PERSONAL PROJECT

Pedolu Andreea – Emilia

ARDUINO PROJECT – PIANO PLAYER

Theme: an interactive piano which can play a song from a SD card with sound spectrum visualizer

TABLE OF CONTENTS

PURPOSE OF THE PROJECT

For my first Arduino project, I wanted to create a simple yet engaging application to familiarize myself with Arduino and it's components, combining creativity with learning.

In order to have a nice experience and avoid potential difficulties, as I did not study Arduino in college , I decided to start with something easy like a piano and gradually increase it's complexity. For example I added a button which displays the song from a sd card and the LED matrix to display the spectrum captured by the sound sensor module.

Moreover, I wanted to develop something entertaining and accessible so that even people without a technical background could easily understand its functionality. All the information I acquired comes from online resources such as websites, forums and YouTube tutorials.

GENERAL DESCRIPTION

The project consists of three main parts:

<u>1) An interactive piano</u>

- Once a button is pressed it will send a signal to the multiplexor which is connected to the Arduino Uno board and the speaker will display the musical note corresponding to that button while the LCD display will show the name of the note.

- Before pressing a button the message “Hi” will be displayed.

<u>2) SD card music player</u>

- A specific button which once it’s pressed a song from the SD card plays.

- On the display, will appear the message “Playing music” and if it is pressed again the music stops and the message “Music stopped” will be displayed on the LCD.

- If a note button is pressed while the music is playing the music immediately stopes.

<u>3) Sound spectrum visualizer</u>

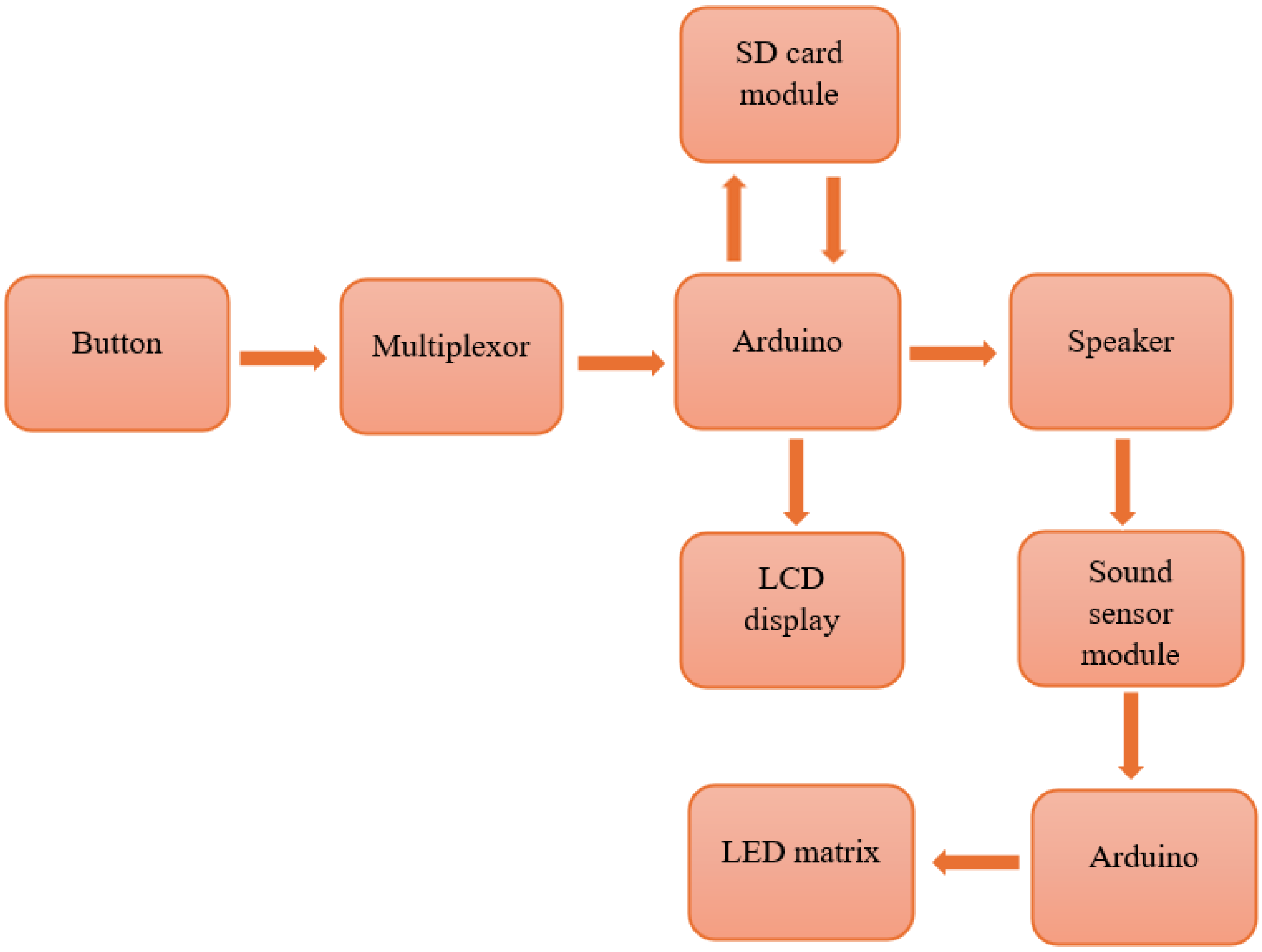
- A sound sensor captures audio input, including the song from the SD card or the notes played in the piano.

- An 8x8 LED matrix displays the frequency spectrum of the detected sound in real time.

- Currently, the LED matrix doesn't display the spectrum corectly, but I am still working at this part and after further revisions and improvements everything will function as expected.

HARDWARE DESIGN

Block diagram



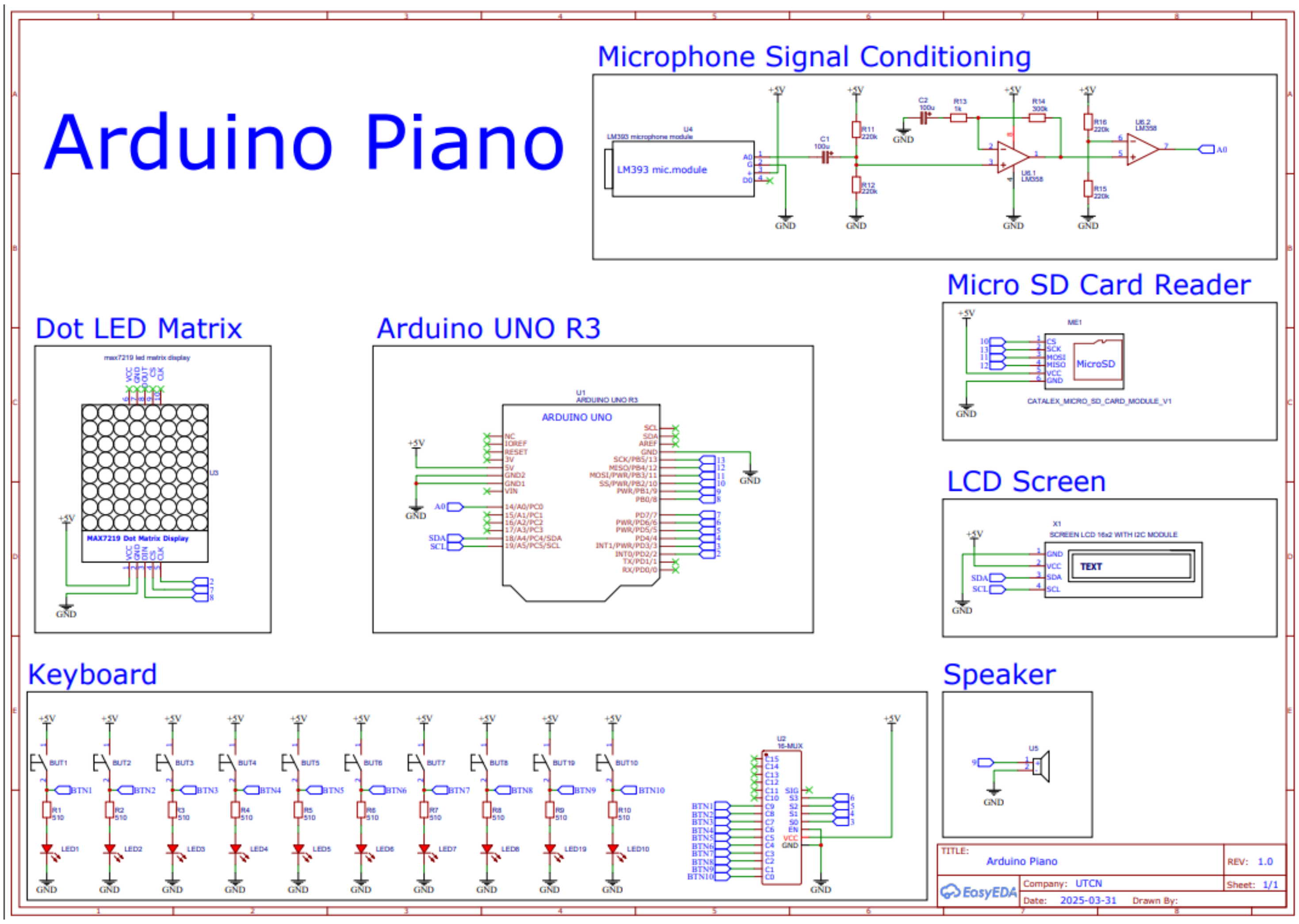
Working principle

Button presses are routed through a 16-channel multiplexer to the Arduino’s analog pins, which map inputs to specific notes using frequency tables. The SD card module streams audio files via SPI protocol, while the sound sensor’s analog output is processed with FFT algorithms for the LED matrix.

Components required to complete the project

- Arduino Uno board – The main microcontroller that processes inputs and controls outputs
- 10 Buttons
  - 9 buttons for musical notes (DO, RE, MI, FA, SOL, LA, SI, DO#, RE#)
  - 1 button to play/stop music from the SD card
- MicroSD Compatible Slot Card Module – To store and play the song.
- 1602 LCD with I2C Interface and Blue Backlight – to display text
- Miniature Mylar Cone Loudspeaker – To play musical notes and songs.
- 16-Channel Analog Digital Multiplexer – To connect the buttons in order tohave enough pins for the LED matrix and sound sensor
- Sound Sensor Module – To capture sound input for the spectrum visualizer
- 8x8 LEDs matrix module – To display the sound spectrum in real time
- 2 Breadboards
- Resistors
- Wires

Electrical scheme



SOFTWARE DESIGN

Significant parts

- ***Multiplexer & Button Handling Setup***

```
int muxS0 = 3;
int muxS1 = 4;
int muxS2 = 5;
int muxS3 = 6;
int muxSIG = A0;
```

The 16-channel multiplexer is connected to the Arduino on the pins 3, 4, 5, and 6, which are declared as muxS0, muxS1, muxS2, and muxS3. These pins are used to select which channel of the multiplexer is active (from 0 to 15).

The signal pin, muxSIG, is connected to analog pin A0. This is where the selected channel's signal is read by the Arduino. In this project, the multiplexer is used to read multiple button inputs using just one analog pin.

```
void handleButtons() {
    for (byte i = 0; i < 10; i++) {
        SetMuxChannel(i);
        int buttonState = digitalRead(muxSIG);

        if (buttonState == HIGH) {
            noTone(buzzer);  //stop any ongoing tone before setting LED
            lcd.clear();
            delay(10);
            switch (i) {
                case 0:
                    if (lastButtonStateB10 == LOW && stateB10 == 0) {
                        counter++;
                        lcd.clear();
                        delay(10);
                        if (counter % 2 == 1) {
                            lcd.print("Playing Music");
                            sound.play("song.wav");
                        } else {
                            lcd.print("Music Stopped");
                            sound.stopPlayback();
                        }
                        stateB10 = 1;
                    }
                    break;
                //controls button 9
                case 1: lcd.print("Note: Re#"); tone(buzzer, 587, 100); break;
                //controls button 8
                case 2: lcd.print("Note: Do#"); tone(buzzer, 523, 100); break;
                //button 7
                case 3: lcd.print("Note: Si"); tone(buzzer, 494, 100); break;
                //button 6
                case 4: lcd.print("Note: La"); tone(buzzer, 440, 100); break;
                //button 5
                case 5: lcd.print("Note: Sol"); tone(buzzer, 392, 100); break;
                //button 4
                case 6: lcd.print("Note: Fa"); tone(buzzer, 349, 100); break;
                //button 3
                case 7: lcd.print("Note: Mi"); tone(buzzer, 330, 100); break;
                //button 2
                case 8: lcd.print("Note: Re"); tone(buzzer, 294, 100); break;
                //button 1
                case 9: lcd.print("Note: Do"); tone(buzzer, 262, 100); break;
            }
        } else {
            if (lastButtonStateB10 == HIGH) {
                stateB10 = 0;
            }
        }
        lastButtonStateB10 = buttonState;
```

This function handles the 10 buttons conected through the multiplexer. Each button is accessed using the for loop ”for (byte i = 0; i < 10; i++)”. I used byte i instead of int i to save memory. The “SetMuxChannel(i);” sets the multiplexer to listen to a specific input, in this case, the corresponding button channel. Afterwards the state of the selected button is read with ” int buttonState = digitalRead(muxSIG);”. If the button state is HIGH, meaning the button is pressed then any previously playing tone on the speaker stops so it doesn’t overlap (noTone(buzzer);). The LCD screen is cleard before showing a new message and with „switch(i)” it’s checked which button is pressed.

For the case 0, which coresponds to the button 10, the music starts playing from the file „song.wav” from the SD card orstops it, depending on the number of presses.

For the cases 1 to 9 the plays a specific tone for each note using the tone() function, and the LCD displays the name of the note.

- ***Playing a song from the SD card***

The song from the SD card is played using the TMRpcm library. The playback process is implemented within the „handleButtons()” function, which is continuously called in the „loop()” function.

When button 10 is pressed, the code checks if it's the first press or if the button has been pressed again. It does this by checking the stateB10 variable. If stateB10 is 0 (meaning the button wasn't recently pressed), it increments a counter (counter++), which alternates between starting and stopping the song.

The lcd.clear() function is used to clear the display, and a short delay (delay(10)) ensures the screen refreshes properly.

The program checks if the counter is odd or even using if (counter % 2 == 1). If odd, the song starts playing with sound.play("song.wav"), and the LCD shows "Playing Music". If even, the song stops with sound.stopPlayback(), and the LCD shows "Music Stopped".

The stateB10 = 1 ensures the button press is only registered once, and stateB10 is reset to 0 once the button is released, allowing it to be pressed again to switch between playing and stopping the song.

The stateB10 = 1; line ensures that the button press is only registered once, and stateB10 is set to 1 to prevent continuous activation. Once the button is released, stateB10 is reset to 0, allowing the user to press the button again to switch between playing and stopping the song.

This functionality is made possible by initializing the SD card with „SD.begin(SD_ChipSelectPin)" and setting the audio pin with „sound.speakerPin = 9" in the setup() function. The „sound.quality(1);" line ensures higher quality playback, and „sound.setVolume(5);" sets the playback volume.

- ***Sound sensor implementation***

```
   void readSoundSensor() {
       startMillis = millis();             // Store the current time
       signalMax = 0;                      // Reset max signal value
       signalMin = 1024;                   // Reset min signal value (max for
analog input)

       // Sample sound for a short window (50 ms)
       while (millis() - startMillis < sampleWindow) {
           sample = analogRead(sensorPin); // Read from the sound sensor (A1)
           if (sample < 1024) {            // Filter out any invalid readings
               signalMax = max(signalMax, sample); // Track highest reading
               signalMin = min(signalMin, sample); // Track lowest reading
           }
       }

       // Store the max and min values in circular buffers
       signalMaxBuff[index] = signalMax;
       signalMinBuff[index] = signalMin;

       index = (index + 1) % 8; // Move to the next index (looping from 0 to 7)
   }
```

This part of the code reads input from the sound sensor module connected to analog pin A1. The reading process takes place inside the „readSoundSensor()" function, which is called continuously in the „loop()" function.

At the beginning of „readSoundSensor()", the current time is stored using „startMillis = millis();". Then signalMax is initialized with 0 at the start of each sample window („signalMax = 0;") in order to later compare and find the maximum sound level detected during that time. Then signalMin is initialized to 1024, the highest possible value for the analogRead() function to find the minimum sound level during the sample window by comparing values.

The code then enters a while loop „while (millis() - startMillis < sampleWindow)" that runs for a short duration defined by „sampleWindow". During this time, the sensor is read repeatedly using „analogRead(sensorPin)", and the variables „signalMax" and „signalMin" which keep track of the highest and lowest sound levels detected within that time interval.

This gives a short reading of the sound activity in the form of a peak-to-peak amplitude, calculated by measuring the difference between the highest and lowest values during that time window, which represents the sound intenity of the environment.

These max and min values are saved into signalMaxBuff[index] and signalMinBuff[index], where index loops from 0 to 7. This way, the program stores sound readings in 8 positions, making it possible to update all 8 columns of the LED matrix based on sound levels over time.

The result of these readings is later used in updateLEDMatrix() to calculate how high each bar should be displayed on the LED matrix, creating a real-time visual equalizer effect.

- ***The LED Matrix Display***

```
void updateLEDMatrix() {
    for (int i = 0; i < 8; i++) {
        peakToPeak[i] = signalMaxBuff[i] - signalMinBuff[i];       // Amplitude
        displayPeak[i] = map(peakToPeak[i], 0, 1023, 0, maxScale);  // Scale it

        // Smooth the drop-off effect
        if (displayPeak[i] >= temp[i]) {
            temp[i] = displayPeak[i];
        } else {
            temp[i]--;
        }

        lc.setLed(0, i, temp[i], true);  // Set LED level
        delayMicroseconds(250);         // Short delay
    }
}
```

This part of the code is responsible for updating the LED matrix display based on the sound activity detected by the sensor. The update process happens inside the updateLEDMatrix() function, which is continuously called in the loop() function.

At the start of the updateLEDMatrix() function, a for loop is used to iterate through the 8 columns of the LED matrix, represented by the variable i. This loop ensures that each column gets updated individually.

For each column, the program calculates the peak-to-peak amplitude (the difference between the highest and lowest sound levels) from the data stored in signalMaxBuff[i] and signalMinBuff[i]. This value is stored in peakToPeak[i].

Then, the map() function is used to scale the peak-to-peak amplitude into a range suitable for the LED matrix, from 0 to maxScale. This scaled value is stored in displayPeak[i] and represents how high the LED bar should be for that column.

To create a smooth visual transition, the code checks if the new displayPeak[i] value is greater than or equal to the previous value stored in temp[i] (which holds the current height of the LED bar). If it is, the temp[i] value is updated to match displayPeak[i], causing the bar to rise. If the new value is lower, the bar gradually lowers by decrementing temp[i] by 1, avoiding an abrupt drop.

The lc.setLed(0, i, temp[i], true); command is used to update the actual LED matrix, setting the height of each LED bar according to the temp[i] value. A short delay of 250 microseconds (delayMicroseconds(250);) ensures the update happens quickly enough for a smooth visual effect.

The result of this process is a real-time visual equalizer effect, where the height of each LED bar represents the intensity of sound detected by the sensor, and the bars move smoothly with changes in sound levels.

The complete code

Here's the full code, with comments to make it easier to follow.

```
#include <Wire.h>  // For I2C communication
#include <LiquidCrystal_I2C.h>        //LCD library for I2C LCDs
#include <TMRpcm.h>                   //for audio playback from SD card
#define SD_ChipSelectPin 10           //chip select pin for SD card
#include <SPI.h>                      //SPI communication (used for SD card)
#include <SD.h>                       //SD card library
#include "LedControl.h"              //for controlling LED matrix display

//initialize audio, LCD, and LED matrix
TMRpcm sound;
LiquidCrystal_I2C lcd(0x27, 16, 2);
LedControl lc = LedControl(8, 2, 7, 1); //DIN=8, CLK=2, CS=7, 1 device
const int maxScale = 11;              //maximum LED scale height

//sound sensor setup
const int sensorPin = A1;
const int sampleWindow = 50;          //sampling time window in milliseconds
unsigned int sample;                  //variable to store analog read sample

//timing variables
unsigned long startMillis;
unsigned long timeCycle;

//signal tracking variables
unsigned int signalMax = 0;
unsigned int signalMin = 1024;
unsigned char index = 0;

//arrays to track signal peak and display values
unsigned int peakToPeak[8];
unsigned int displayPeak[8];
unsigned int temp[8] = {0, 0, 0, 0, 0, 0, 0, 0};//temporary buffer for smooth
LED update
unsigned int signalMaxBuff[8];
unsigned int signalMinBuff[8];

//multiplexer setup
int muxS0 = 3;
int muxS1 = 4;
int muxS2 = 5;
int muxS3 = 6;
int muxSIG = A0;//signal pin from multiplexer
int buzzer = 9;//buzzer pin

//button handling variables
int counter = 0;
int stateB10 = 0;
int lastButtonStateB10 = LOW;

//function to set the multiplexer channel
void SetMuxChannel(byte channel) {
    digitalWrite(muxS0, bitRead(channel, 0));
    digitalWrite(muxS1, bitRead(channel, 1));
    digitalWrite(muxS2, bitRead(channel, 2));
    digitalWrite(muxS3, bitRead(channel, 3));
    delay(5);
}

void setup() {
    Serial.begin(9600);
    Serial.println("Initializing...");

    if (!SD.begin(SD_ChipSelectPin)) {
        Serial.println("SD Initialization failed...");
        return;
    }
    Serial.println("SD Initialization done...");

    //setup sound parameters
    sound.speakerPin = 9;
    sound.quality(1);        //set higher quality
    sound.setVolume(5);      //volume level 0-7

    //initialize LCD
    lcd.init();
    lcd.backlight();
    lcd.print("Hi");
    delay(10);

  //set multiplexer control pins as output
    pinMode(muxS0, OUTPUT);
    pinMode(muxS1, OUTPUT);
    pinMode(muxS2, OUTPUT);
    pinMode(muxS3, OUTPUT);

    pinMode(buzzer, OUTPUT);// Set buzzer pin as output

    //initialize LED matrix
    lc.shutdown(0, false);   //wake up LED matrix
    lc.setIntensity(0, 1);   //set brightness (0-15)
    lc.clearDisplay(0);      //clear display
}
```

```
void loop() {
    readSoundSensor();   //sample audio levels
    updateLEDMatrix();   //display levels on LED matrix
    handleButtons();     //read button inputs
}

//function to read audio sensor and store min/max signal
void readSoundSensor() {
    startMillis = millis();
    signalMax = 0;
    signalMin = 1024;

    //collect samples for a given window
    while (millis() - startMillis < sampleWindow) {
        sample = analogRead(sensorPin);
        if (sample < 1024) {
            signalMax = max(signalMax, sample);
            signalMin = min(signalMin, sample);
        }
    }

    //store values in circular buffers
    signalMaxBuff[index] = signalMax;
    signalMinBuff[index] = signalMin;
    index = (index + 1) % 8; //move to next index
}

//function to update LED matrix based on sound levels
void updateLEDMatrix() {
    for (int i = 0; i < 8; i++) {
        peakToPeak[i] = signalMaxBuff[i] - signalMinBuff[i];       //amplitude
        displayPeak[i] = map(peakToPeak[i], 0, 1023, 0, maxScale);  //scaleing
it

        //drop-off effect
        if (displayPeak[i] >= temp[i]) {
            temp[i] = displayPeak[i];
        } else {
            temp[i]--;
        }

        lc.setLed(0, i, temp[i], true);  //set LED level
        delayMicroseconds(250);         //short delay
    }
}

void handleButtons() {
    for (byte i = 0; i < 10; i++) {
        SetMuxChannel(i);
        int buttonState = digitalRead(muxSIG);

        if (buttonState == HIGH) {
            noTone(buzzer);  //stop any ongoing tone before setting LED
            lcd.clear();
            delay(10);
            switch (i) {
                case 0:
                    if (lastButtonStateB10 == LOW && stateB10 == 0) {
                        counter++;
                        lcd.clear();
                        delay(10);
                        if (counter % 2 == 1) {
                            lcd.print("Playing Music");
                            sound.play("song.wav");
                        } else {
                            lcd.print("Music Stopped");
                            sound.stopPlayback();
                        }
                        stateB10 = 1;
                    }
                    break;
                //controls button 9
                case 1: lcd.print("Note: Re#"); tone(buzzer, 587, 100); break;
                //controls button 8
                case 2: lcd.print("Note: Do#"); tone(buzzer, 523, 100); break;
                //button 7
                case 3: lcd.print("Note: Si"); tone(buzzer, 494, 100); break;
                //button 6
                case 4: lcd.print("Note: La"); tone(buzzer, 440, 100); break;
                //button 5
                case 5: lcd.print("Note: Sol"); tone(buzzer, 392, 100); break;
                //button 4
                case 6: lcd.print("Note: Fa"); tone(buzzer, 349, 100); break;
                //button 3
                case 7: lcd.print("Note: Mi"); tone(buzzer, 330, 100); break;
                //button 2
                case 8: lcd.print("Note: Re"); tone(buzzer, 294, 100); break;
                //button 1
                case 9: lcd.print("Note: Do"); tone(buzzer, 262, 100); break;
            }
        } else {
            if (lastButtonStateB10 == HIGH) {
                stateB10 = 0;
            }
        }
        lastButtonStateB10 = buttonState;
    }
}
```

CURRENT STATUS

The base functions of the project, including the Piano Mode and Music Player, are fully operational. In Piano Mode, the buttons correctly play notes, and the music player can play/pause songs stored on the SD card.

However, I noticed that the LED matrix wasn’t displaying the sound frequencies the way it should. After checking the MAX7221 datasheet, I found out that the matrix driver actually communicates using a serial protocol that needs specific digital pins (10, 11, 12, and 13 on the Arduino), while in my case I had connected the matrix to pins 2, 7, and 8 instead of the correct communication pins.

The problem is, those same pins (10 to 13) are already being used by the SD card module. Since both the SD card and the matrix need to use the same serial communication lines, connecting them at the same time without proper coordination can cause them to interfere with each other. Basically, both devices are trying to talk over the same wires, and if they do it at the same time, the data gets messed up, which explains the weird behavior I saw on the matrix.

To fix this, each device needs its own unique Chip Select (CS) pin. This pin acts like an "on/off switch" for communication. Only the device whose CS pin is pulled LOW will be active and allowed to communicate. The others stay quiet with their CS pins set to HIGH. As long as only one device is active at a time, they can safely share the same serial lines without interfering with each other.

CONCLUSIONS

I think with this project, I learned a lot of new things, and while working on it, I kept getting more and more ideas for future projects. In the beginning, I thought it would be pretty difficult, but now that it’s almost finished, I feel the opposite. As my first project made entirely by myself, without any guidance and only using information from the internet, I think it turned out well and was a great opportunity to develop my knowledge.

One thing I particularly enjoyed was the joy of seeing each component finally work after adding it. It was a great experience to create something physical with a real purpose.

Future work will focus on refining the sound spectrum visualizer and expanding features. As a first independent project, it has solidified my confidence in pursuing more advanced embedded systems applications.

BIBLIOGRAPHY

The book "Introducere în Arduino", by Optimus Digital
https://arduinogetstarted.com/tutorials/arduino-lcd-i2c

https://projecthub.arduino.cc/Dziubym/controlling-8x8-dot-matrix-with-max7219-and-arduino-0c417a

https://randomnerdtutorials.com

https://www.google.com/url?sa=t&source=web&rct=j&opi=89978449&url=https://lastminuteengineers.com/arduino-micro-sd-card-module-tutorial/&ved=2ahUKEwicu9mE26-MAxU-gP0HHZKOEUMQFnoFCIQBEAE&usg=AOvVaw3QcMQ1ulq7ZyJA10Ct8IKn

https://randomnerdtutorials.com/guide-to-sd-card-module-with-arduino/

https://forum.arduino.cc/t/sd-card-reader-connection-to-arduino-uno/1025932

https://www.google.com/amp/s/www.instructables.com/Micro-SD-Card-Tutorial/%3famp_page=true

https://randomnerdtutorials.com/guide-for-microphone-sound-sensor-with-arduino/

https://youtu.be/6DRayPCqK04?si=WP8pLxRdsyVs1f2a

https://youtu.be/UN9XPWHamHw?si=bEVFcp07ncF4qt9P

https://youtu.be/sS_oW81NweI?si=bDBHyud24kOy9cWw

https://youtu.be/7WiSeQxb1bU?si=8lxRRJMNe_78QCji

https://youtu.be/Q9fPfiPWsAg?si=NJG7DzjhMEProXm8

https://youtu.be/3awCkLS7gHI?si=AEiCdN_HqMty7Xqc

https://youtu.be/cd04o5yqSAU?si=m5dB0RenbUy1gU9t

https://youtu.be/xQfC72VeV7Y?si=c_fZQXfUc3Y9ilhC

https://youtu.be/H6Vs98-Cev0?si=YNcSUGTKQ64L4e4r

https://youtu.be/Dco6jo9xgAo?si=zvSm_A48EOACd7EW

https://youtu.be/a1Kp1OtSwu8?si=RMzi7GOaMv1eTj0h

https://youtu.be/oeK9KaQv8Qk?si=91Fdxbf1oK1ctD72

https://www.instructables.com/How-to-Use-a-Sound-Sensor-With-Arduino/

https://www.youtube.com/watch?v=9cxAjRHdMVY&ab_channel=RootSaid-Robotics%2CElectronics%26Technology