Patrick Faley, Cesar Parra-Castro, Matthew Peine

EE 10115 Embedded Systems

Professor Scott Howard

4/24/2019

**Embedded Systems Arduino Project Report**

**Executive Summary**

Our project was to use an Arduino to create a music player. Sound is played through a Piezo buzzer controlled by the Arduino, with the current song being displayed on an LCD display. The user can adjust the volume through a potentiometer, and there are two buttons which allow the user to pause/play the current song as well as toggle to the next song. The player is also reconfigurable, and any file in the RTTTL format can be loaded onto the Arduino with the help of a Python script we wrote, making the project reconfigurable.

**Background**

The main motivation for our project came out of the lecture on Piezo buzzers and using the tone command to control them. Two of us are from Stanford hall, and we have our own fight song. So after the Piezo lecture, we figured out how to play our fight song on the Piezo using the Arduino. This project was very enjoyable, and the sound of the song coming from the Piezo was interesting and reminded us of old video game soundtracks. The single song we programmed in was definitely entertaining, and we decided it would be even more entertaining to expand the capabilities of our Arduino-powered music player. If nothing else, there is definitely a novelty to hearing famous songs played on an Arduino. Our project has been an effort to explore the possibilities of Piezo-based music, and to create something fun and engaging in the process.

While the idea for our project did not come from a pre-existing project, we did look to see how other people had managed to play music on an Arduino. We stumbled across the idea for the RTTTL music format from an Instructables blog written by makeosaurus, found at <https://www.instructables.com/id/RTTL-Tunes-on-arduino/>. However, our project is significantly different from what was implemented in that blog. Their solution to playing multiple songs was to paste several RTTTL strings directly into the Arduino sketch and comment out all but whatever was currently being played. They then parsed the file onboard the Arduino and played it from there. Our project, by contrast, is able to play multiple songs, control volume, display the current song name, and pause the audio.

Several devices exist in the marketplace which can play a similar role to our audio player. The RTTTL file format was first invented for use in ringtones on Nokia phones, which would give a similar ability to play music to that of our Arduino. Additionally, several software applications exist which can play RTTTL files. The most similar product to our project would be an MP3 player or an iPod. However, these devices may have a different functionality and not be able to play RTTTL files specifically.

**Functional Description and Product Operation**

Our product functions similarly to how an MP3 player would function. It is able to play an arbitrary number of songs written in the RTTTL format, assuming that they do not exceed the memory limits of the Arduino. Through interaction with the LCD display, potentiometer, and push-buttons, the user can see the current song, control the volume, change track, and pause or play the current song.

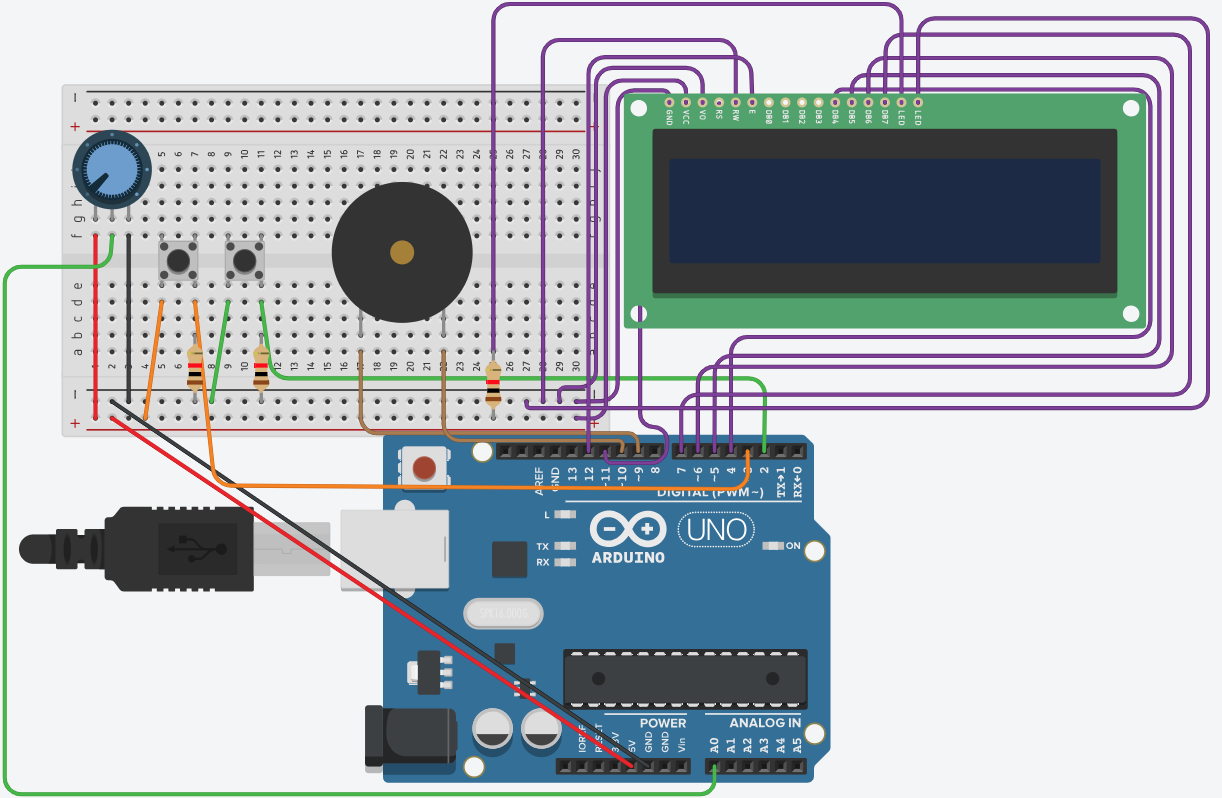
There are two main ways to interact with our project: loading songs and playing songs. To load the songs, you can run RTTTL\_Parser.py. Within RTTTL\_Parser.py, there is space to write a list of songs to be included in the Arduino code, along with how many octaves to offset each song by. This was done in order to allow for the correction of any overly high-pitched and annoying songs. Once these lists are specified, the user can run RTTTL\_Parser.py. Doing so will generate a few lines of C code saved in “OutputCode.ino”, which can be copy-pasted into Final\_Code.ino in order to configure everything for the desired songs.

Once set up with the desired songs, the user can interact with the sensors and actuators to gain a richer experience. The Piezo buzzer acts as a speaker, playing whatever songs are stored on the Arduino. The LCD display shows the current song, so the user can see what song they are currently playing. A potentiometer acts to control the volume. When fully to one side, the Piezo will be muted. Rotating the potentiometer will cause the volume to increment accordingly. There are also two push-buttons attached to interrupts. The first button increments the current track. Whenever the user presses this button, the current song will change. The second button acts as a pause or play button. If currently paused, no audio will be played by the Piezo. If currently in play mode, the Arduino will loop through its current song indefinitely. Pressing the pause/play button toggles between these two modes.

**Hardware**

Our product uses several sensors and actuators. This includes the following:

* Piezo Buzzer: This is an analog actuator which produces a tone when fed a PWM signal. This tone can be controlled by altering the PWM signal given. Through the Arduino, one can control the frequency, volume, and duration that this buzzer sounds at.
* Liquid Crystal Display (LCD): This is a 16x2 display capable of displaying text. It can be interfaced with the Arduino to control what is displayed in each of the 32 cells. The LCD is used to display the name of the song that is currently playing on the piezo.
* Potentiometer: This acts as an analog input. The voltage between the wiper and ground can be changed by twisting the knob, allowing the Arduino to detect the current position of the potentiometer, which makes it useful as a volume dial in our product.
* Push-Buttons: These buttons simply complete a circuit whenever pushed. They act as simple digital inputs and can be wired to the Arduino with a pull-down resistor to detect whenever they are pushed. Push buttons were chosen because they can be programmed to carry out different functions like interrupts.

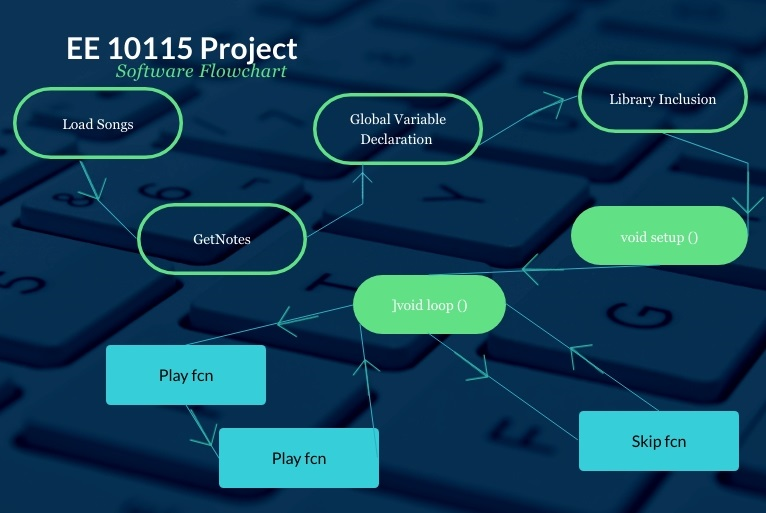


The first component, the potentiometer, has the wiper pin connected to analog pin A0 so that the Arduino can read the voltage across that pin. The two push-buttons are attached to pull-down resistors so that they register as high whenever they are pushed, and low otherwise. They are connected to digital pins 2 and 3 because they serve as the trigger for the interrupts. The piezo is connected to digital pins 9 and 10, which are both enabled for PWM and allow the toneAC library to play music through the Piezo. Instead of having the LCD wired up to a potentiometer to control the brightness of the screen, pin 3 of the LCD was connected straight to the ground rail.

**Software**

The software is divided into 6 main segments, which include the following:

* Global Variable Declaration: At the beginning of the program, global variables are declared. These include arrays storing the name of each song, length of each song, and the frequencies and durations of each song. There are also static variables controlling the current track and whether the product is in pause mode or play mode. The arrays in this section are generated by the Python script. Additionally we set up the LCD.
* Library inclusion: We had to load in the <LiquidCrystal.h> and the <toneAC.h> libraries for the code. LiquidCrystal allows us to wire and programs the LCD. The toneAC library allows us to use the toneAC function, which in contrast to tone, allows us to adjust volume and also gives us a maximum volume of twice that of tone.
* getNotes: This function takes a track number and an address inside the track as input, and returns a pointer to an array containing the frequency and duration of that note. This function is meant to work with the arrays generated by the Python script and allow the loop function to easily access any point in any song. The function works by taking the modulus (% operator) of the input track and address, allowing the function to work regardless of the values passed into it. It then reads the frequencies and durations arrays to find the correct songs, then reads the correct values from those songs, before storing the frequency and duration in a 2-element array and returning that array.
* setup: In this function, we set the pins 2 and 3 as input pins, and declare the interrupts for those pins for the skip and play functions. It builds on the previous setup of the LCD by setting it up as a 16x2 array, setting the cursor to the (0,0) position, and printing the name of the first song.
* loop: The code inside void loop () will only run if the value of pause is equal to 1. First, the program takes input from the potentiometer and then scales using the map function to make those values usable for adjusting the volume at which the piezo plays. The function then reads the track which is currently selected and uses the values of frequency and duration from the track arrays, combined with the volume value from the potentiometer to play the song using the toneAC function. After the song finishes playing, the variable ‘addr ’is incremented so the next song will play.
* play: This function, which is triggered by an interrupt, sets the pause variable to the negative of what it is at the moment. Because the loop function will only run if the pause variable is 1, this function will pause the track if it is playing, and play if the track is paused.
* skip: This function, which is triggered by an interrupt, increments the ‘addr’ variable so the next song is immediately played and also updates the LCD with the name of the new track.



**Design Post-Mortem**

Our final project did not differ significantly from what we outlined in our initial proposal. We left the proposal vague about how we would specifically load in the songs, and that was the only real design aspect which has been decided on since the initial proposal.

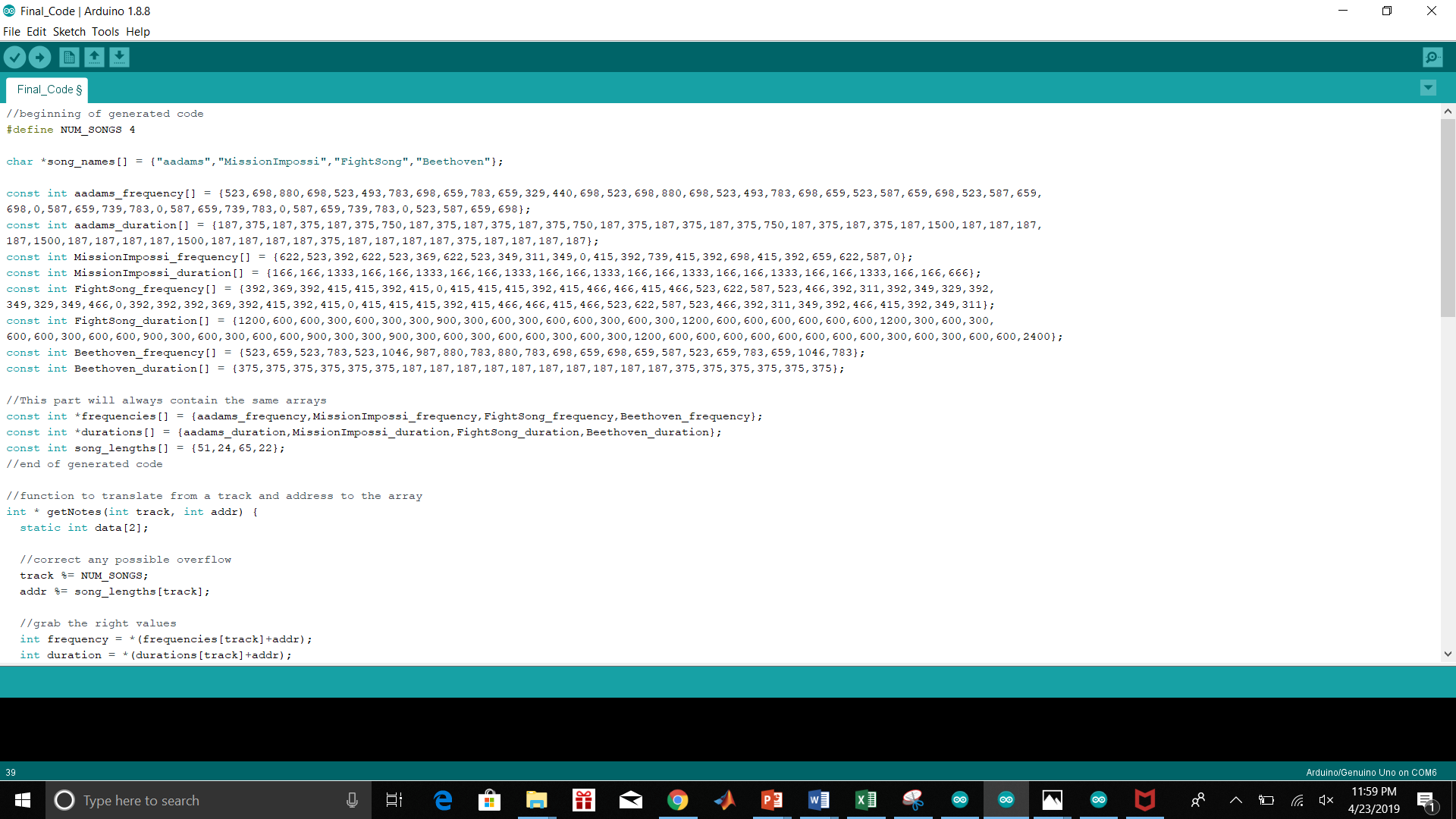
The most interesting part of our project, in our opinion, is the way in which we load in songs. RTTTL is a fairly common song format, and many songs have been converted into this format. Additionally, the format makes it fairly straightforward to manually program music, which we managed to do with the Notre Dame fight song. The C function used to interact with the output from our Python script was also fairly interesting, because in order to implement it we had to learn how to work with pointers and how arrays are stored in memory.

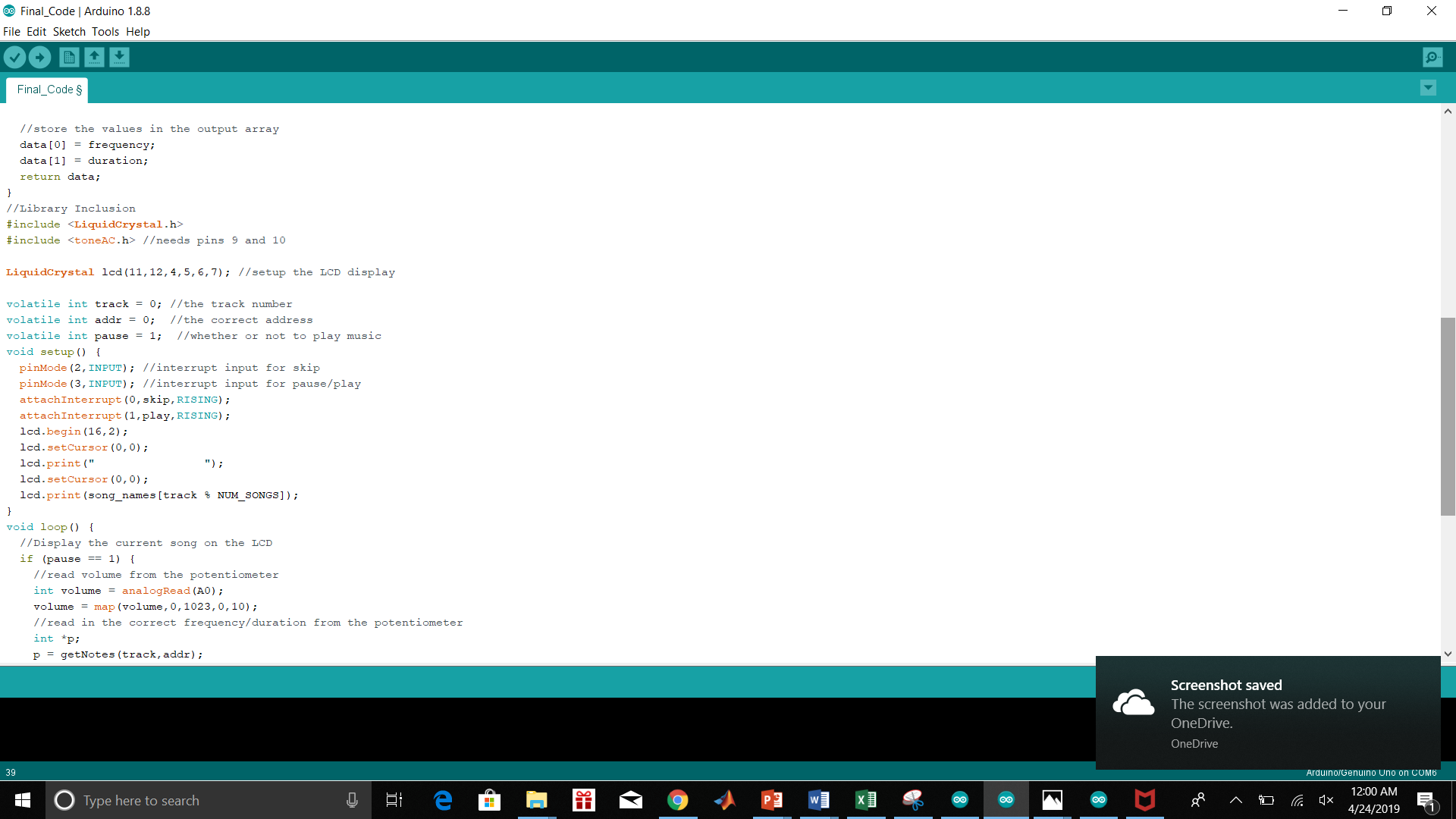
The most difficult part of the project was figuring out how to upload songs on the Arduino. We examined .wav, .mp3, and .mid files before arriving at RTTTL files. It was hard to figure out how to efficiently store any sort of song on an Arduino and play it back, especially with the limited capabilities of the Arduino.

If we were to do this project again, we would probably attempt to improve the sound quality. A single Piezo buzzer is fairly limited in what it can accomplish, so adding a few more may have given us the ability to play multiple frequencies at a given time, which would allow for more complicated songs. Alternately, we could have looked into purchasing a speaker and not relied on the Piezo buzzers in our kits.

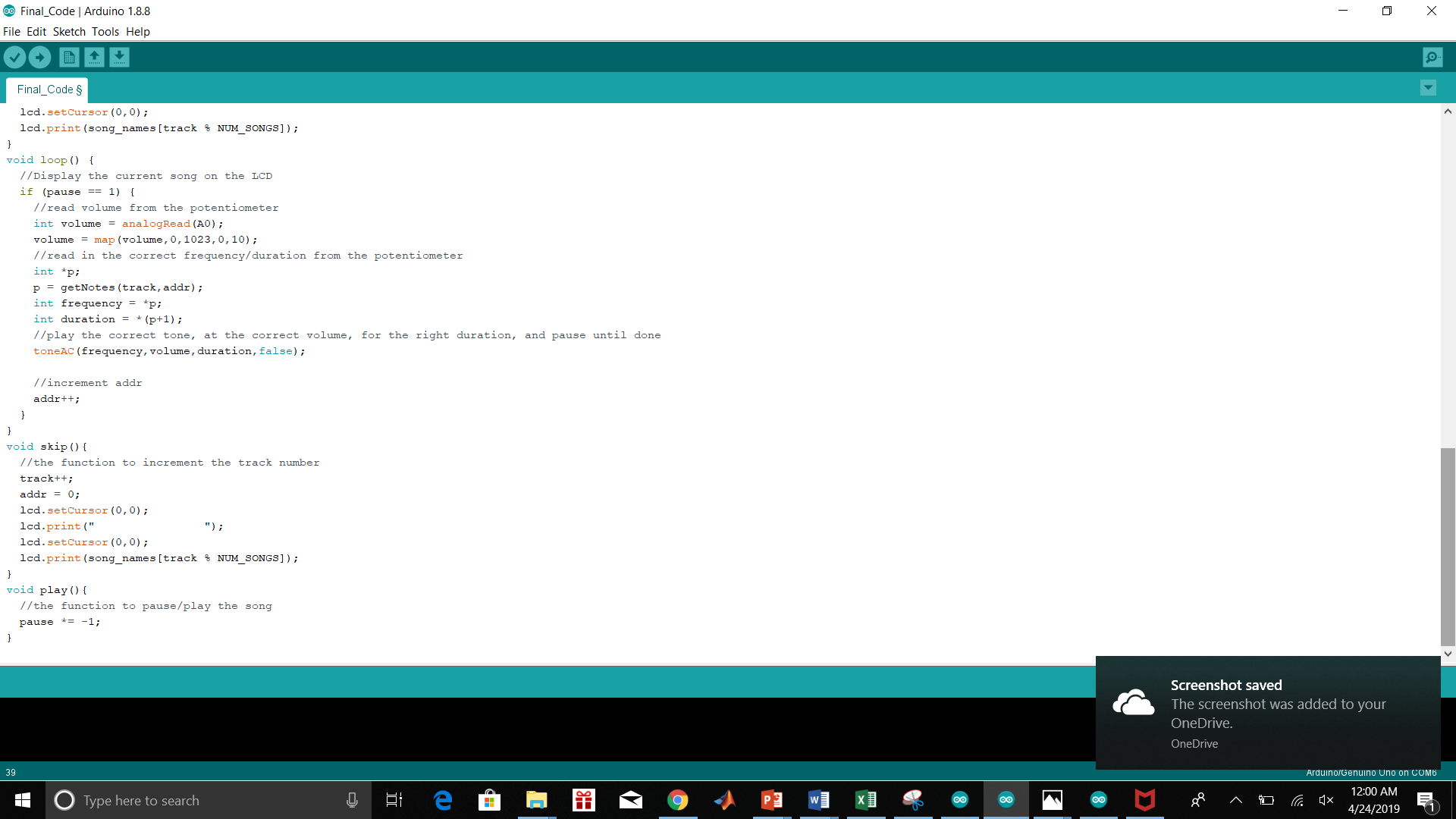
**Appendix**

C Code

****

****

Python Code

****

