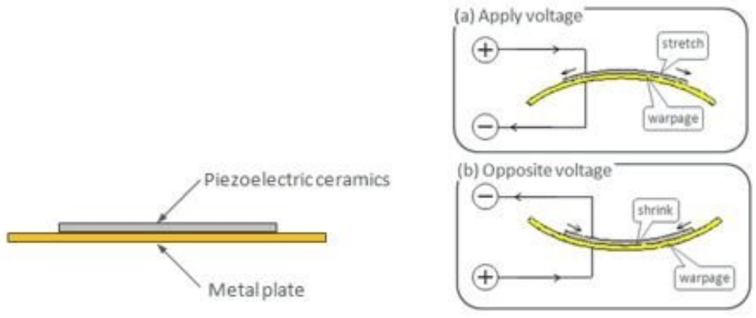
Kurt Dawiec
Fall 2019

Prof. Shattuck
PHYS 371

Final Report

Background:

The piezo buzzers included with the Arduino kit can be used to generate musical sounds. An increase in voltage causes a stretch of this plate and a shrinkage upon decreased voltage. When the ceramic plates oscillate at certain frequencies, they generate a distinct sound.



By passing a square wave through the buzzer and then turning the signal off, the time in between these events could be considered the frequency of oscillation for the plates. The first step was to determine which frequencies coincide with which notes. This is easily found on the internet.

Next, I needed a way to generate a sequence of notes to play a song without having to sit down and read sheet music and type it directly into the Arduino ide. This was surprisingly harder to find a solution to, but I decided to restrict myself to work with MIDI files. MIDI files are designed for electronic audio interfaces and are much easier to work with than raw MP3's or WAV files. I managed to find a python library that helped me accomplish what I needed.

Using the midi library, you can extract the musical notes of an audio track along with when they are played. After some work you get something like this:

```
notes = ['B3', 'C4', 'G2', 'G3', 'B3', 'D4', 'F4', 'G2', 'D4', 'F4']
ticks = [0, 1, 12, 12, 12, 15, 15, 22, 22,]
```

A tick is basically the time domain for midi. It represents when that note plays for the first time and the list `ticks` corresponds to `notes` (`notes[0]` plays at time `ticks[0]`). Here's where I ran into my first issue. I needed the data structure to support chords; multiple notes played at the same time.
e.g. the above track would be played like this;

- channel0 plays B3 at t=0
- channel0 plays C4 at t=1

- channels0,1 & 2 play G2,G3 & B3 at t=12
- channels0 & 1 play D4 & F4 at t=15
- channels0,1 & 2 play G2, D4 & F4 at t=22

Problem: Given `notes` and `ticks`, create instructions for channels 0-3 with the correct timing and notes. If a chord has more than 4 notes, I simply omit them because there's not much I can do with them. Basically, I needed the data structure to look like the one below:

```
channel0 = ['B3', 'C4', 'G2', 'D4']
channel1 = [  0,    0,  'G3', 'F4']
channel2 = [  0,    0,  'B3',   0 ]
channel3 = [  0,    0,    0,    0 ]
```

I first attempted to brute force it with a monstrosity that was a bunch of if statements, but ultimately came up with this.

Solution: First, I created a list of tuples to compare how many times each tick occured in the track, but that was pointless. I just made it an array.

```
res = []
for i in ticks:
    if i not in res:
        res.append(i)
counts = [(ticks.count(x)) for x in res]
```

Then, I made a list of lists for the channels. I ordered them backwards because it was easier to work the "matrix" bottom to top.

```
ch0, ch1, ch2, ch3 = ([] for i in range(4))
final = [ch3, ch2, ch1, ch0]
```

Finally, this shameful thing somehow worked.

```
    countsidx = -1
    while breakMe:
        countsidx += 1
        for finalidx, vchan in enumerate(final):
            if int(finalidx) >= counts[countsidx]:
                vchan.append(0)
            else:
                vchan.append(notes[notesidx])
                notesidx += 1
            if notesidx == len(notes):
                return final
                breakMe = False
    return final
```

Upon sending this to the Arduino, however, I quickly discovered the major flaw in my original project idea. The Arduino only has 1 internal 8 bit timer for me to use. Since this is necessary for me to generate any frequencies, I can only play 1 note at a time.

The only possible solution for this is to wire multiple Arduinos together. Or, better yet, learn to play an instrument myself.