CS121 – Computer Organization

Team Synth Final Report

Group Members:
Sam Wisotzki
Drew Parkinson

Instructor:
Jim Eddy

I was never worried that synthesizers would replace musicians. You have to be a musician first in order to make music with a synthesizer.
– Robert Moog, Inventor of Moog synthesizer

Contents

1 Definitions, Acronyms, and Abbreviations

Abbreviation/Phrase	Meaning
GPIO	General-purpose input/output
AUX	Auxiliary audio port
PWR	Power, typically refers to 5V source
GND	Ground
USB	Universal Serial Bus, common connector
Arpeggiation	sequential playing of multiple notes
BPM	beats per minute
ea	each

2 Introduction

Our synthesizer uses the Raspberry Pi as an audio interface. The user is be able to play a total of twelve notes (one octave), manipulate the tone of each note, and arpegiate through the octave. By using switches to control each key in our user interface, the player of our synth will be able to play each note continuously without needing a sustain pedal. Initially, our goal was to implement momentary and continuous switches for the notes, however our final product only has continuous switches.

3 Project Details

3.1 Software (Audio Interface)

The audio interface (what creates the sound) was created in Python. Using a plug-in call PyAudio, we can simulate sine waves by storing approximated float values in an array. This array can then be outputted as an audio stream. This allows us to create sine waves with different frequencies, and manipulate them with mathematical operators. We can even combine notes by adding their sine waves one value at a time.

In the PyAudio stream, there is a callback function that allows for continuous looping of a sine wave, which allows for the audio stream to simultaneously play notes while the main loop continues checking for user input. When everything is linked together, we are able to map each switch to a boolean value. From there, we can use those booleans to turn notes on and off, and effect their characteristics respectively. See section 9 for a listing of all source code.

3.2 Hardware

The electrical side consists of on/off switches connected to GPIO pins. We used a USB sound card and converted the output to 1/4 inch AUX. From there an amplifier can be connected and used for sound output. However, as this is intended to be a synthesizer, we will not mount a permanent speaker to the chassis. This gives the user more control over the sound.

Each of the 12 notes maps to a switch. From there, users can toggle each note, allowing for sustained and rapid play. There will be pulse-style buttons for octave, volume, and BPM up/down. Finally, there will be an arpeggiation, and tone toggle switches as well.

The chassis will be a reclaimed plastic box resembling a lunch box, approximately a foot in length by half that in width and depth. Below is our initially proposed design:

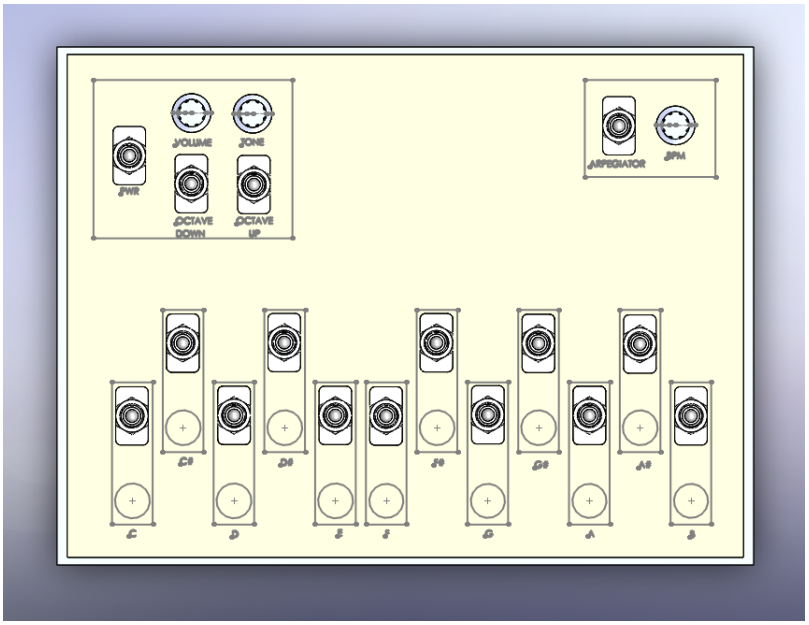

Figure 1: CAD model of initial proposed design.

After some difficulties in setting up analog input for the PI (and a change of heart in the desired design), we decided to scrap the usage pushbuttons for the note keys. See the picture below for our final design:

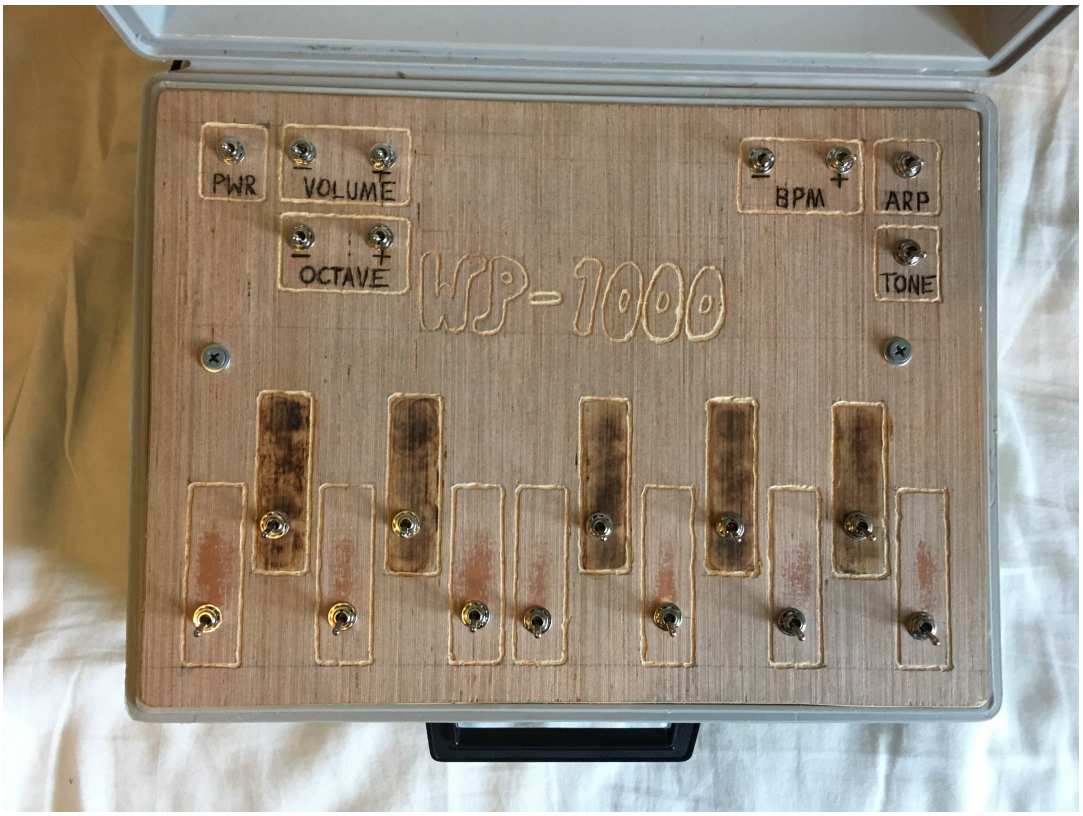

Figure 2: Final Mechanical Design.

The switches were attached with their included nuts and lock washers. All the 3.3V/GND sides were tied together and each of the the middle contacts is associated with a GPIO pin. To be on the safe side for the digital read (so that wire resistance wouldn't mess up the readings), we utilized both 3.3V source pins and half the switches go to one and the other half go to the other.

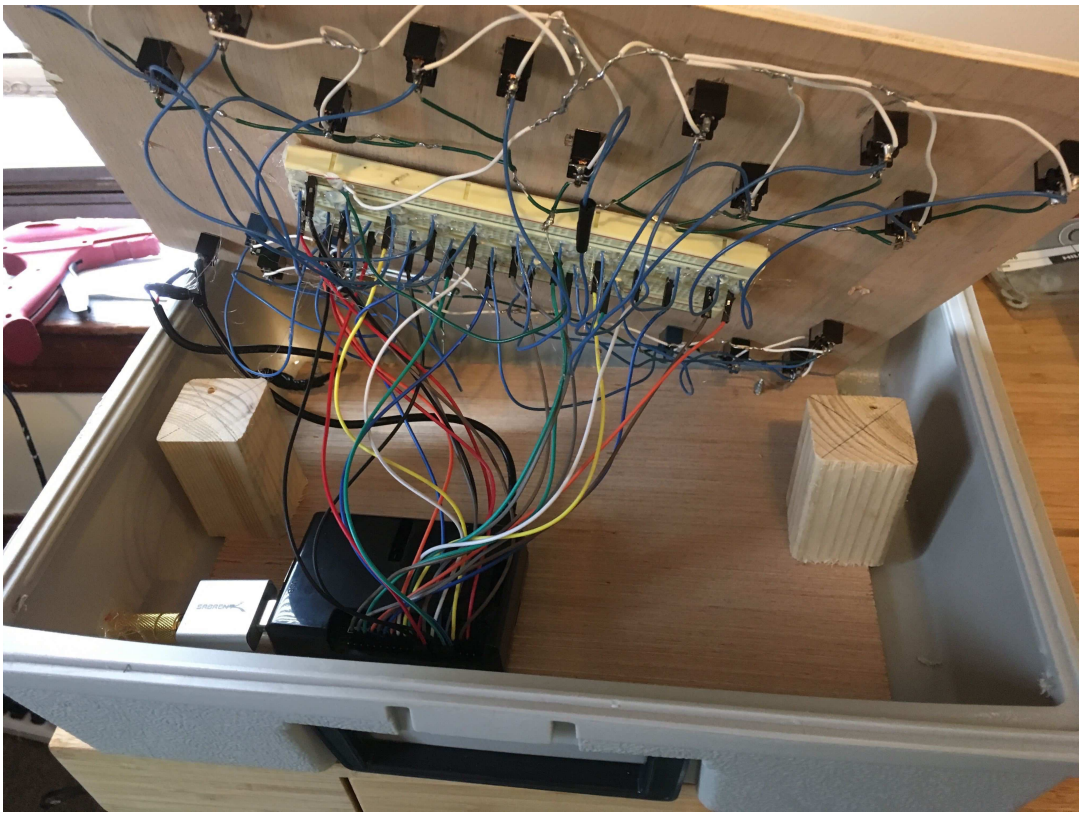

Figure 3: Final design, internals.

On the inside, our circuit hasn't changed from our initial proposal, aside from removing the secondary note buttons:

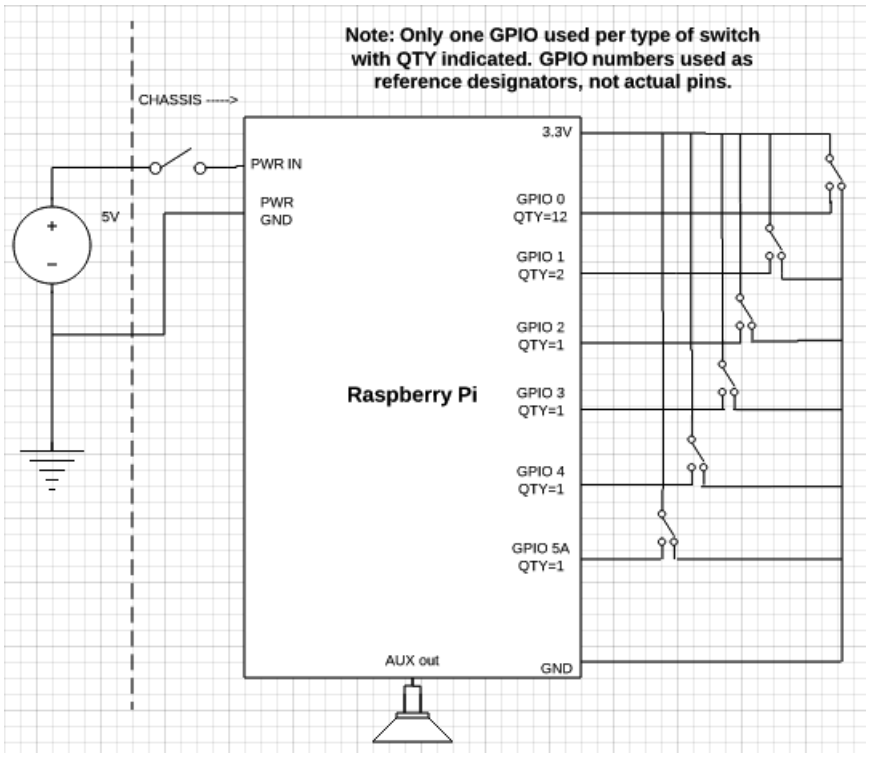

Figure 4: Final circuit used.

Below is a description of each switch listed in the circuit diagram:

Pin	Switch Style(s)	Role	Notes
GPIO 0	Always on	Note keys	
GPIO 1	Pulse-style momentary	Octave up/down	
GPIO 2	Always on	Arpeggiation	
GPIO 3	Pulse-style momentary	BPM control	
GPIO 4	Pulse-style momentary	Volume control	
GPIO 5A	Always on	'Tone' on/off	adds distortion
PWR IN	Built-in MicroUSB	power in	soldered to switch
PWR GND	Built-in MicroUSB	power ground	soldered to switch
AUX out	Built-in AUX	sound output	speaker used for visualization only

Table 1: Description of I/O pin usage.

4 Budget

Component	Quantity	Total Cost(USD)
Raspberry Pi	1 ea	$30.00
Casing/Paneling	1 ea	$0.00
Pushbuttons	20 ea	$10.00
Switches	25 ea	$0.00
Knobs	10 ea	$0.00
Misc. Wiring	approx. 10ft	$0.00
Sound Card	1	$0.00
1/4 inch Aux Converter	1	$5.00
Combined Labor (estimated)	25 hrs@15$/hr	$375
Combined Labor (actual)	34 hrs@15$/hr	$510
Total (actual+labor)		$545

Note: all zero cost components were found in electronic waste or possessed prior to project. See Figure 6 for a more detailed list of labor costs.

5 Project Management Details

Below are the team role assignments:

Name	Role
Drew	Project Manager/Software Engineer
Sam	Lead Developer/Hardware Engineer

Drew was in charge of the project and took lead on developing the software for the synthesizer. Because he also has access to the Williams wood shop, he also took lead on the woodworking necessary for the control panel. Sam took the lead in bringing the hardware together (mounting/soldering/connecting).

See the Gantt chart below for a detailed description of the project timeline:

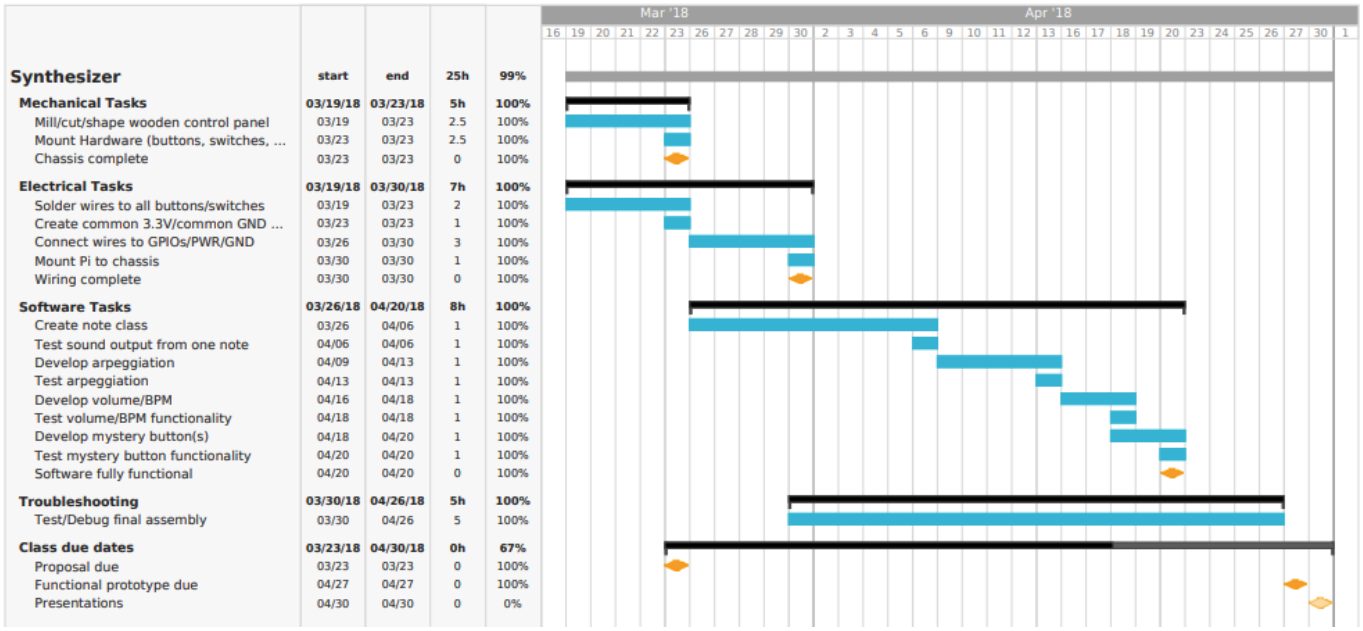

Figure 5: Gantt chart detailing project timeline and milestones.

Below is the same chart, but tabulated with total work hours on the project:

WBS #	Name / Title	Type	Start Date	End Date	Estimated Ho	Actual Hours	Percent Com	Resources	Predecessors	Note
1	Synthesizer	project	3/19/18	4/30/18	25	33.61	98.55			
1.1	Mechanical Tasks	group	3/19/18	3/23/18	5	4	100			
1.1.1	Mill/cut/shape wooden control panel	task	3/19/18	3/23/18	2.5	2	100	Drew Parkinson		
1.1.2	Mount Hardware (buttons, switches, knobs) to control panel	task	3/23/18	3/23/18	2.5	2	100	Drew Parkinson		
1.1.3	Chassis complete	milestone	3/23/18	3/23/18	0	0	100			
1.2	Electrical Tasks	group	3/19/18	3/30/18	7	7.11	100			
1.2.1	Solder wires to all buttons/switches	task	3/19/18	3/23/18	2	2.11	100	Sam Wisotzki		
1.2.2	Create common 3.3V/common GND to (via perfboard)	task	3/23/18	3/23/18	1	2	100	Sam Wisotzki		
1.2.3	Connect wires to GPIOs/PWR/GND	task	3/26/18	3/30/18	3	2	100	Sam Wisotzki		
1.2.4	Mount Pi to chassis	task	3/30/18	3/30/18	1	1	100	Sam Wisotzki		
1.2.5	Wiring complete	milestone	3/30/18	3/30/18	0	0	100			
1.3	Software Tasks	group	3/26/18	4/20/18	8	12	100			
1.3.1	Create note class	task	3/26/18	4/6/18	1	1	100	Drew Parkinson, Sam Wisotzki		
1.3.2	Test sound output from one note	task	4/6/18	4/6/18	1	3	100	Drew Parkinson		
1.3.3	Develop arpeggiation	task	4/9/18	4/13/18	1	1	100	Drew Parkinson, Sam Wisotzki		
1.3.4	Test arpeggiation	task	4/13/18	4/13/18	1	1	100	Drew Parkinson		
1.3.5	Develop volume/BPM	task	4/16/18	4/18/18	1	1	100	Drew Parkinson, Sam Wisotzki		
1.3.6	Test volume/BPM functionality	task	4/18/18	4/18/18	1	3	100	Drew Parkinson		
1.3.7	Develop mystery button(s)	task	4/18/18	4/20/18	1	1	100	Sam Wisotzki, Drew Parkinson		
1.3.8	Test mystery button functionality	task	4/20/18	4/20/18	1	1	100	Drew Parkinson		
1.3.9	Software fully functional	milestone	4/20/18	4/20/18	0	0	100			
1.4	Troubleshooting	group	3/30/18	4/26/18	5	10.5	100			
1.4.1	Test/Debug final assembly	task	3/30/18	4/26/18	5	10.5	100	Sam Wisotzki, Drew Parkinson		
1.5	Class due dates	group	3/23/18	4/30/18	0	0	66.67			
1.5.1	Proposal due	milestone	3/23/18	3/23/18	0	0	100			
1.5.2	Functional prototype due	milestone	4/27/18	4/27/18	0	0	100			
1.5.3	Presentations	milestone	4/30/18	4/30/18	0	0	0			

Figure 6: Resource hours used per task.

6 Target Market

The main goal of this project is to find a happy middle ground somewhere between something you would see in a production studio and something you'd see at a Toys R Us. The ideal business partner would be an Urban Outfitters or a mom and pop record store. The target audience is an 18-35 year old who doesn't have (or want to dish out) enough money to buy a professional synthesizer but still wants to be able to create music.

They want something portable, don't mind giving up the flexibility of a full keyboard, and love the 'briefcase' look.

7 Final Thoughts, Takeaways

As figure 6 indicates, we underestimated the time it took to test/troubleshoot the synthesizer. Half of that time was spent attempting to get the buttons working (and realizing that analog input would be infeasible), and the other half was spent debugging an array of software issues. From getting the output stream to not trip up over itself due to a not high enough sampling rate, to adjusting the sine waveforms, there was a lot of debugging involved.

Overall, this process was a fun one, and both of us really enjoyed tinkering with the pi.

8 Conclusion

In summary, we aimed to create a cost-effective, portable synthesizer that can offer the user a modest array of sound modulating capability but isn't perceived as a toy. By using switches we gained the playability of an analog instrument, with a more modern sound driven by the Raspberry Pi.

Demo: https://youtu.be/rFQ1jkSd0z4

9 Appendix

Below is all the source code used. Python language was used.

```
import pyaudio
import time
import math
from itertools import count
import numpy as np
import sys
import wave
import struct
import argparse
from itertools import *
import random
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)


on = True
C = 2
Cs = 3
D = 4
Ds = 17
E = 27
F = 22
Fs = 10
G = 9
Gs = 11
A = 0
As= 5
B = 6
```

```
octaveUp = 15
octaveDown = 24
arpeg = 13
bpmUp = 26
bpmDown = 14
volumeUp = 18
volumeDown = 23
volume = 0.5
tVolume = 0.5
bpm = 0.8
total = 0
past = 0
octaveTracker = 1
effect = 19
toneTracker = False

GPIO.setup(C, GPIO.IN)
GPIO.setup(Cs, GPIO.IN)
GPIO.setup(D, GPIO.IN)
GPIO.setup(Ds, GPIO.IN)
GPIO.setup(E, GPIO.IN)
GPIO.setup(F, GPIO.IN)
GPIO.setup(Fs, GPIO.IN)
GPIO.setup(G, GPIO.IN)
GPIO.setup(Gs, GPIO.IN)
GPIO.setup(A, GPIO.IN)
GPIO.setup(As, GPIO.IN)
GPIO.setup(B, GPIO.IN)
GPIO.setup(octaveUp, GPIO.IN)
GPIO.setup(octaveDown, GPIO.IN)
```

```
GPIO.setup(volumeUp, GPIO.IN)
GPIO.setup(volumeDown, GPIO.IN)
GPIO.setup(bpmUp, GPIO.IN)
GPIO.setup(bpmDown, GPIO.IN)
GPIO.setup(arpeg, GPIO.IN)
GPIO.setup(effect, GPIO.IN)

p = pyaudio.PyAudio()
#Creates a 32 float sine wave stored in an array
def sine_wave(frequency=440.0, framerate=48000, amplitude=0.5):
    period = int(framerate / frequency)
    if amplitude > 1.0: amplitude = 1.0
    if amplitude < 0.0: amplitude = 0.0
    lookup_table = [float(amplitude) *
                                    math.sin(2.0*math.pi*float(frequency)*
                    (float(i%period)/float(framerate)))
                    for i in xrange(period)]
    return (lookup_table[i%period] for i in count(0))

def square_func(t, amp):
    if(t < (amp / 2)):
        return 0
    if(t > (amp / 2)):
        return amp

def square_wave(frequency=440.0, framerate=48000, amplitude=0.5):
    period = int(framerate / frequency)
    if amplitude > 1.0: amplitude = 1.0
    if amplitude < 0.0: amplitude = 0.0
    lookup_table = [square_func(float(amplitude) *
```

```
                                    math.sin(2.0*math.pi*float(frequency)*
                        (float(i%period)/float(framerate))), amplitude)
                        for i in xrange(period)]
    return (lookup_table[i%period] for i in count(0))

def white_noise(amplitude=0.5):
    return (float(amplitude) * random.uniform(-1, 1) for _ in count(0))
#Creates a Note which stores a sine wave
#based off frequency parameter and an on/off state
class Note:
    def __init__(self, freq):
        self.frequency = freq
        self.on = False
        self.tone = False
        self.sine1 = sine_wave(self.frequency, 48000, volume)
    def turnOn(self):
        self.on = True
    def turnOff(self):
        self.on = False
    def octaveUp(self):
        self.frequency = 2 * self.frequency
        if(self.tone):
            self.sine1 = square_wave(self.frequency, 48000, volume)
        else:
            self.sine1 = sine_wave(self.frequency, 48000, volume)
    def octaveDown(self):
        self.frequency = self.frequency / 2
        if(self.tone):
            self.sine1 = square_wave(self.frequency, 48000, volume)
        else:
```

```
            self.sine1 = sine_wave(self.frequency, 48000, volume)
    def redeclare(self):
        if(self.tone):
            self.sine1 = square_wave(self.frequency, 48000, volume)
        else:
            self.sine1 = sine_wave(self.frequency, 48000, volume)
    def toneOn(self):
        self.tone = True
        self.sine1 = square_wave(self.frequency, 48000, volume)
    def toneOff(self):
        self.tone = False
        self.sine1 = sine_wave(self.frequency, 48000, volume)
#Creates a sine wave to be outputted
#by adding together the sine waves of turned on Notes
class Output:
    def __init__(self):
        self.outputs = [white_noise(0)]
        self.toBeAdded1 = []

    def callback(self, in_data, frame_count, time_info, status):
        wave = self.outputs[0]
        data = [wave.next()]
        for i in range(frame_count - 1):
            data.append(wave.next())
        ret_array =np.array(data).astype(np.float32).tostring()
        return (ret_array, pyaudio.paContinue)

    def computeOutput(self, scale):
        total = 0
        self.toBeAdded1 = []
```

```
        for x in range(len(scale)):
            if(scale[x].on):
                total = total + 1
                self.toBeAdded1.append(scale[x].sine1)
        if(total == 0):
            self.outputs = []
            self.outputs.append(white_noise(0))
        else:
            out1 = islice(izip((imap(sum, izip(*self.toBeAdded1)))), None)
            self.outputs = []
            self.outputs.append(out1)

    def computeArpeg(self, scale, index):
        self.outputs = []
        if(index > -1 and index < 12):
            self.outputs.append(scale[x].sine1)
        else:
            self.outputs.append(white_noise(0))

#scale is an array of all twelve notes in synth scale
scale = [Note(130.81), Note(138.59), Note(146.83),
                Note(155.56), Note(164.81),
         Note(174.61), Note(185.00), Note(196.00),
         Note(207.65), Note(220.00),
         Note(233.08), Note(246.94)]

sound = Output()
stream = p.open(format=pyaudio.paFloat32,
                channels=1,
                rate=48000,
```

```
                frames_per_buffer=4096,
                output=True,
                stream_callback=sound.callback)

#infinte loop where all user operations are performed
stream.start_stream()
while(on):
    #change becomes true if any GPIO state changes during current cycle.
    #Resets at start of each loop
    #refreshs state of each note
    #C
    if GPIO.input(C) == 1 and total < 8:
        scale[0].turnOn()
    if GPIO.input(C) == 0:
        scale[0].turnOff()
    #Csharp
    if GPIO.input(Cs) == 1 and total < 8:
        scale[1].turnOn()
    if GPIO.input(Cs) == 0:
        scale[1].turnOff()
    #D
    if GPIO.input(D) == 1 and total < 8:
        scale[2].turnOn()
    if GPIO.input(D) == 0:
        scale[2].turnOff()
    #Dsharp
    if GPIO.input(Ds) == 1 and total < 8:
        scale[3].turnOn()
    if GPIO.input(Ds) == 0:
        scale[3].turnOff()
```

```
#E
if GPIO.input(E) == 1 and total < 8:
    scale[4].turnOn()
if GPIO.input(E) == 0:
    scale[4].turnOff()
#F
if GPIO.input(F) == 1 and total < 8:
    scale[5].turnOn()
if GPIO.input(F) == 0:
    scale[5].turnOff()
#Fs
if GPIO.input(Fs) == 1 and total < 8:
    scale[6].turnOn()
if GPIO.input(Fs) == 0:
    scale[6].turnOff()
#G
if GPIO.input(G) == 1 and total < 8:
    scale[7].turnOn()
if GPIO.input(G) == 0:
    scale[7].turnOff()
#Gs
if GPIO.input(Gs) == 1 and total < 8:
    scale[8].turnOn()
if GPIO.input(Gs) == 0:
    scale[8].turnOff()
#A
if GPIO.input(A) == 1 and total < 8:
    scale[9].turnOn()
if GPIO.input(A) == 0:
    scale[9].turnOff()
```

```
#As
if GPIO.input(As) == 1 and total < 8:
    scale[10].turnOn()
if GPIO.input(As) == 0:
    scale[10].turnOff()
#B
if GPIO.input(B) == 1 and total < 8:
    scale[11].turnOn()
if GPIO.input(B) == 0:
    scale[11].turnOff()
#Octave Shifts
if GPIO.input(octaveUp) == 1:
    if(octaveTracker < 7):
        octaveTracker = octaveTracker + 1
        for x in range(len(scale)):
            scale[x].octaveUp()
if GPIO.input(octaveDown) == 1:
    if(octaveTracker > 0):
        octaveTracker = octaveTracker - 1
        for x in range(len(scale)):
            scale[x].octaveDown()
#Bpm increase/decrease
if GPIO.input(bpmDown) == 1:
    if bpm < 2:
        bpm = bpm + .1
if GPIO.input(bpmUp) == 1:
    if bpm > .05:
        bpm = bpm - .1
if GPIO.input(effect) == 1 and not(toneTracker):
    toneTracker = True
```

```
        for x in range(len(scale)):
            scale[x].toneOn()
    if GPIO.input(effect) == 0 and toneTracker:
        toneTracker = False
        for x in range(len(scale)):
            scale[x].toneOff()
    #Continuous mode
    if(GPIO.input(arpeg) == 0):
        #Volume increase/decrease(calculates for chorus playing)
        if GPIO.input(volumeUp) == 1:
            if tVolume < 0.9:
                tVolume = tVolume + 0.1
                total = 0
                for x in range(len(scale)):
                    if(scale[x].on):
                        total = total + 1
                if(total > 0):
                    volume = float(tVolume / total)
                    for x in range(len(scale)):
                        scale[x].redeclare()
        if GPIO.input(volumeDown) == 1:
            if tVolume > 0:
                tVolume = tVolume - 0.1
                total = 0
                for x in range(len(scale)):
                    if(scale[x].on):
                        total = total + 1
                if(total > 0):
                    volume = float(tVolume / total)
                    for x in range(len(scale)):
```

```
                    scale[x].redeclare()
        total = 0
        for x in range(len(scale)):
            if(scale[x].on):
                total = total + 1
        if(total > 0 and past != total):
            volume = float(tVolume / total)
            for x in range(len(scale)):
                scale[x].redeclare()
        past = total
        #Computes output sine wave if any thing has changed
        sound.computeOutput(scale)
        #pause
        time.sleep(.2)
    #arpegiator mode
    if GPIO.input(arpeg) == 1:
        total = 0
        past = 0
        for x in range(len(scale)):
            #Volume increase/decrease(different for arpeg)
            if GPIO.input(volumeUp) == 1:
                if tVolume < 1:
                    tVolume = tVolume + 0.05
                    volume = tVolume
                    for y in range(len(scale)):
                        scale[y].redeclare()
            if GPIO.input(volumeDown) == 1:
                if tVolume > 0:
                    tVolume = tVolume - 0.05
                    volume = tVolume
```

```
                    for y in range(len(scale)):
                        scale[y].redeclare()
            if GPIO.input(bpmDown) == 1:
                if bpm < 2:
                    bpm = bpm + .1
            if GPIO.input(bpmUp) == 1:
                if bpm > .05:
                    bpm = bpm - .1
            if(scale[x].on):
                total = total + 1
                sound.computeArpeg(scale, x)
                time.sleep(bpm)
        if(total == 0):
            sound.computeArpeg(scale, 100)
            time.sleep(.2)
        if GPIO.input(volumeUp) == 1:
            if tVolume < 1:
                tVolume = tVolume + 0.1
                total = 0
                for x in range(len(scale)):
                    if(scale[x].on):
                        total = total + 1
                if(total > 0):
                    volume = float(tVolume / total)
                    for x in range(len(scale)):
                        scale[x].redeclare()

stream.close()

p.terminate()
```