Squidbox (E22)

By: Sebastian Baldini, William Merry, Christopher Langevin

Table of Contents:

Introduction:

The aim of this project is to create a Bluetooth musical instrument digital interface controller that plays velocity-sensitive diatonic chords. The project draws inspiration from other projects, including the Chord Board and Bluetooth Potentiometer. The envisioned design contains 10 buttons: 8 buttons with velocity sensitivity for triggering diatonic chords within a selected scale, along with two other buttons used for transposing up and down by half-steps.

Project Overview:

The project contained multiple different parts, each needed to make the design work. These parts included a physical design, electrical design, and a software component.

What We Did:

The focus of the E23 Squidbox team was to create a velocity sensitive button and integrate it with existing code. To do this we took two buttons and put them into a small footprint and created a button cap with different pin sizes so they would strike the individual buttons at different times. We then took the difference in time to calculate the velocity. We made 8 sets of buttons and put them into a case with an LCD screen to make an ergonomic hand-held instrument.

How we did it:

Our project had two main components, one software competent and one CAD component. The software was done in C++ in Visual Studio Code, while the CAD was performed in Solidworks. The parts were printed on a Creality Ender 3 printer with PLA filament.

The appropriate resources for continuation of the project can be found in the Squidbox wiki. The information that is most vital to understanding what our group accomplished can be found in both the C++ file that contains the code written for the Squidbox, as well as the Solidworks file that contains the design for our 3D model.

```
void playNotes(int mode, int i) {
  // handles MIDI out based on buttons read by readButtons()
  // mode 0 -> USB, mode 1 -> BLE
    if(digitalRead(buttons[i])==LOW && (bitRead(previousButtons, i) == 0)){
      bitWrite(previousButtons, i, 1);
      if (mode == 0) {
        startTime = micros();
        noteOn(0, notePitches[i], 64);
        if(digitalRead(buttonsb[i])==LOW){
          endTime= micros();
          float elapsedTime = (float)(endTime - startTime); // Calculate time difference in seconds

          float velocity = 768 / elapsedTime; // Adjust this formula based on your requirements
          velocity = 128-velocity;
            Serial.print("The Velocity is: ");
            Serial.println(velocity);
            Serial.print("The MIDI Code for the root of the chord is: ");
            Serial.println(notePitches[i]);

          startTime = 0;

          }

```

Moving Forward:

Code Fixes:

The code currently prints the MIDI code of the root of the chord and the velocity of the button pressed. This velocity is between 0 and 128 as required for MIDI. The next iteration will have to take the existing code and implement the MIDI sounds for the device.

CAD and device fixes:

In the future the device could be CADed to have additional buttons for octave and half step changes. This would allow the board to be used in any key. The existing buttons could also have a better top sheet so they don’t fall out. Currently they are press fit, but if you add another layer, they will stay in.

Built in Speaker:

A potential addition to the Squidbox would be to include a small speaker on the device itself so you can hear the notes that are being played. There is a fairly simple way to do this with a small speaker and the tone() function (look at link below). This would be a good way to help with testing and a potential method to improve the final product.

https://www.arduino.cc/en/Tutorial/BuiltInExamples/toneMelody

Minimizing Buttons Pin Requirements:

Currently velocity is calculated by the Arduino receiving 2 digital inputs from buttons and calculating the difference in time between those two presses. With the Squidbox having 8 velocity sensitive buttons this method the use of 16 digital I/O pins which is fairly inefficient. A better way to do this would be with a XOR gate IC, such as the Texas Instruments SN74LVC1G86DCKT. This would allow for the pair of buttons that require a pair of digital I/O pins to only require one pin for the arduino to calculate velocity. Instead of measuring the time between two pins going from low to high, the velocity would instead be calculated by the time that the signal on a single pin is high. This would be much simpler and then the note would be stopped when that same pin had a high signal again. This would be much simpler and leave more pins open for more additions in the future.

A	B	Q
0	0	0
0	1	1
1	0	1
1	1	0

Connecting to DAW:

While we didn't have time to dig into how to connect this device to a DAW, we developed the MIDI code so this would be a clear next step.

Additional Control:

These are other ideas we have thought about as part of future progression for this project.

- Buttons to toggle up and down an octave
- A slider or knob to control volume on the device
- Buttons to toggle up and down a half step