Modern Day Jukebox

Neha Tarakad
ASR D Block
Dr. Dann
16 May 2019

Abstract:

A Modern Jukebox was built to fit into the technological era of today. The digitized version maintains the encasing and vintage feel of scouring through a song selection and playing a song. It replaces the use of records with the ability to access and play music files stored in a Raspberry Pi. Furthermore, it displays the song selection using a 1024 by 600 pixel sized monitor using images created and resized which are stored in the RPi. The jukebox displays the song selection as well as the current playing song. In order to select the music, the jukebox has push buttons that are associated with numbers 0 through 9, correspond to a certain song in a particular album. When the user is ready to hear the song, they can hit the done button for their chosen song to play. If the user messes up the song code number, they have the option of clearing the selection.

Introduction:

The purpose of the jukebox was to create way for people to enjoy their favorite music when change was inserted into the machine, especially back in the 50s before iPods or any form of digitally playing music was created. The jukebox would drop records in an order based on when a song was paid for and selected. Today, while there hasn't been a practical use for a jukebox due to digital music technology, there are people who are still intrigued by its retro value and make their own DIY versions. In reading their articles, I've picked up that most seem to use a Raspberry Pi microcontroller to access the music which might come in handy when I think about making that decision for my version. There are versions that also involve taking apart old radios and laptops for the case and display. It could be valuable to extract a monitor from a computer that might allow me to display album covers and song choices, but as for the case, I think I will just be creating a wooden frame from scratch with a shelf or a wooden backing to hold the circuit inside as well as to keep the monitor facing flat for users to easily read the song selection.

In my project, I am essentially maintaining the vintage encasing but replacing the use of records with accessing music files online. I am embracing the uniqueness and old-fashioned concept of a jukebox and recreating it such that it fits in our modern society. Its purpose is to bring back the decorative components of a traditional jukebox, with a modern twist. Internally, I use a Raspberry Pi (RPi) that can access and store music files from different albums and play the selected song. The frame is made of wood that will mount the external speaker and monitor for the RPi as well as the circuit using its pins. As for the display, there is a song selection that is displayed on the monitor of the 55 songs that are stored in the RPi. The monitor constantly displays the song selection along with what song is being currently played. The push buttons labeled 0-9 are used to select a song based on a corresponding two digit code. After a user chooses a song, they press the done button for the correct song to play and the image to display. If the user messes up the two digit code, they can press the clear button and try again.

For my project, I knew I wanted to do something with music, or any type of audio. I got the idea when scrolling through a blog called *Hackaday* which publishes different hardware and software projects for convenience or creative reasons. I came across an article where someone created a DIY jukebox where they had several push buttons controlled by a Raspberry Pi to play different songs.[1] I thought I could add to this project by adding the monitor to display the song

[1] https://hackaday.io/project/25130

choices and current song playing. The original project had an attached Phillips hue for a more decorative LED component that would go along with the music playing.

As music players have evolved, our world has moved away from physical machines, like tapes and CDs, and progressed towards incorporating software in order to digitize music, through audio files and applications like Pandora and Spotify. Jukeboxes started to grow popular in the 1930s after the idea of an electric amplifier came into existence, allowing more people to hear the music being played. But it was actually in the early 1900s when Thomas Edison invented the phonograph which played music from a wax cylinder when a coin was inserted. Later, John Gabel created a music machine that had a selection of disc recordings instead of the wax cylinder which became the popular choice for a few decades later. The jukebox was then adapted into many different versions, but the one that became the most popular was the "Wurlitzer 1015" that made a public appearance in about 1950. It was a machine that finally allowed popular music of every variety to be played in any venue.[2] The records used in these jukeboxes actually have grooves in them when looked at closely. When the record is placed on a cylinder, a needle, usually made of a material like diamond due to its hardness, essentially "reads" the grooves as the record turns. The needle picks up vibrations as it moves along the grooved pattern on the record. These vibrations then vibrate a magnet in and out of a coil creating an electrical signal due to at the end of the arm of the needle. The electric signals are transferred to the amplifier to be turned into sound for all to hear.[3]

After records and jukeboxes, came tapes such as 8-track tapes and magnetic cassette tapes. The 8-track tapes has its reel wrapped around a hub as shown in Figure 1 and they are are divided into 8 channels and when the tape reel reaches the end of the program, a metal sensing strip attaches to a solenoid in the tape as shown in the same figure. The coil causes the playback head to move into a new position to play the next program in the sequence.[4]

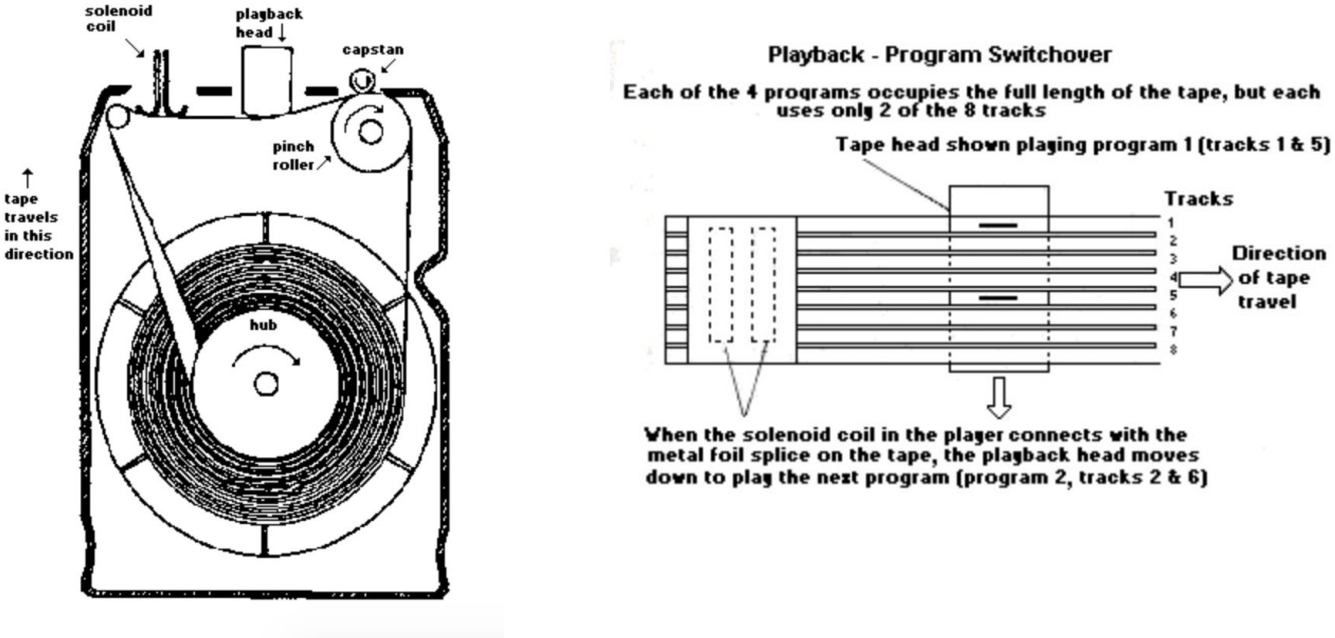
---

[2] https://www.encyclopedia.com/science-and-technology/computers-and-electrical-engineering/computers-and-computing/jukebox

[3] https://www.livescience.com/33793-record-players-work.html

[4] http://www.8trackheaven.com/archive/work.html

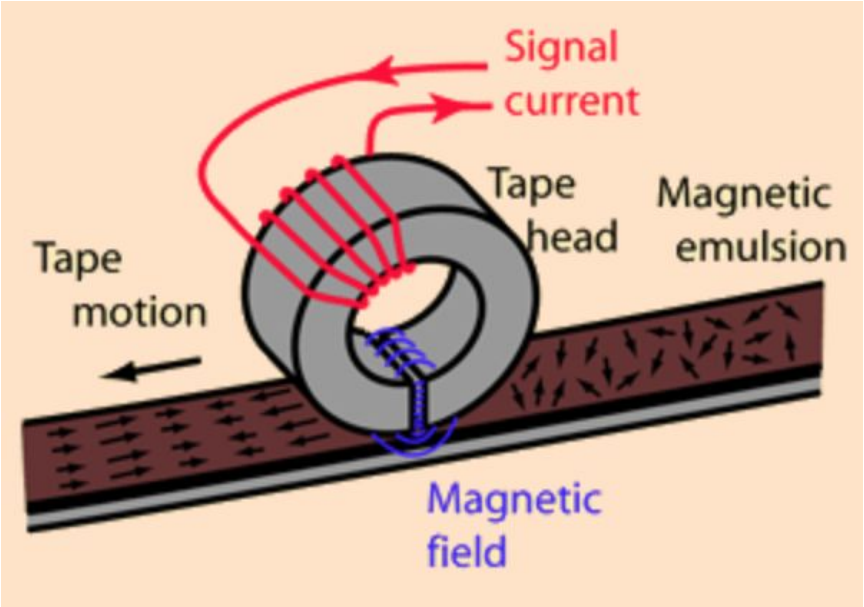
Figure 1: Diagram and Functionality of 8-Track Tape[5]



Magnetic tapes have their film coated with tiny magnetic particles and work when there is an oscillating current in a coil which produces a magnetic field as seen in Figure 2. The tape passes over a magnetic head that convert the encoded magnetic domains into electrical signals that go to any speakers. The magnetic energy is then "read" by a playback head that converts it into electrical signals to be amplified.[6]

[5] http://www.8trackheaven.com/archive/work.html
[6] http://hyperphysics.phy-astr.gsu.edu/hbase/Audio/tape2.html

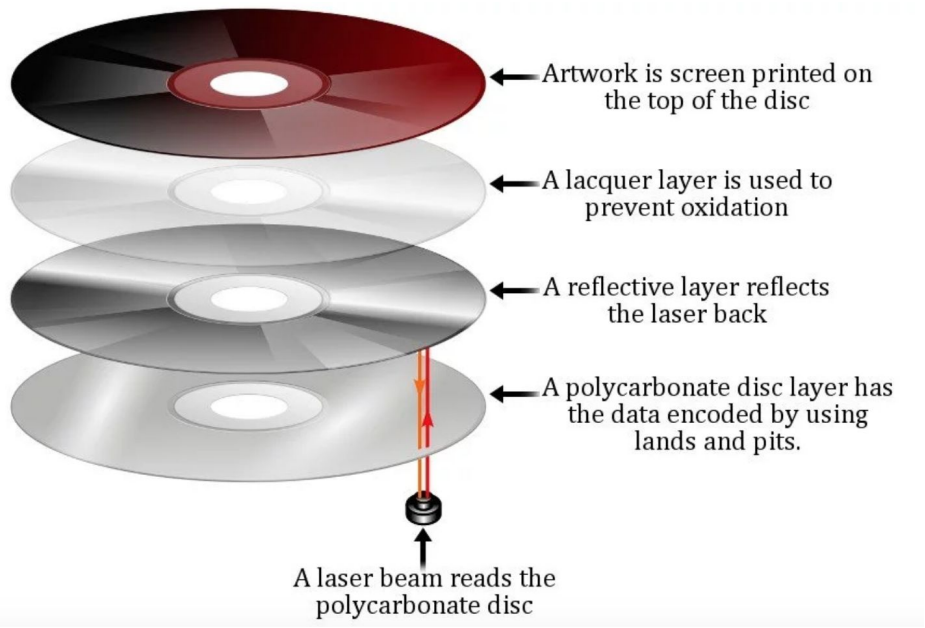
Figure 2: Magnetic Tape Diagram[7]



The evolution of music players continue to CDs that are composed of different layers as seen in Figure 3 that allows its data to be encoded by a laser beam that creates tiny indentations (bumps) on its surface. The data is represented in 0's and 1's, where a bump represents a 0 and the absence of a bump represents a 1. A CD player is able to read these discs through the use of a laser beam that scans for the indentations, reflecting straight back when it finds a 1 and scatters light when it finds a 0. Depending on if the beam finds an indentation or not, the light detector in the CD player sends an electric signal back to be amplified.[8]

[7] http://hyperphysics.phy-astr.gsu.edu/hbase/Audio/tape2.html
[8] https://www.scienceabc.com/innovation/how-does-a-compact-disk-cd-work.html

Figure 3: Layers of a CD[9]



Since CDs, music has started to become digitized, getting rid of the middle man so to speak. Mp3 files no longer have to recreate parts of the audio signal that humans can typically hear, which is why music enthusiasts claim that the modern way of playing music and storing them in such audio files has a different quality of sound. This is because nowadays, the music files don't have as many bits otherwise the files would be very large if they contained every single audio signal obtained. Music players have evolved immensely in the last century, moving from completely physical machines to completely virtual files. In the future, there probably wouldn't be as drastic as a change in the music world considering we are already in the technological era. As teenagers today, we missed out on a huge musical revolution.

Design

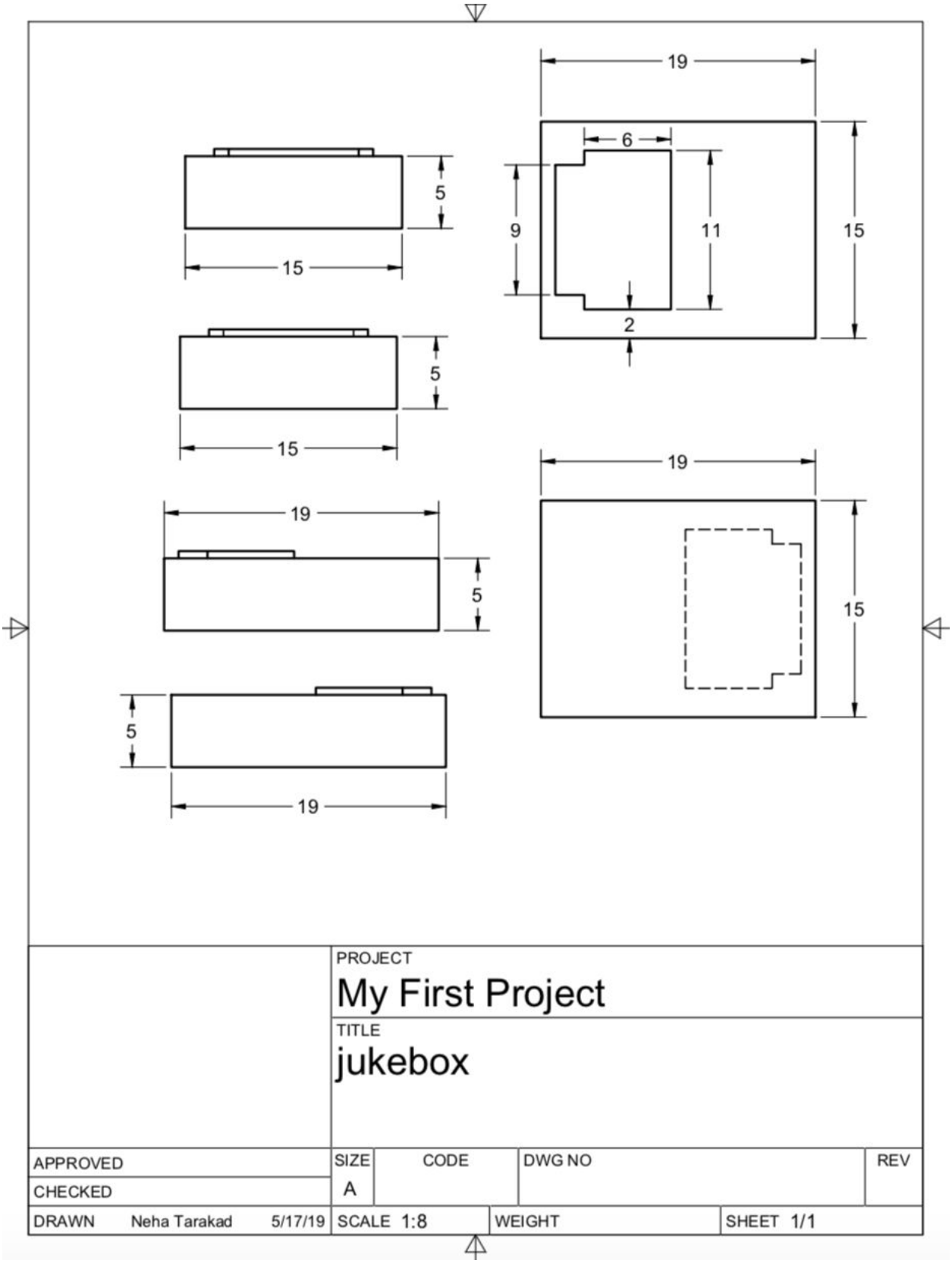
While the focus has been on the software as coding is a primary part of the project, the design of the jukebox case is pretty simplistic. I have decided on a wired speaker to be implemented with the Raspberry Pi, one that attaches to the top of the monitor and fits into the headphone jack of the Pi. It will be positioned it such that the monitor/speaker combination fit and been seen in case of the jukebox. While the speaker sits on the top of the monitor, the monitor itself is laid flat on the opening of the case to be seen, using pieces of wood screwed against its backing to secure it in place. The wire of the speaker would probably be hidden in toward the back of the encasing such that it is able to reach the Raspberry Pi audio output.

The monitor is visible through a rectangular opening that matches its size, with extra space above the top that fits the size of the speaker such that it can be seen as well. The case is essentially a box created from a layout from MakerCase, with the rectangular and circular cutouts made from my own design based on measurements of the monitor, speaker, and pushbuttons. The entire case was created using the laser cutter with .25 inch wood. I made sure to get certified and trained on the bandsaw in case I wanted a more sturdy frame, but the .25 inch

[9] https://www.scienceabc.com/innovation/how-does-a-compact-disk-cd-work.html

seemed to work really well. As for attaching the pushbuttons, the design calls for laser cutting 12 holes in a piece of wood so the buttons barely fit through, still allowing full pressing depth, but attempting to hide the soldering behind the protoboard. In addition, the leads of the pushbuttons will be extended by tinning the button pins and attaching longer wires in a hook-like manner such that only the buttons can be seen through the wood and the wires can be hidden inside the case. Finally, the back of the case is designed to have an opening such that there is room for a plug to connect to power supply to provide power to the Raspberry Pi and an outlet to provide power for the monitor.

Figure 4: Sketch of Design



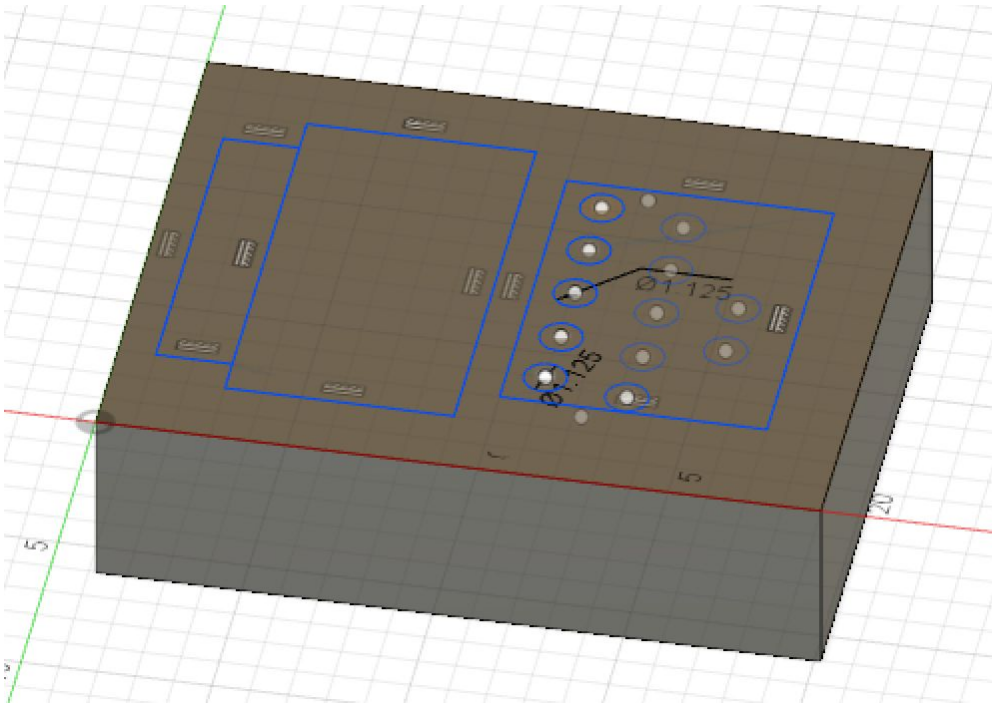
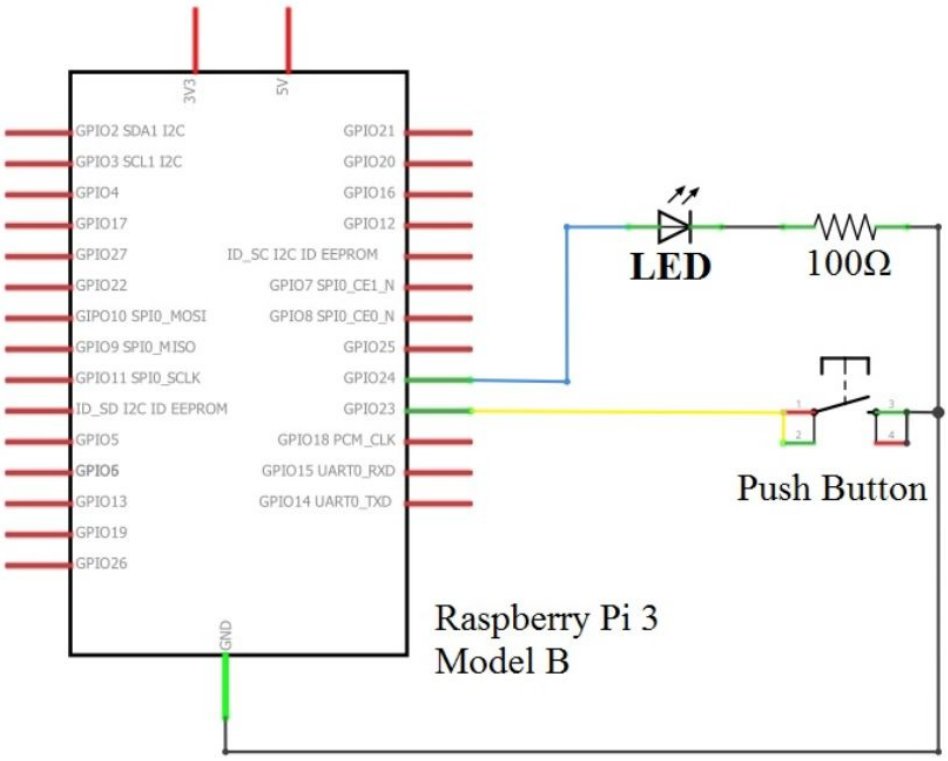


Figure 5: Pushbutton/LED Circuit Schematic



Above was the schematic of the circuit internal to GPIO that I used when learning to control a pushbutton with a Raspberry Pi. In this circuit, when the push button was pressed, the LED would turn on. In my project, I will essentially have 12 of these buttons with each pin connected to a different GPIO pin on the Raspberry Pi and another to GND. The code above would be able to read inputs from the buttons and trigger defined actions as a result.

Figure 6: Initialization for Function

```
import RPi.GPIO as GPIO
import time
import pygame

GPIO.setmode(GPIO.BCM)

#Initializing buttons 0-9
GPIO.setup(17, GPIO.IN, pull_up_down = GPIO.PUD_UP)
GPIO.setup(18, GPIO.IN, pull_up_down = GPIO.PUD_UP)
GPIO.setup(19, GPIO.IN, pull_up_down = GPIO.PUD_UP)
GPIO.setup(20, GPIO.IN, pull_up_down = GPIO.PUD_UP)
GPIO.setup(21, GPIO.IN, pull_up_down = GPIO.PUD_UP)
GPIO.setup(22, GPIO.IN, pull_up_down = GPIO.PUD_UP)
GPIO.setup(23, GPIO.IN, pull_up_down = GPIO.PUD_UP)
GPIO.setup(24, GPIO.IN, pull_up_down = GPIO.PUD_UP)
GPIO.setup(25, GPIO.IN, pull_up_down = GPIO.PUD_UP)
GPIO.setup(26, GPIO.IN, pull_up_down = GPIO.PUD_UP)
pygame.mixer.init()
```

This piece of code imports the GPIO module that will allow me to take advantage of the GPIO pins on the Raspberry Pi to connect the pushbuttons to and perform desired actions. It also imports a time module that will serve as a delay and the pygame module that will allow me to play audio files and display images. The *pygame.mixer.init()* initializes the pygame mixer component that will essentially get ready to play audio when a file is loaded.

Figure 7: Reading Inputs From Pushbuttons

```
while True:
    stateZero = GPIO.input(17)
    stateOne = GPIO.input(18)
    stateTwo = GPIO.input(19)
    stateThree = GPIO.input(20)
    stateFour = GPIO.input(21)
    stateFive = GPIO.input(22)
    stateSix = GPIO.input(23)
    stateSeven = GPIO.input(24)
    stateEight = GPIO.input(25)
    stateNine = GPIO.input(26)
```

This piece of code is how I am going to keep track of my 12 push buttons and their states (pressed or not). Reading the input is how I am able to tell whether or not an action should be triggered. Being able to read the status of each button will also help me determine the order of which button was pressed. For example, I would be able to differentiate button 1 being pressed before button 2, triggering the song assigned to the number 12 to play.
Figure 8: 2-D Array of Music/Image Titles

```
songs = [["WannaBeStartinSomethin", "BabyBeMine", "TheGirlIsMine", "Thriller", "BeatIt", "BillieJean", "HumanNature", "PYT", "TheLadyInMyLife"],
                    ["PianoMan", "TheEntertainer", "NYStateOfMind", "ShesAlwaysAWoman", "RockAndRollToMe", "DidntStartFire", "RiverDreams", "Vienna", "LongestTime"],
                    ["UptownGirl", "BabyOneMoreTime","Crazy", "Overprotected", "Everytime", "Sometimes", "Lucky", "Slave4U", "Oops", "Toxic"],
                    ["Stronger", "S&M", "WhatsMyName", "Cheers", "Fading", "OnlyGirl", "CaliforniaKingBed", "ManDown", "RainingMen", "Complicated"],
                    ["Skin", "LoveWayLie", "InMyBlood", "Nervous", "LostInJapan", "WWYMorning", "LikeToBeYou", "FallinAllInYou", "ParticularTaste", "Why"],
                    [ "BcHadYou", "Queen", "Youth", "Mutual", "PerfectlyWrong", "WhenYoureReady", "Africa", "LetsGetItOn"]]
```

The above piece of code shows how I plan to store the music and images in my code for the jukebox. Each music and image file has the same name as the string in this array, except followed by ".mp3" for music files and ".png" for image files. Having one two dimensional array store the string for a title of a corresponding image and song allows for the concision of my code and the simplicity of accessing both from the same array. In this array, it is has 5 rows with 10 titles and a 6th row with 7 titles. The song selection displayed as 55 songs, the last two were added just for fun as requested by friends. For example, pressing buttons 0 and 1 correspond to the song and image in row 0 column 1, or "BabyBeMine" in this array.
Figure 9: Global Variables in Code

```
#Dealing with playing audio
pin_count = 12
buttons_pressed = deque()
current_states=[True]*pin_count

#Dealing with displaying images
display_width = 1024
display_height = 600
game_display = pygame.display.set_mode((display_width, display_height))
pygame.display.set_caption("Jukebox")
black = (0,0,0)
white = (255, 255, 255)
```

Here I am defining the pin count, which is the variable for how many pins are used in the code file. Because my final jukebox used 12 push buttons, the pin count was 12. The index of buttons pressed are stored in a queue which will allow me to keep track of which buttons are pressed, and popping their indices off the queue such that it correlates to a row and column in the two dimensional array of song/image titles. The current states array keeps track of which buttons are pressed with the boolean False for pressed and the boolean True for not pressed. The code block below is used for setting up the pygame module for displaying the image. The 1024 and 600 dimensions fit the size of my monitor. Together, this part of code displays an image that is opened in a new window using the pygame module. It initializes the module ready for an image to be load, allows the user to choose a dimension size, and will display the image.

Figure 10: Methods Implemented

```
def button_index_to_GPIO_pin(i):
    pins = [22, 21, 20, 19, 18, 4, 5, 6, 12, 13, 16, 17]
    #pins = [20, 21, 22, 23]
    return pins[i]

def play_song(str):
    pygame.mixer.music.load("/home/pi/Music/"+ str + ".mp3")
    pygame.mixer.music.play()
    time.sleep(.2)

def display_image(str, x, y):
    print("here")
    game_display.blit(pygame.image.load("/home/pi/Pictures/" + str + ".png"), (x, y))
```

These are the methods that I use in my program. The button_index_to_GPIO_pin method takes in an index as a parameter and based on the array of GPIO pins used, it will return the index of that pin. For example, if the button that corresponds to the number 2 is pressed, that button is connected to GPIO pin 20. This method will grab the 2nd index in the pins array, or GPIO pin 20. The play_song method loads and plays the audio file stored in the RPi based on a string that corresponds to a row and column in the two dimensional array of songs. The songs files are all stored in the Music folder of the RPi and in order for the pygame mixer to play the song, it grabs the string from the array and checks for it in the Music folder. The display_image method loads and displays the image file stored in the RPi based on a string that corresponds to a row and column in the two dimensional array of songs. The songs files are all stored in the Pictures folder of the RPi and in order for the pygame mixer to play the song, it grabs the string from the array and checks for it in the Pictures folder.

Figure 11: Main Loop

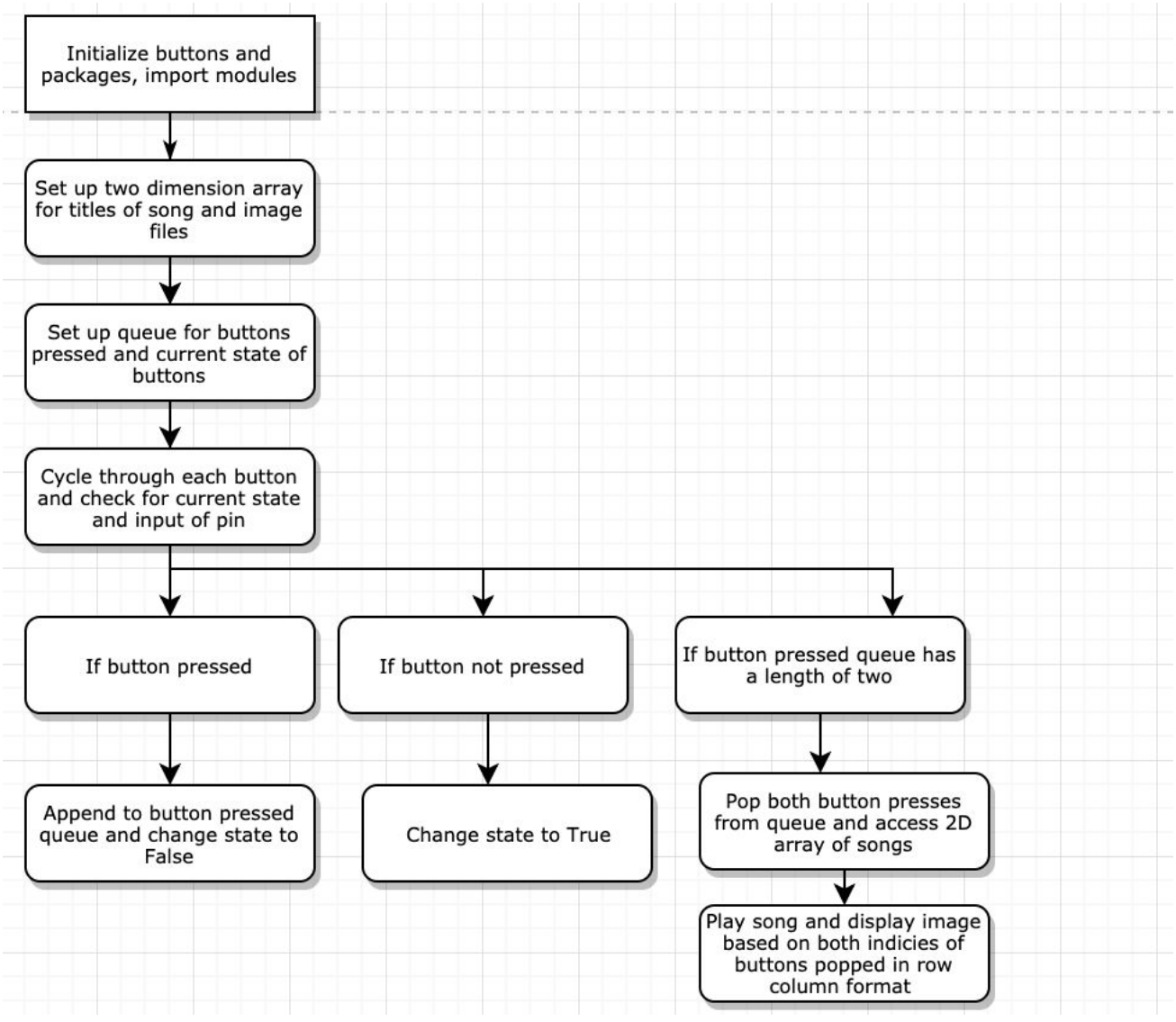
```
while True:
    for i in range(pin_count):
        time.sleep(.001)
        if (current_states[i] == True) and (GPIO.input(button_index_to_GPIO_pin(i))== False):
            print("button pressed")
            current_states[i] = False
            buttons_pressed.append(i)
            if i == 10:
                buttons_pressed.clear()
            #print(len(buttons_pressed))
        elif (current_states[i] == False) and (GPIO.input(button_index_to_GPIO_pin(i))==True):
            current_states[i] = True

   if len(buttons_pressed) == 3:
        if (buttons_pressed[0] == 0):
            buttons_pressed[1] = buttons_pressed[1] - 1
        try:
            if (buttons_pressed[2] == 11):
                buttons = (buttons_pressed.pop(), buttons_pressed.pop(), buttons_pr$
                if buttons[2] == 6 and buttons [1] == 9:
                    play_song(songs[5][7])
                    display_image(songs[5][7], x, y)
                else:
                    play_song(songs[buttons[2]][buttons[1]])
                    display_image(songs[buttons[2]][buttons[1]], x, y)
                pygame.display.update()
        except:
            buttons_pressed.clear()

while pygame.mixer.music.get_busy() == True:
        continue
```

The above code shows the main loop of my code, the part that runs constantly. Essentially, this loop cycles through each pin and first checks for the state that it just became, True for not pressed and False for pressed, as well as the input from the corresponding GPIO pin of the button. Then if the button is pressed down, it changes its current state to False and appends its pin index to the buttons_pressed queue. If the button was pressed previously but now its input reads that it isn't pressed (the button was released from press) its current state becomes True. Only when the length of the buttons_pressed queue is 3 and the third button pressed is the done button, will the buttons that correspond to a song and image be popped from the queue. For example, if button 3 was first pressed, then button 4, the queue stores the index 4 then the index 3. If the done button is pressed after the 3 and the 4, the queue will store the index of the done button pin, which happens to be 10. So the program will check if the index of the third button press is 10, and if so, will move on to selecting the corresponding song. When the queue pops the indices of the buttons pressed, they are stored in a tuple where the play_song and display_image method can easily grab the string 2nd index, button 3, and the 3rd index, the 4, and play/display the song in the third row and fourth column of the 2D songs array. When the clear button is pressed before the buttons_pressed array has a length of 3, it will simply clear the queue of pressed buttons. Furthermore, if the user presses a two digit combination that is not in the bounds of the song selection of 55 songs, the array of pressed buttons will be cleared.

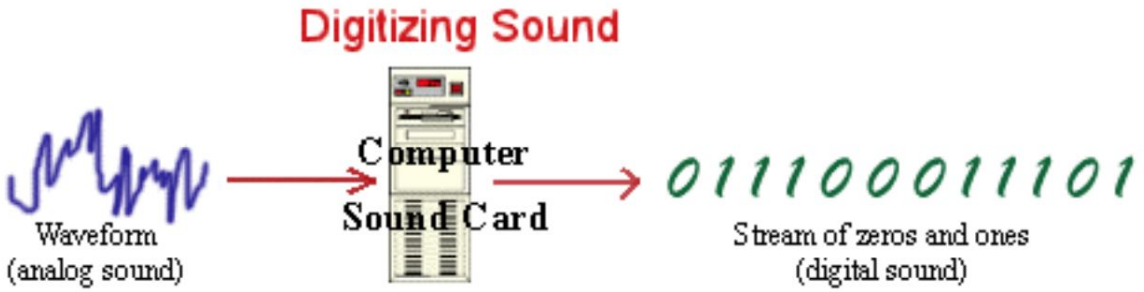
Figure 12: Flowchart of Code



Theory

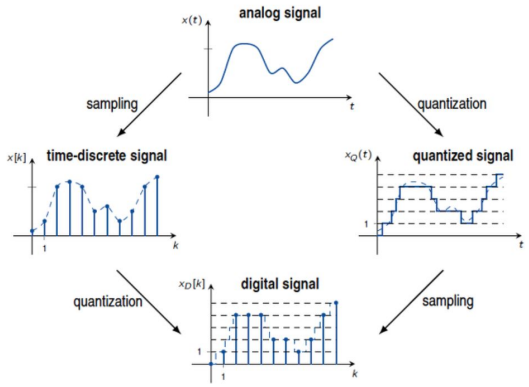
This project relies heavily on three main components: the digitization of music and the Raspberry Pi.

Figure 13: Basic Process of Digitizing Music[10]



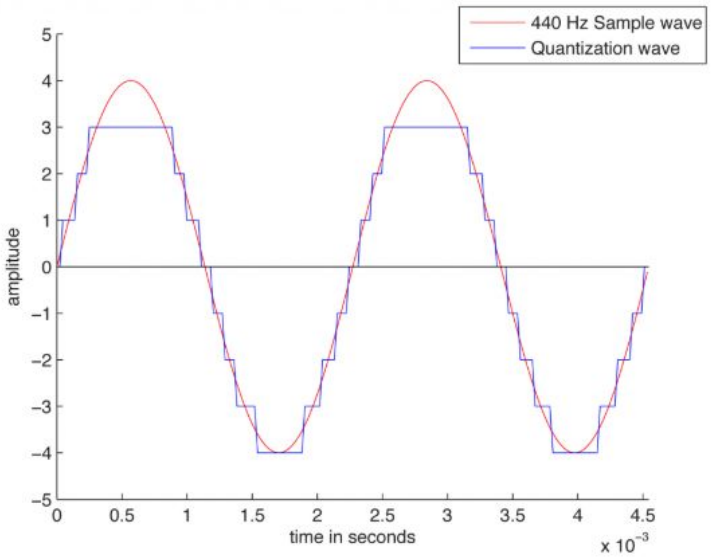
As Figure 11 explains, the digitization of music is essentially the conversion of analog to electrical and digital signals in a binary format. As sound is produced when objects vibrate such that human ears can pick them up, the waves produced are called analog signals. When these analog signals are to be used in digital devices, the sound waves must be converted to an electrical signal in a digital format of 0's and 1's. As seen in Figure 12, this process involves two important steps: sampling and quantization. The sampling component is when air pressure amplitude is measured in intervals.[11]

Figure 14: Sampling and Quantization[12]



[10] https://www.researchgate.net
[11] http://digitalsoundandmusic.com/5-1-2-digitization/
[12] https://dsp-nbsphinx.readthedocs.io

Figure 15: Sampling and Quantized Wave Example[13]



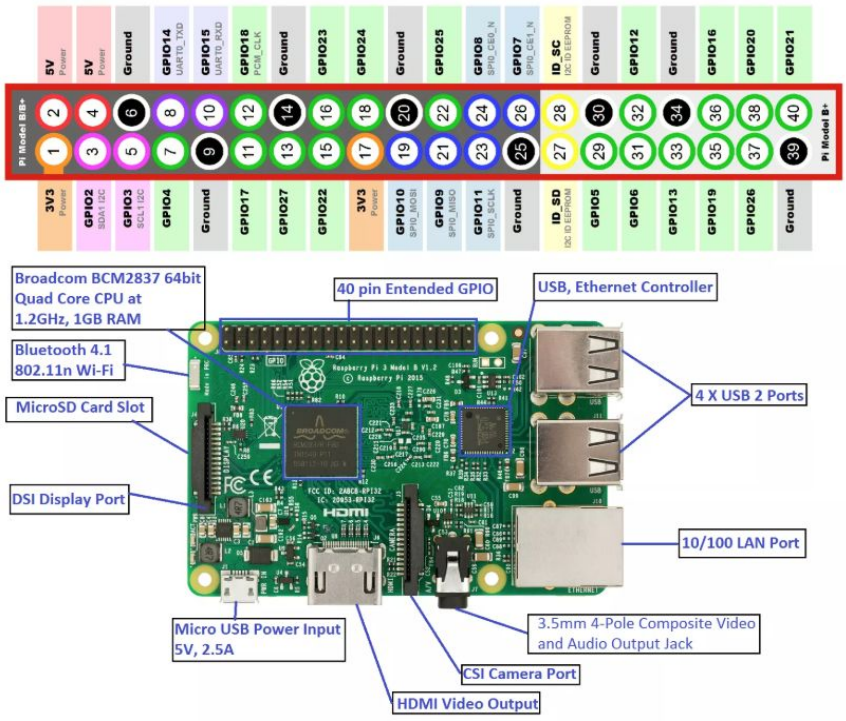
As seen in Figure 13, the changing air pressure of sound is represented by a sine function. The quantization component is when the air pressure amplitude is represented in binary integers. Quantization also describes the process of rounding off the sampled values to the closest value of the digital code being used. The sampling rate as well as the bit depth, or the range of integers produced in quantization that captures the precision of the represented amplitudes, are defined when sound is recorded in digital format. As seen in Figure 13, there is a difference between the original and quantized samples which is due to rounding error, or noise, due to bit depth. During digitization, when a microphone detects changes in air pressure, it sends the corresponding voltage changes down a wire for the values to be sampled. The quantity of these values in an interval helps to set the frequency of the audio signal. The higher the frequency, a greater sampling rate is needed to represent the signal digitally. The lower the frequency, a smaller sampling rate is needed to represent the signal digitally. However, when the sampling rate is insufficient, there is an incorrect digitation of the sound frequency. As a result, in order for the correct digitaion of a sound wave, the sampling rate should be at least twice the frequency of the sound wave.[14]

When sound is digitized into the form of audio files, it allows computers to read and play them. Because my modern jukebox relies on music files in MP3 format, I am using a Raspberry Pi to store the files and run code that allows me to play a song when specific push buttons, or specific inputs, are activated.

---

[13] http://digitalsoundandmusic.com/5-1-2-digitization/
[14] Ibid.

Figure 16: Diagram of Raspberry Pi[15]



A Raspberry Pi is essentially a small sized computer that runs a Linux operating system originally designed for learning how to program or giving 3rd world countries full access to a fully functioning computer, but today, they are commonly used by hobbyists for making projects. It has also been compared to an Arduino but the difference between the two is that the Arduino is a microcontroller that can only run a single program at a time whereas the Raspberry Pi is for general purposes and can run multiple.[16] Figure 15 depicts a diagram of a Raspberry Pi with all of its components from audio outputs, USB ports, GPIO pins, ethernet controllers, and more. Its internal storage comes from a micro SD card in which users must install a Raspbian operating system on it. Through the USB ports on the RPi, users can connect an external monitor, keyboard, and mouse just like a full computer.

[15] https://electronicshobbyists.com
[16] https://electronicshobbyists.com/tutorial-1-what-is-raspberry-pi-getting-started-with-raspberry-pi-3/

Results

In terms of specs, the RPi that I was using had an SD card with 32GB of space. Theoretically it can more than 55 mp3 and 55 png files, but for the sake of my project and maintaining the selection feature of the classic jukebox, I kept it at the number I did.

In terms of code, the Modern Jukebox will display an image for the current song playing as well as maintaining the display of the song selection. When observing users at the Maker Faire, typically I didn't have to give much instruction. People would approach, take a look at the song selection, and press the corresponding buttons. The only instruction that I had to give occasionally, was to click the done button after selecting the two songs so that their chosen song would play. There was one occurence where a teacher approached and clicked a bunch of the numbered buttons and noticed nothing was happening. This would be due to the fact that there should have been an else statement after the array of buttons pressed reached 3 to check if its not the done button, then display a message stating to the user that their input is invalid. People never seemed to make a mistake when choosing a song using the two digit number, so unfortunately no user testing was performed with the use of the clear button.

Figure 17: Current, Voltage, and Power Required for RPi Jukebox Function

Current Drawn (V)	**Voltage Used (V)**	**Power Output (W)**
.68 A	5.4 V	3.672 W

The chart above shows the power output of the Modern Jukebox when it is running. Initially, I had been powering the RPi using a USB connection to an outlet, but as soon as there had been a mouse, keyboard, monitor, and speaker connected to its HDMI and USB ports and headphone jack, I kept experiencing an undervoltage error with the 5 volts I was getting from plugging it into an outlet. This is when I made the decision to use the 5V and GND pins on the RPi and connect it to a power supply that would allow me to bypass the undervoltage error that I was getting and allow for the jukebox to function even if all these external components are connected.

Conclusion:

The Modern Jukebox is ready for use as a vintage, yet digital way of listening to music. Creating a mini jukebox brings back an era of poodle skirts, Elvis, and diners that seemed so fun to be a part of. I love music and being able to listen to it through my own version of a retro machine is rewarding. While the user can pick out of the song selection given to them, they are limited to a certain number of songs. In the past, the jukebox allowed for a physical and emotional connection to music and brought communities of people together. Nowadays, we can listen to whatever music we want, and sometimes it causes us to listen to the same style over and over again. Jukeboxes provide for an experience where the songs were limited and people had to wait for their turn, allowing them to listen to the choices of others and expanding their music taste by introducing them to new songs.

Finally, as the basics of the Modern Jukebox are complete, if given more time, I would want to add more decor to the project. For example, to achieve the vintage look of the classic jukebox, I would add two curved surfaces to the top of the jukebox case and use wooden dowels to connect the pieces in an aesthetic manner. I would also add LED strips for light and color as a final touch to bring the bright elegance of listening to music alive. I would also want to include a small LCD screen that would show in the moment, exactly the buttons the user is clicking when selecting a song. I feel that this would add a more intuitive touch being able to keep track of the digits being typed in as well as being able to clear the numbers an retype if a mistake is made.

Acknowledgements:

I would first like to thank Dr. Dann for being as excited about this project as me, and for putting a temporary ban on music in the lab until the jukebox was functioning. I appreciated the support more than you know as it kept me motivated the entire process! I would like to thank Mr. Ward for so kindly teaching Simon and I how to use the Band Saw for an hour on a Saturday. I have to say it might be one of my favorite tools in the lab now! I would like to thank Clark Kovacs and Nic Hernandez for always letting me crash B block ASR and for good friendly banter. Finally I would like to thank Alec Vercruysse for helping me with the more concise manner of optimizing my code using queues, and Will Buxton and Luke Arnold for requesting to add in their own special two digit combination for a secret song to play.

Bibliography:

[1]https://hackaday.io/project/25130, *Website accessed on 1/31/19*
[2]https://www.encyclopedia.com/science-and-technology/computers-and-electrical-engineering/computers-and-computing/jukebox, *Website accessed on 1/31/19*
[3]https://www.livescience.com/33793-record-players-work.html, *Website accessed on 2/10/19*
[4]http://www.8trackheaven.com/archive/work.html, *Website accessed on 2/03/19*
[5, 6]http://hyperphysics.phy-astr.gsu.edu/hbase/Audio/tape2.html, *Website accessed on 2/03/19*
[7]https://www.scienceabc.com/innovation/how-does-a-compact-disk-cd-work.html, *Website accessed on 2/13/19*
[8]https://www.scienceabc.com/innovation/how-does-a-compact-disk-cd-work.html, *Website accessed on 2/07/19*
[9]https://www.researchgate.net, *Website accessed on 3/20/19*
[10]http://digitalsoundandmusic.com/5-1-2-digitization/, *Website accessed on 3/18/19*
[11]https://dsp-nbsphinx.readthedocs.io, *Website accessed on 3/24/19*
[12, 13]http://digitalsoundandmusic.com/5-1-2-digitization/, *Website accessed on 3/24/19*
[14]https://electronicshobbyists.com, *Website accessed on 3/20/19*
[15]https://electronicshobbyists.com/tutorial-1-what-is-raspberry-pi-getting-started-with-raspberry-pi-3/, *Website accessed on 3/24/19*

Appendix A: Parts List

Part Description	What Needed For	Cost	Where to Buy
RPI HDMI monitor (10 inch)	To display tracklists, album covers, and song names	$50	Amazon
White push buttons (12)	Selecting songs	$5	Arcade Store
Speaker	Amplify music	$15-30	Amazon
RPi 3B+	Digitization of Music	$35	Lab
LED Strips	Decoration		Lab
Protoboards	Soldering buttons		Lab
Wood (1/4 inch)	Case		Lab