet will still be needed.

Datasheets

[15] Interlink Electronics, “FSR 400 Series Square Force Sensing Resistor“ FRS 408 Datasheet, [Online]. Avilable: https://dlnmh9ip6v2uc.cloudfront.net/datasheets/Sensors/Pressure/FSR408-Layout2.pdf [Accessed: Oct. 1, 2014].

[16] Spectra Symbol, “ThinPot,” TSP-L-0500-203-3%-ST Datasheet, [Online]. Avilable: http://media.digikey.com/pdf/Data%20Sheets/Spectra%20Symbol/TSP%20Series%20ThinPot.pdf [Accessed: Oct. 1, 2014].

[17] Freescale Semiconductor, “Proximity Capacitive Touch Sensor Controller” MPR121 - Capacitive Touch Sensor Controller Datasheet, [Online] Available: https://dlnmh9ip6v2uc.cloudfront.net/datasheets/Sensors/Capacitive/MPR121.pdf [Accessed: Oct. 1, 2014].

**Analysis:**

Project Title: Analog MIDI Guitar

Student: Noah Baker

Advisor: Wayne Pilkington

I. Summary of Functional Requirements

A. Describe the capabilities of your project or design and what is does.

1. The Analog MIDI Guitar  is an Electronic MIDI synthesizer, capable of creating complex chords of 12 simultaneous voices. It features advanced control change capabilities, on the fly retuning, and user customizable presets.

II. Primary Constraints

A. Describe challenges and difficulties related to the design or implementation of your project. What factors influenced your approach?

1. So far, one of the biggest challenges has been acquiring components to test with. Several of the components I wish to use must be ordered directly from the manufacture, and are difficult to purchase in small quantities. These components are non-stock on most distributor websites, and in final implementation, special versions of the components will be ordered directly from the manufacture.

III. Economic Impacts

A. What are the resulting economic impacts of the project?

1. Human Capital: The construction, and design of this hand crafted instrument could create jobs for artisan guitar makers, machine shops, and manufactures of the electronic components present in the product.
2. Financial Capital: This new MIDI instrument could assist in the creation of music by artists, and indirectly create jobs through the artists success.
3. Natural Capital: The neck and body of the instrument will be made of wood sourced from sustainable providers. The electronic components of the device are

made from plastic and silicon and should be responsibly recycled after the end of the instruments life.

4. Costs: The cost of the initial prototype will be much higher than the production costs of the final design. The final design will not need much, if any firmware modifications, and will cost less to build in bulk. The prototype has an estimated cost of $340 in materials, and $2000 in coding of the microprocessor. The final design will cost around 300$ to manufacture, including labor, and retail for $750. Information related to costs can be seen in table 3.

IV. Commercial Manufacturing Considerations

A. Estimated number of devices sold yearly: Between 50, and 500 Instruments (dependent on marketing)
B. Estimated manufacturing costs: around $300 per device (see table 3)
C. Estimated MSRP: $750
D. Estimated yearly profit: Between $22,500 and $225,000
E. Estimated cost to operate device: (5W @$0.10/kWh) about $0.012/day (cost of electricity)

V. Environmental Impact

A. The materials used in the assembly of the Analog MIDI Guitar include wood, plastics, silicon and metal. The wood will be bought from sustainable sources, and is a renewable resource. The plastics, silicon, and metal, on the other hand must be properly recycled to avoid environmental damage. All of the components will have a carbon footprint due to extraction, refinement, and manufacturing. These could be offset by donating to environmental trusts if the company and production grow.

VI. Manufacturability

A. To streamline the manufacturing of the device, the body of the final version will be machined on a CNC. 3 separate parts will be made: the neck, the body, and the pickguard. Sanding, fitting, and assembly will be done by hand and each instrument will be tested individually before it is shipped. Off the shelf parts will be used unless production grows to a point where proprietary boards become more cost effective.

VII. Sustainability

A. What issues may arise in maintaining the device over its lifespan?

1. The components, such as the linear, and ForceSense resistor may wear out during the lifespan of the device. Our company will provide replacement, plug and play, necks, in addition to information about replacing individual components. We will stock and supply many of the smaller, hard to find parts, and sell them at a minimal profit.

B. What upgrades could be made to improve the device?

1. After launch hardware upgrades will have to be implemented on later models. However, firmware updates which add additional functionality to the device will be provided periodically. In addition to this, Users will have access to the code, and be able to make their own personal modifications to the operation of the microcontroller.

C. What challenges could come from upgrading the design

1. Some challenges in upgrading the design include being able to release one firmware update for all versions of the device, and detecting the proper hardware configuration to use. The microcontroller itself might need to be changed at some point to accommodate more analog inputs, and the firmware would have to be rewritten for this.

VIII. Ethical Impacts

A. What ethical implications does the use, design, manufacture of this device bare?

1. Because some of the components are being bought off the shelf, it can be difficult to determine the conditions and locations they were manufactured in. My company will attempt to primarily use components assembled in the United States, but several of the proprietary parts are manufactured in China. The quality of living of the workers manufacturing the semiconductors may be subpar, compared to the ethical standards of the United States. This may lead to us manufacturing our own components if production exceeds the predicted limits.

IX. Health and Safety

A. What Health concerns could arise from the manufacture or use of the device?

1. There are little to no safety concerns for this device. It is relatively low power, and made from components designed for human interaction. Shorts, or damages to the device could result in failure, but should be contained internally and not harm the user. Standard safety procedures should be used when manufacturing the device and operating machinery such as the sander or CNC. Assemblers will be required to go through tech training before working on assembly of the instrument.

X. Social and Political Impact

A. What social and political issues could arise from the use of the device?

1. This Device will have an impact on any artists who make music, but mostly electronic musicians. I don't foresee any negative impacts of this device to music, or society, as its goal is only to add flexibility to the artists personal playing style.

XI. Development

A. What new techniques or tools that were used in the creation of this device were gained independently?

1. During the development of this device I learned how the MIDI protocol worked, its limitations, and workarounds to these limitations. Individual notes can’t easily be pitch shifted, but using control signals and aftertouch, the virtual instrument can remedy this. In addition to this, I learned about guitar construction and assembly, after discovering that a standard guitar neck could not be used, as the vast majority of them taper at the top.