# Guitar Says
# CS 435 Final Project Report

Joshua Larson, Noah Thornton, Saul Romero

California State University San Marcos

December 12, 2023

## Abstract

Inspired by the games “Simon Says” and “Guitar Hero,” this project, "Guitar Says," aimed to design an interactive game for enhancing memorization skills by replicating sequences to classic tunes. The user selects a song from an LCD screen, listens to it, replicates sections prompted by the LEDs and buttons, and receives a performance score. Unlike similar projects, our code was developed from scratch, incorporating concepts from the Digital Input/Output, Analog Input/Output, and Serial labs. The hardware setup includes a Nucleo F401RE board, LCD display, shift register, Piezo buzzer, potentiometer, four LEDs, four buttons, two breadboards, and multiple resistors.

The conceptual design guides users through song selection, gameplay, and scoring. Each song has the following level of difficulty: “Twinkle-Twinkle Little Star” easy, “Happy Birthday” medium, “Star-Spangled Banner” hard.

From this project, we learned the importance of starting projects early, clear communication, and the efficiency of team programming. Challenges in code implementation were overcome with creative solutions, like using a separate Char array to successfully compare notes.

The system's functionalities include SPI for LCD printing, a potentiometer for song selection, buttons for note input, and LEDs for visual cues. The code is structured for easy adaptation and expansion. Future improvements could enhance wiring, packaging, and song selection, making the game more appealing and engaging. "Guitar Says" successfully blends technical proficiency, collaborative effort, and problem-solving, providing a fun, yet challenging experience for users.


## Introduction

Our project aimed to design an engaging game that allows a player to work on their memorization skills by replicating sequences to classic tunes. The inspiration for this project comes from the classic games “Simon Says” and “Guitar Hero.” This project will allow the user to select one of three pre-programmed songs from an LCD screen, prompt the user to replicate a section of a melody and output a score based on the user’s performance.

Other projects have a similar concept to our project, however, we did not rely on other projects’ implementations to create our code. We developed our code from scratch, implementing the techniques learned from the Digital Input Output Lab, Analog Input Output Lab, and Serial Lab, along with data sheets [1, 2, 3] and previous curriculum knowledge, to successfully play sounds through the Piezo buzzers, create and map recognizable melodies correctly to LEDs and buttons, develop a menu driven system, properly wire all components, and correctly register a user’s button inputs (avoiding the infamous bouncing problem).

# Conceptual Design of the System

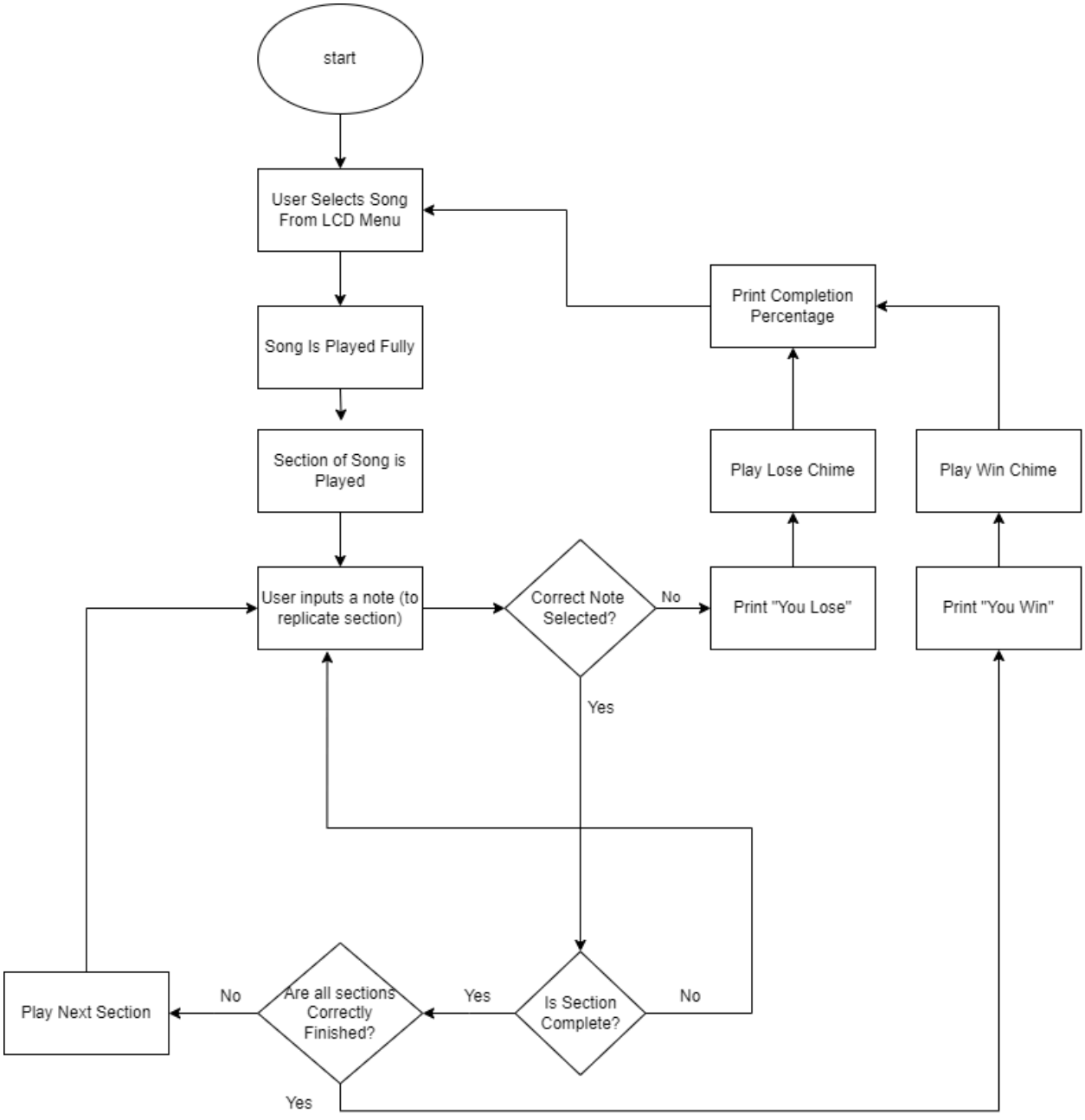

Fig. 1: Guitar Says Program Flow Diagram

The program flow for our project “Guitar Says” is simple. After the program is flashed, the user should direct their attention to the LCD display, where the titles of each available song will appear. After scrolling through the available songs using the potentiometer, the user should select the song they want to play with the select button located below the potentiometer. Next, the song will play once in its entirety. While the song is playing, the LEDs will turn on to the corresponding note played, revealing the winning combination of lights/ buttons to the user. After a short wait time, the user will be prompted with a sequence to replicate. After the small sequence is done, the user must replicate the pattern they just saw. The LEDs are directly above their corresponding buttons to limit confusion. If a user presses an incorrect button (meaning an incorrect note is played), the user loses the game. On the LCD screen, the user will see a “You Lose” message and will hear a lose chime from the speaker. Moments later, a completion percentage will be shown on the LCD display to show how close the user was to winning the game. Then, the player will automatically be returned to the main menu where they can choose another song to play again. If the user plays a correct note, the game will wait for the next correct note in the sequence to be pressed. Once a sequence is successfully repeated, the game will play the next sequence in the song to be completed. As before, the correct combination of notes and LEDs will be given, and the user will have to replicate it as seen. If there are no more sequences for a player to complete (meaning the song has successfully been played in its entirety), the player has won. On the LCD screen, a “You won” message will appear before returning the player to the main menu to play again. Every song has the following level of difficulty: “Twinkle-Twinkle” Easy, “Happy Birthday” medium, “Star-Spangled Banner” hard.

## Hardware Design/Setup

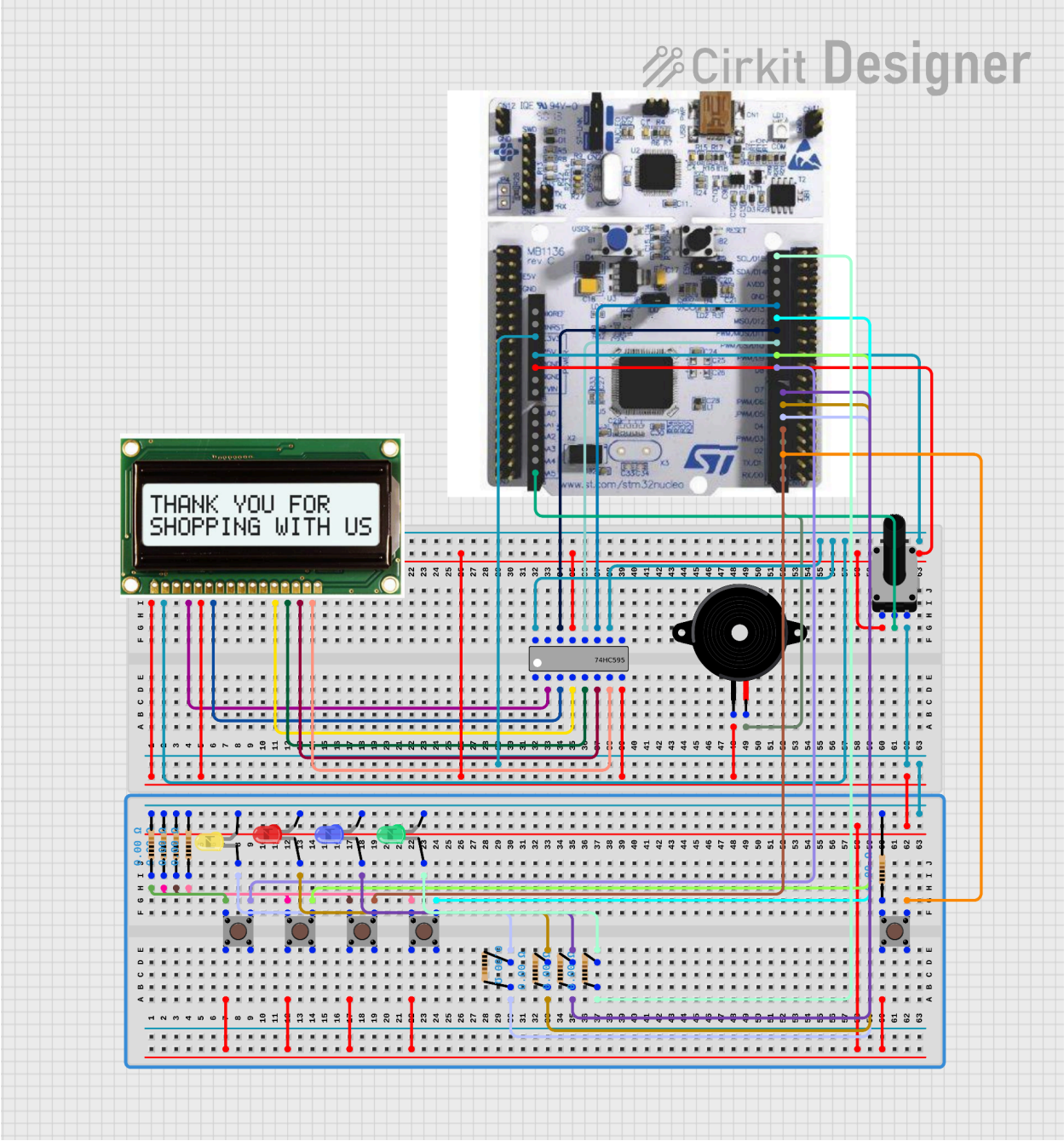


Fig.2: Hardware Wiring Diagram

Components

- One Nucleo F401RE Board
- One LCD (NHD_0216HZ) display
- One  shift register (74HC595N)
- One PWM speaker
- One potentiometer
- Four LEDs
- Five Buttons
- Four 100 ohm resistors
- Five 330 ohm resistors
- 2 Breadboards

Functionalities

- SPI is used in order to print characters to the LCD screen (with the use of the shift register) [3] [2] which will display a list of songs.
- The potentiometer is connected to an analog pin[1], and the voltage is used in order to iterate through the list array which will in turn be outputted to the LCD as the voltage changes.
- Speaker (buzzer) is connected to a PWM pin[1], it is used to output the frequencies of an array of notes, formulating a song.
- All 5 buttons are connected to D (digital I/O) pins[1]. The first one is used in the manner of an action event, such that whenever it is pressed it will trigger a series of actions that will play the indicated song on the LCD as well as prompt for user input and evaluate their performance. The remaining buttons are used together to simulate the action of entering notes. All buttons use 330 ohm resistors.
- LEDs are connected to D pins[1] and each are used with the respective “note” buttons, and are used to indicate which note belongs to a button. All LEDs use 100 ohm resistors.

## Software

Printing menu to LCD

Part of the idea of this project was to implement something that could be built with the provided kit as well as to apply everything covered in the course, therefore there are sections of code designed similar to how they were used in the labs. This is mostly the case for the functionality of printing information to the LCD, in which the entire code for building the “NHD_0216HZ” object was reused. To print the song titles, a char* array was used which respectively stores a song title per entry. To iterate through the array, the potentiometer voltage is read and casted into an int, then it is multiplied times the song name array size, and the resulting value is used as the index of the array.

```
      float potValue = potentiometer.read();
            scrollPosition = static_cast<int>(potValue * arraySize);

            if (scrollPosition >= arraySize) {
                scrollPosition = arraySize - 1;
            }
```

Fig. 3: Potentiometer implementation. “scrollPosition” is the variable used as an index to the song name array.

Selecting a song

Whenever the action button (button 1) is pressed, the string at the corresponding index will be used to determine the song to be used. Every song has 2 main functions: one that plays the song in its entirety and one that only plays the indicated section and proceeds to obtain user input.

Fig.4: Sample code that triggers a song:

```
if (button1 == 0) {
          // Button is pressed, turn on the corresponding LED
          if (strcmp(textArray[scrollPosition], "Star-Spangled Banner")
== 0) {
              int stuf  = 0;
              bool stop = true;
              //led1 = 0;
              switch (stuf) {
              case 0: starFull(); stuf++;
```

```
                wait(1);
                case 1: stop = starSections(0, 12);stuf++; if(!stop)break;
                case 2: stop = starSections(12,24);stuf++; if(!stop)break;
                case 3: stop = starSections(24,36);stuf++;if(!stop)break;

                //user wins and program plays full song
                clear_lcd();
                print_lcd("You Win");
                starFull();
                }
```

Playing a song

To play a song, a similar approach was used as the one used for the analog lab. For that, the frequencies of notes ranging from Do4-Sol5 were defined and were used for the songs, each of which consists of a note array (float) a beat array(float) and a name array (char*, described below). To play the song, the program iterates through the array and outputs the frequency of the note to the buzzer, and waits according to the beat value. To light up the corresponding notes with the LEDs, notes were distributed manually in a sequential/circular manner (as shown in the code below). The only exceptions to this approach were in cases of equal notes in different octaves (placed as far apart as possible) and cases where notes were mapped in a manner that was too easy given the level of difficulty of the song.

Fig. 5: Code that plays a song and light LEDs:

```
void starFull(){
   k = 0;  // start by considering the first music note

                while (k < sizeof(note) / sizeof(note[0])) {
                    led1 = 1;
                    led2 = 1;
                    led3 = 1;
                    led4 = 1;
                    if (note[k] == No) {
                        speak.write(0);
                    } else {
                        if (names[k] == "Do" || names[k] == "Do5") {
                            led1 = 0;
                            sequence[k] = 1;
                        } else if (names[k] == "Mi" || names[k] == "Re5") {
```

```
                led2 = 0;
                sequence[k] = 2;
            } else if (names[k] == "FaS" || names[k] == "Mi5")
{
                led3 = 0;
                sequence[k] = 3;
            } else if (names[k] == "So" || names[k] == "Fa5") {
                led4 = 0;
                sequence[k] = 4;
            }

            float frequency = note[k];
            speak.period(1 / frequency);
            speak.write(0.05);
        }

        wait(beat[k]);
        k++;
    }
```

The array "names[]" is used to store the musical notes of a song in string format (char*), this was done for the sake of clarity and simplifying implementation. This was determined to be more straightforward than using numerical frequency values, enhancing readability and ease of understanding the program.

Scoring user input

When the program prompts the user to replicate a note, it will use an integer and a conditional to determine which button was pressed. Then, a conditional will be used to determine if the button pressed matches the corresponding LED. If correct, the program proceeds to the next node, otherwise, it exists the song function. Since polling was used, a wait statement was used to prevent a button from reading excessive values.

To calculate the percentage of completion, the program simply counts the number of correct notes entered by a user. If a mistake is made, the size of the note array is used to determine the completion percentage.

## Lessons Learned

Many lessons were learned while developing “Guitar Says.” We learned the importance of beginning projects promptly and documenting a clear vision for the project given the time constraints of the course. We had thorough discussions of the scope of the project, making sure all team members were on the same page. We documented the agreed-upon scope and pinned the comment on Discord, our choice of communication channel, to be easily referenced. Additionally, we used Discord to share updated versions of the code and to coordinate the times all team members could meet.

Our project group found that team programming was the best solution to develop Guitar Says. Team programming, otherwise known as “mob programming,” is when all team members work on the project at the same time, in the same location, on the same computer. It is derived from pair programming. Our group faced difficulties when we tried to split tasks and merge code later on. Combining each team member’s knowledge during a single coding session wasted less time by allowing members to pose solutions to issues when other team members were stuck. We utilized multiple study rooms in Kellogg Library and Markstein Hall, spending many hours per session, to create this project.

The Digital In Digital Out lab, the Analog In/Out lab and the Serial Communication Lab provided the majority if the material consulted. We also reference the techotronic Arduino Music player project to implement better-sounding frequencies so each song is more recognizable [7]. We also referenced sheet music to determine the correct melody for each preprogrammed song, rather than recreating each song from scratch [4, 5, 6].

Our team faced many challenges in creating this project. Many challenges were in code implementation. Developing the menu-driven structure was difficult. The initial version of the menu-driven system did not loop properly and allowed the user to continue playing through a selected song despite if an incorrect button was pressed. This was solved by implementing a switch statement and counter to successfully iterate through the sections while breaking the switch statement if a user error occurs. Mapping the notes to LEDs and buttons was time-consuming. We learned that we needed a separate Char array to evaluate which LEDs needed to be illuminated and which buttons we were expecting to push. Comparing notes[k] to a predefined note produced an error, so a separate Char array holding the note names in strings (ex: “Do”, “Mi”, etc.) allowed for correct comparisons and LEDs to be lit. Initially, we attempted to implement the audio portion with Interrupts but found that it would be too complex to implement given the class’ time constraints. We found a more intuitive solution in polling. The original frequency values we used from the Serial Lab for the PwmOut speaker created songs that were unrecognizable to the user. We found that some notes needed a frequency change, and we needed more notes to accurately recreate the songs. This led to another challenge of attempting to map more than thirteen discrete values to four LEDs and four buttons. Luckily, we implemented the solution of reusing buttons for

multiple notes. This design choice reduces the complexity of the game for the user. Lastly, there was a bug in some songs where an LED would not flash again if a previous note with the same LED was just played from a previous sequence. This would ruin the user experience, as they would not be able to see the proper combination of LEDs/ notes to replicate, thus losing every time. This issue was quickly resolved by shutting off each LED after a section was successfully played. This was a tricky bug that almost went undetected because it occurred so quickly.

Given more time, we would improve our project's wiring by hiding wires and reducing the number of wires used. Furthermore, we would package the project in a suitable project box to make it more presentable to consumers, use illuminating buttons corresponding to the LED colors used, and program more up-to-date and popular songs into the game.

## Summary

In summary, our project, "Guitar Says," achieved its goal of creating an engaging game to enhance memorization skills by replicating sequences to classic tunes. Drawing inspiration from "Simon Says" and "Guitar Hero," our program allows users to select preprogrammed songs, replicate melodies through multiple sequences, and receive a performance-based score.

Our code and hardware setup utilized concepts from the digital input/output, analog input/output, and serial communication labs. The game's conceptual design, provided in Fig.1, provides a clear program flow for users, from selecting a song to playing and receiving a score. The lessons learned during this project highlighted the importance of early project initiation, clear communication, and the effectiveness of team programming.

There were many challenges in code implementation, such as refining the menu system, solving the bouncing button issue, and mapping notes to LEDs. The decision to use a separate Char array to compare notes and reusing buttons for multiple notes streamlined the program's complexity.

Given additional time, multiple things can be added and improved, such as refining our wiring and packaging. Nevertheless, our game successfully provided an engaging experience through memorization and music.

## Code Listings

https://csusm-my.sharepoint.com/:f:/g/personal/romer381_csusm_edu/Er7W85a2cm9PqAjrpJeVQssBhST-DcH-egI2Z38QrLhyWQ?e=tnD6Fy

## Video Demonstration

https://csusm-my.sharepoint.com/:v:/g/personal/romer381_csusm_edu/Edsrx1vW7TtGg6nggzhJhVgB1kdbQ4j-UcZdj8E-8i1TTQ?nav=eyJyZWZlcnJhbEluZm8iOnsicmVmZXJyYWxBcHAiOiJPbmVEcml2ZUZvckJ1c2luZXNzIiwicmVmZXJyYWxBcHBQbGF0Zm9ybS

I6IldlYiIsInJlZmVycmFsTW9kZSI6InZpZXciLCJyZWZlcnJhbFZpZXciOiJNeUZpbGVzTGlua0NvcHkifX0&e=1O8jII

## References

[1] STMicroelectronics, “UM1724 user manual - stmicroelectronics,” STM32 Nucleo-64 boards (MB1136) - UM1724, https://www.st.com/resource/en/user_manual/um1724-stm32-nucleo64-boards-mb1136-stmicroelectronics.pdf (accessed Dec. 13, 2023).

[2] Philips Semiconductors, “Data Sheet 74HC595; 74HCT595,” 74HC595N-Philips-datasheet-7085704.pdf, https://datasheet.octopart.com/74HC595N-Philips-datasheet-7085704.pdf (accessed Dec. 13, 2023).

[3] New Haven Display International, “NHD-0216HZ-fl-YBW-C - newhaven display,” NHD-0216HZ-fl-YBW-C.pdf, https://newhavendisplay.com/content/specs/NHD-0216HZ-FL-YBW-C.pdf (accessed Dec. 13, 2023).

[4] P. Hill and M. J. Hill, “Happy birthday (to you) C major,” Happy Birthday (To You) C Major Sheet music for Piano (Solo) | Musescore.com, https://musescore.com/user/173585/scores/166951 (accessed Dec. 12, 2023).

[5] B. Dunnett, “Star Spangled Banner Piano,” Star Spangled Banner - Music Theory Academy - Easy Piano Sheet Music, https://www.musictheoryacademy.com/piano-sheet-music/national-anthems/star-spangled-banner/ (accessed Dec. 12, 2023).

[6] B. Dunnett, “Twinkle Twinkle Little Star Piano Music,” Twinkle twinkle little star - music theory academy - easy piano music, https://www.musictheoryacademy.com/piano-sheet-music/nursery-rhymes/twinkle-twinkle-little-star/ (accessed Dec. 12, 2023).

[7] Admin, “Arduino Music Player Using Piezo Buzzer,” Arduino Music player using Buzzer | Song with Arduino ,buzzer, https://techatronic.com/arduino-music-player-using-piezo-buzzer/ (accessed Dec. 12, 2023).