**Arthur Mezheritskiy**

**Jolekha Begum**

**Saurav Das**

**Electronic Keyboard with Arduino**

This project was an attempt to make an electronic keyboard using Arduino. We made an electrical keyboard with Arduino which contains a row of 13 buttons. When the buttons are pressed, our program produce different sounds. And 13th button to play a demo song. We used *Super Mario* as a demo song to play. Pressing one of the 12 buttons makes different tones according to the tone function’s signal sent to the buzzer by the Arduino, with the note set by the button and the octave set by the 10k potentiometer. One end of each push button is connected to a digital pin and the other end to ground. The data from the push buttons are read by the digital pins. When the button is pressed, the digital value reading changes as well. The potentiometer is also connected to analogIn. Since a modern piano has just over 7 octaves range, the potentiometer’s input range (0-1027) will be divided into 8 ranges, each one being an octave. Therefore, the lowest note that can be played is 𝐶0 (16.35 Hz) and highest note is 𝐵8 (7902.13 Hz). The 13th button, separate from the keyboard, will simply play a demo tune when pressed. Input from the keyboard will temporarily be suspended, and a demo song, pre-set as an array of keyboard inputs, will be played instead.

For this project we used some components those are given below.

1. Arduinos (2)
2. Breadboard
3. 5v Batteries (2)
4. Push Buttons (13)
5. 10K Resistors (13)
6. 10K Potentiometer (1)
7. Buzzer
8. Wires

Since one of our project requirement is to use two Arduinos and communicate between them, we used two Arduinos. They communicate with each other. Question is how did we use two Arduinos? As we mentioned before, we used 13 buttons. We know that it’s not possible to read data of 13 buttons from only one Arduino. So, we connected 7 buttons to one Arduino and 6 buttons to the other Arduino. As a result, if we press any one of 13 buttons, our program can understand which button is pressed and do his job accordingly. Here our both Arduinos communicate with each other, and we meet one the most important requirements of the project.

We used 13 Resistors(10K) for our 13 push buttons. Piano has 12 notes. They are C, C#, D, D#, E, F, F#, G, G#, A, A#, B. The reason behind to use 12 buttons is to play these 12 notes of piano. As we mentioned before, the last button is to play demo song.

We used a potentiometer to determine the octave. When one button is pressed then one function takes the frequency of one note (different buttons make different note) and return another new frequency according to the value of the potentiometer. That function can transform the original note’s frequency into 7 different frequencies, because a piano has a range of approximately 7 octaves. The idea behind this function is given below.

int getOctave(int frequency) //taking frequency as a parameter

{

int val = AnalogRead(A3) //reading data from potentiometer

//then according to the value of potentiometer returns new frequency

if (val <= 146)

return frequency/8;

//4 more conditions for 4 ranges

else if (val <= 878)

return frequency \* 4;

else

return frequency \* 8;

}

Here we showed only three ranges. Like those our function provides 7 different ranges of frequency. Since a higher octave is double the frequency and a lower octave is half the frequency, multiplying or dividing a note’s frequency by a factor of two keeps the same note but changes the octave it is in.

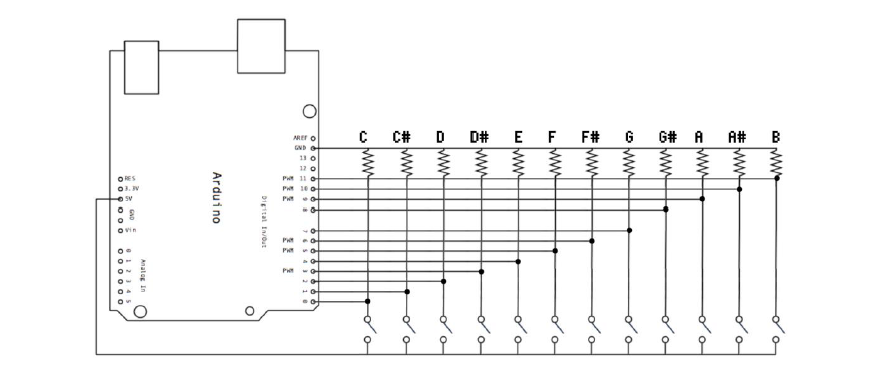


Figure: Sketch of our musical keyboard

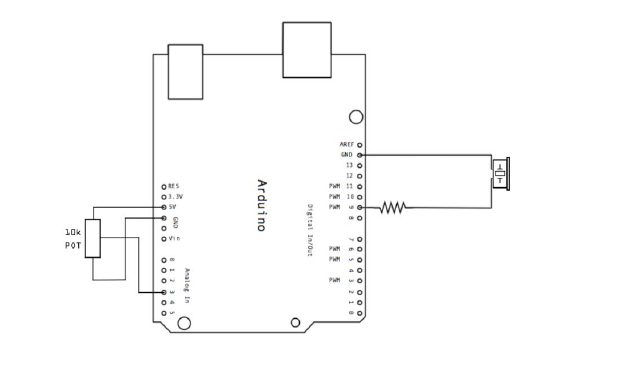


Figure: Sketch of our potentiometer and piezo

We checked 13 states for 13 buttons. If one button state is changed, then it performs its assigned function. Next, if the state is changed, then it is determined whether the new state is HIGH or LOW. If the state is HIGH then our program will call one function of that button. And that function will call another function where we provide one specific frequency for that button and that frequency stands for one specific note of piano. If the state is LOW then our program will call another function that where our program understand that the button is not pressed any more. As a result, that function performs the noTone() command.

val1 = digitalRead(buttonPin1); // read input value of one button and store it in val1

if (val1 != buttonState1) { // the button state has changed!

if (val1 == HIGH) // check the button’s state

{

Button1(); //this function calls another function with one specific frequency

}

if (val1 == LOW) // check the button’s is state

{

//releasing that button

minusButton(); // this function calls another function, so no more tone

}

buttonState1 = val1; //state changed, so store that state

}

Here we described what will happen if one button is pressed, like that one we have 12 more conditions for rest of other buttons.

Different notes have different frequency. Some graphs of the frequency are given below.

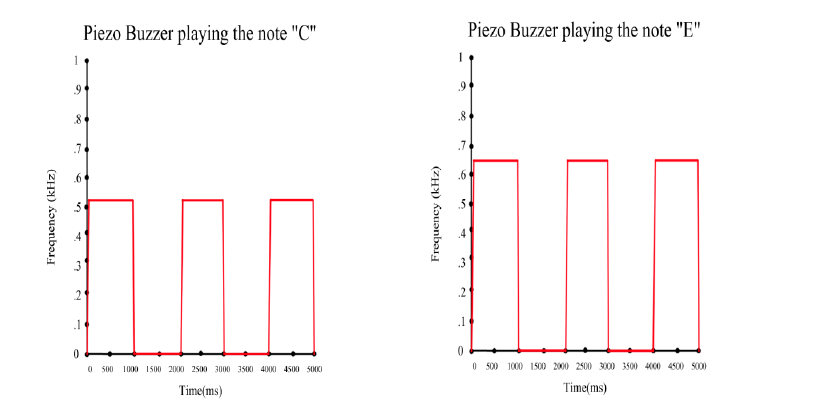


Figure: Frequency of ‘C’ and ‘E’ note

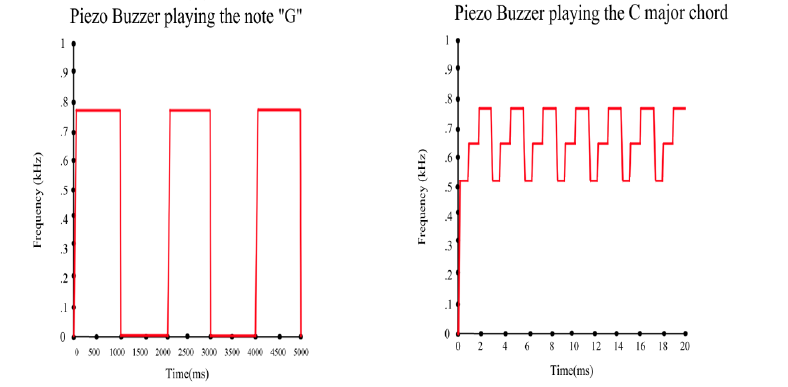


Figure: Frequency of ‘G’ note and C major chord

The original plan for this project was to have 4 piezo buzzers playing notes together, so a chord of up to 4 sounds could be played. However, this was found to be impossible. The tone() function will only work with 1 piezo buzzer, and if there is ever a conflict, it will only play sounds over the first buzzer it detects, which means that only one piezo buzzer can play a sound at any given time. Attempts were made to circumvent this, such as using the active buzzer, writing directly to the buzzer with analogWrite, and writing to several different notes in rapid succession to simulate multiple buzzers via a single buzzer. All of these attempts led to results which the group agreed were unacceptable. The active buzzer did not produce multiple tones, it only produces the same tone with different amplitudes. Using analogWrite was not successful, as the piezo was not able to discern different tones from it. While the piezo buzzer did create sound when it was being powered through analogWrite, neither its frequency nor amplitude changed with different inputs from analogWrite. The most successful attempt at remedying the situation was to play multiple different tones using the tone() function and a very small delay between tones to simulate different piezo buzzers playing different notes in harmony. This experiment was successful in its attempts to create a new tone by combining several tones. However, the tone was muddied and not pleasant to listen to in comparison to just playing a single note. This is likely because piezo buzzers play sounds in a square wave, and the distortions to the frequency caused by hopping between frequencies so quickly created a “jagged”, ugly sound. As such, the group has decided that moving on, we will use a single piezo buzzer playing a single note at a time to minimize issues related to Arduino’s lack of support for multiple buzzers, and the square-wave distortions associated with switching between frequencies rapidly.

Some possible improvements are to find some functionality to have multiple piezo buzzers compatible with the keyboard. There are some tutorials online to make several buzzers work on Arduino, or to make one buzzer make multiple tones, but these tutorials were deemed too complicated to implement into our project in a reasonable amount of time, but this could be a change if given an ample amount of time. A simpler option was, since we have 3 partners, and 3 Arduinos, we can have 1 piezo per Arduino, for 3 piezos total. Similarly, this would allow for more varied demos to be played, instead of simple chiptune music which a piezo is limited to. The original plan was to play Beethoven’s 5th Symphony, however, this was less possible with a single piezo. With multiple piezos, many more songs are possible. If given more time, there could be more demo songs added to the rotation, so that pressing the 13th button would cycle between songs. Another option would be to use something like the Raspberry Pi or some other microcontroller which allows for the employment of multiple piezo speakers. Another slight downside of our build was that there were a lot of wires all over the place, obstructing the buttons. It might have been better to use a larger breadboard, if available, to hide away the circuitry better, so that you aren’t reaching over and around wires and resistors to press the buttons. If we really had a lot of time and resources, it might be better to encase the project in a box, and to have piano keys depress the buttons so that it really looks like a piano. In that case, a lot of features could be abstracted away, such as the button that plays demos being separate from the rest of the keys. While we attempted to clarify it by having natural keys low, sharp/flat keys high, and have the demo button be in the middle, this is probably not obvious, so that could be a source of improvement. One other thing that might need improvement, in some way, is that octave is controlled by a potentiometer, rather than by having the keyboard be much longer. This allowed us to recreate the full sound range of a keyboard on a small breadboard. Our Arduinos do not have room for many buttons, in fact, having the 13 buttons was too much for one Arduino, which required the employment of a second. There is no reasonable amount of Arduinos we could have used in order to have a full 88 keys like on a real piano, as this would require at least 8 Arduinos, possibly 9 because using all 11 slots on an Arduino would mean that there is no room for any other peripherals, such as a piezo buzzer. One final feature that was originally proposed but later removed due to high complexity and low performance value was an LCD screen for the keyboard, which would display what note was being played. We decided to remove it since all the necessary additional pins required would force us to use a third Arduino, LCD screens are annoying to set up and program code for, and the addition of displaying on a screen what note was being played is a relatively small addition to the Arduino project, since you should know what note is being pressed anyway, and real keyboards generally do not display this information to the user since it is assumed to be knowledge the user already has.

Something we might want to do differently if we started the project again would be to possibly choose a different topic that was not dependent on the intricacies and minutia of piezo buzzers. They are very cool and interesting to work with, but the complexities in dealing with multiple piezo buzzers derailed the project immediately and most of the effort spent on the project was reconciling those differences. With this knowledge, it might have been simpler picking a project that does not rely on piezo buzzers as the cornerstone aspect of this project. Some very interesting projects presented in the videos were things like the Pong and Snake games using the LED board. There are many games that can be played on an LED board that we could have made, such as Tetris. Such a Tetris game might have consisted of an LED board and a single Arduino, since there are much fewer inputs to monitor. There would also be 5 buttons, representing WASD + space, or the 5 classic controls on the original Tetris (numpad 7, 8, 9, 4, 5), which would signal left, right, rotate, faster, and drop. Of course, there would also be a piezo playing Korobeiniki on a loop.

A lot was learned though this final project. For a start, Arduinos are a very new and unique experience, and we have not used microcontrollers before this class. This project helped us gain an understanding of computers, and how physical components and connections relate to code we write, and how code and computer interact to achieve results. We soldered components, and learned about circuits, with power flowing from source to ground. If at any time in a circuit there was a break, the entire circuit would not work and we had to find the fault. Breadboards made visualizing the circuits a lot easier. This was definitely a new experience, since in the past, we have only ever programmed on a computer, either to manipulate data, or to create some small console-based game, but in this class we actually used hardware and combined it with software and were able to make more exciting things happen with i/o devices.