Rick Stefanik, Luke Weller, Jack Shea
Professor Fuja, Microprocessors and Multimedia
Arduino Project Report
24 March, 2017

Executive Summary

The project functions as a music sequencer. The circuit and corresponding program are able to take an input of a sequence of notes from the user and play those notes back to the user through the piezo. The user is able to change the pitch of each note, the tempo of the sequence, and the length of the sequence. Through this manipulation, a large number of different sequences can be played. In addition, an LCD allows the user to view the pitch of the note currently selected, the tempo of the sequence, and the length of the sequence.

Background

Given a sequence of notes, the device is able to store and replay that sequence of notes. To some extent, the device functions as an instrument to create music. A recording of the piezo could be used to contribute to a musical piece. Many musical artists today strive to sound different than the mainstream, and this device could easily be used by a musician to add a unique sound to a song.

Group 9 would also consider entertainment to be an important goal of the device. While the piezo used in the circuit can generate a variety of notes, the general sound quality of the piezo is less than optimal for inclusion in a classical music piece, but many modern and independent musicians love to get their hands on anything that can make them sound unique. Using the device to create a wide variety of musical sequences can also be very entertaining for hobbyist or amateur musician.

The product can be considered a variation of a musical sequencer. Professionally designed music sequencers function in much the same way as our device. They often have an interface of buttons that allow a user to input notes into a sequence, and the ability to repeat this sequence of notes at a specific tempo. These sequencers are generally built into synthesizers, which usually have a full set of piano keys, as opposed to the single button present on the device.

Group 9's project is not based off any existing Arduino project. The project is loosely based off a professionally manufactured music sequencer, branded the Teenage Engineering Pocket Operator. The main difference between Group 9's device and the Teenage Engineering Pocket Operator is the removal of the feature that allows a note to be input into the sequence while the sequence is currently playing. This feature would be hard to implement since the Arduino Uno board has only two interrupt pins. The Pocket Operator also has over twenty buttons alone, and the Arduino Uno board simply could not have that many buttons, along with an LCD and potentiometers, so Group 9 had to create the device with these limitations in mind.

Functional Description

The device functions as a music sequencer. The central operation of the device is to store a sequence of notes, and play back those notes through a piezo. The user is able to control the pitch and location of each individual note, the length of the sequence of notes, and the tempo at which the piezo plays through the sequence of notes. These variables allow for a large variety in potential sequences. The user input of the musical sequence is controlled by a number of buttons and potentiometers within the circuit.

The device also includes an LCD screen that acts as a user interface for the device. In the sequence writing state of the function, the LCD screen displays the current note that will be written if the user is to press the write button, the current position in the sequence, the note that currently occupies this position, along with the current tempo and length of the sequence. In the change sequence length and change tempo states of the program, the LCD screen will display the length and tempo, respectively.

In addition, a LED within the circuit is lit with a analog brightness. This brightness is mapped to the current tempo. At a high tempo, LED will be very bright, and at a low tempo the LED will be very dim.

Product Operation

* All location descriptions are oriented with the LCD characters upright to the user

Playing a Note

In order to build a sequence, it is helpful to know the pitch of the note currently selected. The pitch of the note can be varied using the rightmost potentiometer on the right breadboard of the circuit. The name of that note can be observed in the leftmost portion of the first line of the LCD screen. To play the currently selected note, the user can press the second button from the left on the left breadboard. While the user holds this button, the note is played.

Generating a Sequence

A sequence is generated by individually inputting each note to a specific position. The user sets the pitch of the note by controlling the rightmost of the two potentiometers on the right breadboard. In order to help the user, the value of this pitch is displayed on the LCD screen. When the user finds the correct note, the leftmost potentiometer is used on the right breadboard in order to set the location in the sequence that the note plays during. This location is also displayed on the LCD screen. The location is displayed in the first row of LCD, after the current pitch. When the desired pitch and location of the note are selected, the write button (the single button on the right breadboard) is pressed to input the note into the sequence at that location. Once the note is written to the location, that note is displayed at the rightmost position on the first row of LCD. This process can be repeated to fill each of the possible places in the sequence with notes.

Playing the Sequence

After a sequence of notes has been generated, the sequence can be played through the piezo by pressing the play button. The play button is the rightmost button on the left breadboard in the circuit.

Adding a Rest

When changing the pitch using the pitch potentiometer, one “pitch” that can be displayed is “REST”. If the write button is hit when “REST” is displayed, a rest will be written into the pattern at that location, so that when the pattern is played, no note is heard at that location.

Clearing the Sequence

When changing the pitch using the pitch potentiometer, one “pitch” that can be displayed is “CLR”. If “CLR” is displayed, and the write button is hit, then all notes in the sequence are set to “REST”. This is useful when the user wants to start writing an entirely new pattern.

Changing the Tempo of the Sequence

The tempo at which the sequence plays can be altered by the user. The user must press the change tempo button (the third button from the left on the left breadboard) in order to change the tempo. The sequence tempo can then be changed by moving the leftmost potentiometer on the right breadboard. The current tempo value will appear on the LCD screen during this mode, and will also be indicated by the brightness of the circuit LED. When the user finds the optimal tempo, this tempo can be set by pressing the play button (the rightmost button on the left breadboard). Pressing the play button will exit the change tempo mode and play the current sequence back to the user at the new tempo.

Changing the Length of the Sequence

The length of the sequence can be altered by the user. The user must press the change length button (the leftmost button on the left breadboard) in order to enter the change length mode. The length of the sequence can now be changed by moving the leftmost potentiometer on the right breadboard. The length of the sequence is displayed on the LCD screen while in this mode. When the user finds the optimal sequence length, that length can be set by pressing the play button (the rightmost button on the left breadboard). Pressing the play button will exit the change length mode and play the current sequence back to the user.

Changing the Brightness of the LCD Screen

The brightness of the LCD screen is controlled by the single potentiometer on the left breadboard. Moving this potentiometer changes the screen brightness.

Hardware

The sensors used in this project consist of 3 potentiometers (analog inputs) and 5 buttons (digital inputs). Each potentiometer serves a different purpose in the functioning of the sequencer. The far left potentiometer controls the backlight of the LCD screen, while the two on the right breadboard assist with the user's interaction with the program's interface. The right potentiometer on the right breadboard changes the note to write to the music sequence, while the potentiometer directly to its left is used to change the position to which the note will be written. This potentiometer is also used to adjust the sequence length and music tempo in those respective modes. The first button from the left puts the user in length editing mode, where they can change the length of the sequence they wish to write. The second button allows the user to hear the note that they are currently scrolling through. The third button puts the user in tempo editing mode, where they can change how fast the player will run through the notes. The fourth button is "play", and will cause the program to go through the sequence of notes that the user has programmed at the currently specified tempo. The fifth button writes the current note that the user is scrolling through to the current position.

There are three actuators used in the functioning of the project: a piezo, an LCD, and an LED receiving a PWM analog signal. The piezo is the primary actuator of the sequencer, as it will both play the whole sequence of notes specified by the user and the current note when the user wishes to hear the note that he or she is currently writing to the music sequence. The LCD makes the sequencer more usable, as it displays valuable information to the user in all modes. In the writing mode, it displays the note that the user may write, the position to which the user may write it, and the length and tempo of the current sequence. In the playing mode, the LCD will both display the current note and current note position as it cycles through the music sequence. The LED is an indicator of the tempo that will be used by the sequencer as it plays the music sequence. The brighter the LCD, the faster the tempo. The tempo (and therefore the LCD brightness) can be changed in the tempo editing mode, which can be accessed by pressing the third button from the right.

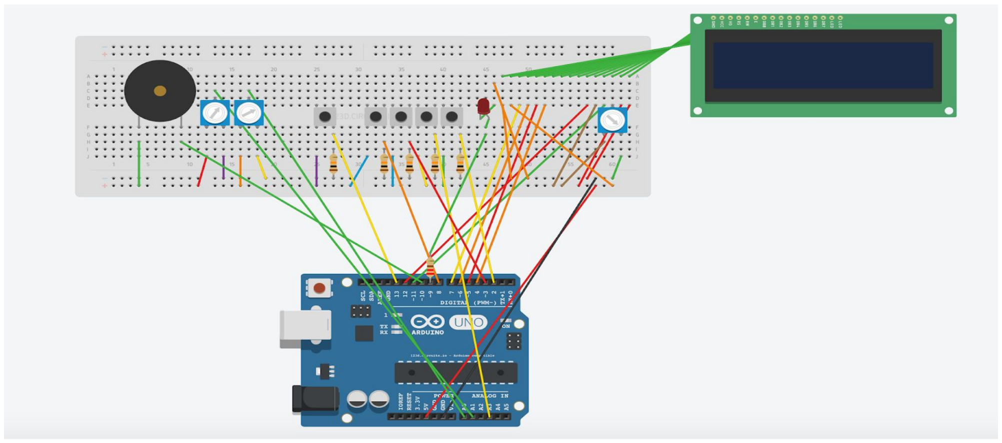


Figure 1: This is a diagram of the sequencer circuitry and how it connects with the arduino board.

Most of the hardware choices for the project were quite straightforward and were very obviously the best choices for their functions. The piezo and LCD were the best tools for implementing this sequencer because they were both very accessible and effective at doing their respective jobs of playing the music sequences and conveying information to the user. With regard to actually writing the music sequences, Group 9 had to choose between using potentiometers or buttons for choosing the notes and their positions. Group 9 ultimately decided that potentiometers would be the best tools for allowing the user to efficiently scroll through notes and positions, as buttons would require a great deal of tedious pressing to reach desired inputs. It was decided that buttons would be better suited for switching between the various modes offered by the sequencer, as these actions have a more digital nature. Pulldown resistors were also used in the implementation of all buttons to prevent phantom voltages being applied to their corresponding input pins.

Software

An overview of the software is given below, but the code in the appendix is very heavily commented if more detail is desired.

Initialization

In the first few lines, the program initializes the LCD and assigns appropriate pins so that the display is able to interact with the arduino board. Next, the program initializes the digital pin variables to be used in the circuit. The following digital pin variables are created: lengthButton, tempoButton, playButton, tempoLED, soundPiezo, and writeButton. A similar process is repeated for the analog pins, which include the variables adjustPot, pitchPot, and soundButton. Next, the program defines two arrays. The first, named Notes, is the character array of notes that the sequencer is able to play. These characters will be used to display the current note on the LCD display. The second array, called Frequencies, is the same length as the Notes array, and contains the frequencies that align with the given notes. The program goes on to define the minimum and maximum values for the length of the sequence (called PatternLength) and the delay value (called DelayValue) . The program initializes the testNoteLength value to 250. This variable sets the length of the tone that plays when the user presses the play note button to 250 ms. The program also sets the initial values for variables that appear on the LCD screen. Finally, two important volatile variables are initialized delayTime and pattern.

void setup()

First the program calls the function setToZeros with the argument of 0. This function will be explained in more detail later, but the primary purpose of the function is to clear the array memory. Next the program writes a starting song into the pattern array. This allows a sequence of notes to be preloaded when the program is compiled. Next, the program assigns the pins that were initialized in the initialization step- e.g., lengthButton, to either input or output. The program then contains two attach interrupt functions. The first allows the program to enter the changeTempo function when the tempoButton is pushed. The second allows the program to enter the changeLength function when the change length button is pushed. Next, the program calls the function displayCurrentValues. This function will be explained in more detail later, but it essentially instructs the LCD of what values to display. The final line sets the initial value of the brightness for the LED within the circuit. The brightness is set by mapping the ratio of the delayTime value within the range of minDelayValue and maxDelayValue onto the range of 0 to 255.

void loop()

The loop function begins by defining three new variables, referencePitchValue, referenceLocationValue, referenceLocationFrequencyValue. These variables are set equal to the initial values of each as defined in the initialization section. These variables are used to determine if any of the variables change each time through the loop. This is done in conjunction with the refreshing of the LCD. Next the pitch potentiometer is read, and that value is appropriately mapped to correspond to an index of the Frequency. The program then checks if the sound button is hit, and if it is, it plays the current note, which is the frequency mapped from the pitch potentiometer. Next the adjust potentiometer is read, and that value is appropriately mapped to correspond to a number between 0 and the maximum pattern length minus one. Then the current location is set to what that value is. Then the program checks if the write button is hit. If it is, it writes the current note to the current location. If the current note is “REST”, the pattern will not play a tone at that moment. If the current note is “CLR”, all elements in the pattern array are set to zero, so every note in the pattern is now a “REST”. In this way, “CLR” clears the pattern. The program then checks if the play button is hit. If it is, the pattern is played using the playPattern() button. The program then checks if any of the original references values declared at the start of the loop have changed. If and only if one of these values has changed, the LCD will update the screen with the new values. The program also displays the tempo, by mapping the delay time, and the current pattern length on the second line of the LCD. The reference values are then updated, so that the next time through the loop, the LCD will be refreshed if any of the values have changed. Finally, the program delays for fifty milliseconds, to alleviate some stress on the Arduino.

void playPattern()

The playPattern function runs through a for loop to play the sequence of notes. The for loop iterates through every integer from 0 to the pattern length. For each value of i, the LCD is first cleared. Then, if the value of pattern array is not a rest value, the piezo plays the note stored in that sequence using the tone function. Next, the program prints on the LCD the current position in the sequence, and the current note played being played. The program then pauses for a length of delayTime. In the last line of the function, displayCurrentValues is called.

void setToZeros(int x)

This function takes in a parameter x. Using a for loop, the function iterates through all the integer values from x to to the size of the current pattern array and sets all the values equal to 0. Since x = 0 in the only time in which this function is called, the function is essentially used to clear the current pattern array.

void changeTempo()

This function takes in no parameters, but sends the program into the tempo editing state. After this function is called, the user has the option to use a potentiometer to changes the tempo at which the music sequence will be played. The current tempo will also be indicated by the LED receiving a PWM signal, as its brightness will be directly correlated with the tempo is currently selected.

void changeLength() -

This function sends the program into the length changing state, in which the user can use the potentiometer to adjust how long the music sequence will be.

void displayCurrentValues() -

This function is used to display the current note, position, tempo, and length values to the LCD to give the user a better understanding of the music sequence they have written.

int findFreqIndex(int x) -

This function, given a frequency, will return the location of that frequency in the Frequency array, which can then be used to identify the corresponding note. This function is useful in getting the name of a note given its frequency, since the Frequencies and Notes arrays are parallel. For instance findFreqIndex(55) returns 2, since 55 is in index 2 of the Frequency array. The name of this note can then be found by looking up this index in the Notes array (Notes[2] = "A1").

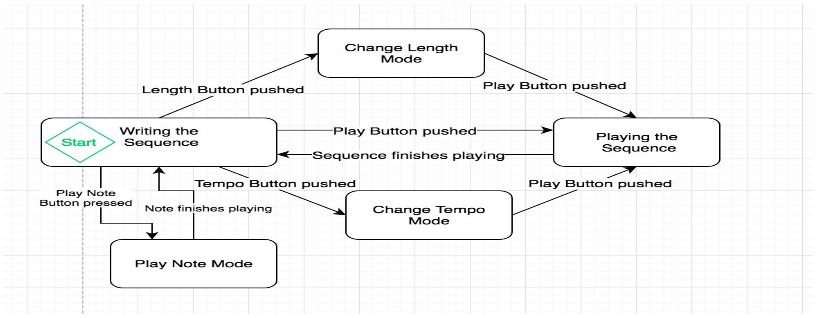


Figure 2: A flowchart of the program shows how the program functions when run.

Design Post-Mortem

The main difference between the final "product" and the proposed product was the elimination for the "tap for tempo" feature. Originally, Group 9 planned to use a piezo as input to set the tempo. The user would tap on the piezo at a certain speed, and this speed would determine the tempo. As it turns out, implementing this feature was much harder than expected, and Mr. Hunte said it may not be possible to do with our current knowledge about the Arduino and C, so rather than use a piezo to adjust the tempo, a potentiometer used instead. The potentiometer still allows the user to change the tempo, and it even gives a finer control over the tempo than the piezo would have.

The feature that changes the length of the pattern was the hardest to implement for two reasons: one regarding the code for the feature, the other regarding the hardware. The original idea for changing the pattern length was to resize the pattern array each time the user changed the length. Because of the way in which C handles arrays, actually resizing the array proved to be a poor approach. Instead a variable called pattern length was created. When the play button is hit, the pattern is played up to the index of pattern length. When the user changes the length of the pattern, they change the value of pattern length, which in turn actually changes how many notes are played. So while the original idea for the code to implement this feature did not work, Group 9 was able to think about the problem in a different way and successfully implement the feature. This feature is triggered with an interrupt, but when Group 9 initially tested the feature, the interrupt never triggered. After scouring the code, no issue regarding the interrupt could be found. Eventually Group 9 found that issue was a faulty button. When a different button was put in the same place, the interrupt worked exactly as desired. So Group 9 learned a valuable lesson: to check both the hardware and software when debugging.

If this project were to be completed again, Group 9 would likely utilize multiple piezos and allow the user to program full chords into the sequencer to give the music a more full sound. Though it would likely take much more complicated wiring and many more circuit components, Group 9 is confident that, given ample time and resources, a much more complex and interesting sequencer could be made.

Appendix

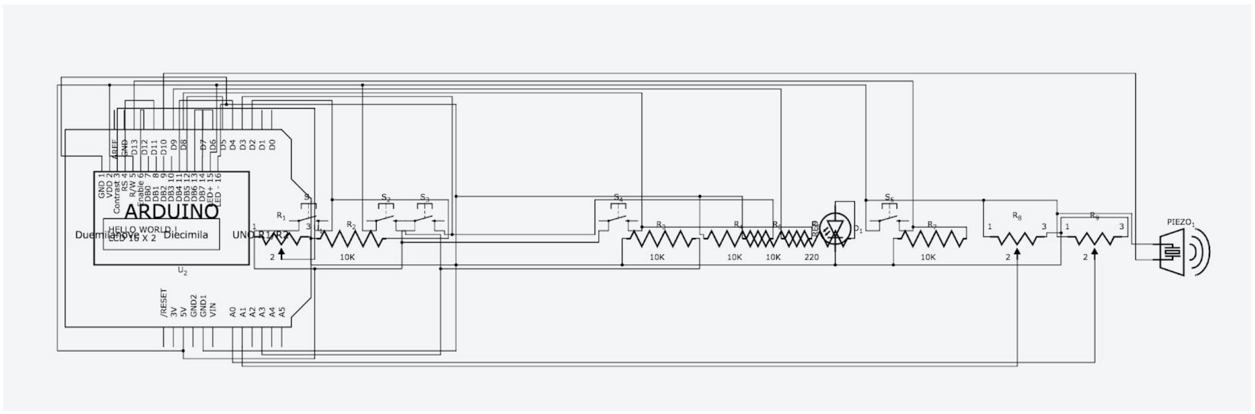
```
// Set up LCD.
#include <LiquidCrystal.h>
LiquidCrystal lcd(11, 12, 4, 5, 6, 7);

// Digital Pins
const int lengthButton = 2;
const int tempoButton = 3;
const int playButton = 8;
const int tempoLED = 9;
const int soundPiezo = 10;
const int writeButton = 13;

// Analog Pins
const int adjustPot = 0;
const int pitchPot = 1;

// soundButton could be a Digital Input, but
// it is an Analog Pin since all other
// Digital Pins are used.
const int soundButton = 3;


// Array of names of Notes (to be displayed on LCD
// CLR and REST are special commands and hence have special "frequencies".
const char* Notes[] =  {"CLR", "REST", "A1", "A#1", "B1", "C2", "C#2", "D2", "D#2", "E2", "F2", "F#2", "G2", "G#2", "A2", "A#2", "B2",
"C3", "C#3", "D3", "D#3", "E3", "F3", "F#3", "G3", "G#3", "A3", "A#3", "B3", "C4", "C#4", "D4", "D#4", "E4", "F4", "F#4", "G4", "G#4",
"A4", "A#4", "B4", "C5", "C#5", "D5", "D#5", "E5", "F5", "F#5", "G5", "G#5", "A5"};
// Parallel Array of Frequencies
const int Frequencies[] = {-1,    0,   55,   58,  62,  65,   69,  73,   78,  82,  87,   92,  98,  104,  110,  117,  123,  131,  139,
147,  156,  165,  175,  185,  196,  208,  220,  233,  247,  262,  277,  294,  311,  330,  349,  370,  392,  415,  440,  466,  494,
523,  554,  587,  622,  659,  698,  740,  784,  831,  880};

const int minPatternLength = 4;
const int maxPatternLength = 64;
volatile int patternLength = 20;

// This array will hold the values of the Frequencies for beat in
// the pattern.  In the setup function, each of its values will be
// set to 0, to ensure C every note is initially a REST.
volatile int pattern[maxPatternLength];

// This variable will change based on the desired tempo.
// The lower this value, the faster the tempo.
volatile int delayTime = 250;

// When the user plays a note simply to hear the tone,
// (not actually write it) the note plays for this many
// many milliseconds.
const int testNoteLength = 250;


// These are simple initial values for certain variables,
// used so the LCD has something to display immediatley on startup.
const int initialPitchValue = 0;
const int initialLocationValue = 0;
const int initialLocationFrequencyValue = 0;

// This two values essentially set the minimum
// and maximum possible tempos for the pattern.
const int minDelayValue = 150;
const int maxDelayValue = 400;

void setup() {

// Ensures that all values of pattern are initally set to zero.
```

```
  setToZeros(0);

//Startup Song
  pattern[0] = 330;
  pattern[1] = 440;
  pattern[2] = 440;
  pattern[3] = 392;
  pattern[4] = 330;
  pattern[5] = 440;
  pattern[6] = 330;
  pattern[7] = 330;
  pattern[8] = 0;
  pattern[9] = 262;
  pattern[10] = 294;
  pattern[11] = 330;
  pattern[12] = 0;
  pattern[13] = 659;
  pattern[14] = 587;
  pattern[15] = 523;
  pattern[16] = 587;
  pattern[17] = 0;
  pattern[18] = 523;
  pattern[19] = 523;

  lcd.begin(16, 2);

  pinMode(playButton, INPUT);
  pinMode(tempoLED, OUTPUT);
  pinMode(lengthButton, INPUT);
  pinMode(tempoButton, INPUT);
  pinMode(soundPiezo, OUTPUT);
  pinMode(writeButton, INPUT);


  // The changeTempo and changeLength features are triggered
  // by using iterrupts.
  attachInterrupt(tempoButton-2, changeTempo, RISING);
  attachInterrupt(lengthButton-2, changeLength, RISING);

  displayCurrentValues();


  // Sets the tempoLED to the appropriate brightness based on tempo (delayTime).
  // The shorter the delay time, the brighter the LED.
  analogWrite(tempoLED, map(delayTime, minDelayValue, maxDelayValue, 255, 0));
}

void loop() {

  // These reference variables keep track of what the values of these variables
  // are at the beginning of the loop.  Later in the loop, the program checks if
  // any of these values have changed from what they originally were.  If they have
  // changed, then the LCD is refreshed.  They are set equal to the dummy initial values
  // to avoid issues on the first time through the loop.  These variables are declared as
  // static, so this variable declaration only occurs once, and not every time through the loop.
  static int referencePitchValue = initialPitchValue;
  static int referenceLocationValue = initialLocationValue;
  static int referenceLocationFrequencyValue = initialLocationFrequencyValue;


  // pitchValue corresponds to an index in the Frequencies array.
  int pitchVoltage = analogRead(pitchPot);
  int pitchValue = map(pitchVoltage, 0, 1023, 0, ((sizeof(Frequencies)) / (sizeof(int))));

  // Some issues with flickering occured when pitchVoltage was mapped to its maximum possible value,
  // so the maximum number it could be mapped to is actually one higher than desired.  If this value
  // is obtained, the program immediately decreases it by one.  In this way, the right number of values
  // are effectively mapped and the flickering is avoided.
  if (pitchValue == (sizeof(Frequencies)) / (sizeof(int)))
  {
     pitchValue = ((sizeof(Frequencies)) / (sizeof(int)) - 1);
  }



  // If the soundButton is hit, the piezo will play the current note.
  // Since soundButton had to be used as an analog input (since all other
  // digital inputs were taken), the if statement uses analogRead.  If soundButton
  // were in a digital input, the statement would read: if (digitalRead(soundButton) == HIGH)

  if (analogRead(soundButton) > (1023/2))
  {

    // When pitchValue is equal to 0, it corresponds to CLR.
    // When pitchValue is equal to 1, it corresponds to REST.
    // Since these values do not have actual sounds corresponding to them,
    // no sounds should be played when the soundButton is hit.  Hence the
    // following if statement.
    if (pitchValue > 1)
    {
    tone(soundPiezo, Frequencies[pitchValue], testNoteLength);
    }
  }

  // Similar to pitchVoltage, to stop the screen from flickering, the map function
  // and following if statement are written in this way.
  int locationVoltage = analogRead(adjustPot);
  int locationValue = map(locationVoltage, 0, 1023, 0, patternLength);

  if (locationValue == patternLength)
  {
     locationValue = patternLength - 1;
  }

  // If the user hits the write button, the current note will be placed at the current location
  if (digitalRead(writeButton) == HIGH)
  {

    // If user hits the write button when CLR is on the screen (when Frequencies[pitchValue] equals -1),
    // then the entire pattern will be cleared.  Useful when wanting to start a new pattern.
    if (Frequencies[pitchValue] == -1)
    {
      setToZeros(0);
    }

    // Otherwise, the current note is simply written into the current location.
    // If a note is already in that location, it is overwritten with the
    // new note.
    else
    {
      pattern[locationValue] = Frequencies[pitchValue];
    }

  }

  // Hitting the play button plays the current pattern, with the
  // ocorrect tempo and length.
  if (digitalRead(playButton) == HIGH)
  {
    playPattern();
  }
```

```
  // The following if statement makes it so that the LCD only
  // refreshes if the values being displayed have been changed.
  if ((pitchValue != referencePitchValue) || (locationValue != referenceLocationValue) || (pattern[locationValue] !=
referenceLocationFrequencyValue))
  {


  //The LCD first displays the current Note that will
  // be heard if the soundButton is hit.  Then " - " is
  // displayed.  Then the current location in the pattern is
  // displayed.  Then " - " is displayed.  Then the current note
  // stored in that location is displayed.  On the next line of the
  // LCD, "T: " is displayed.  Then the current tempo is displayed.
  // Then " L: " is displayed.  Then the current pattern length is displayed.


    lcd.clear();
    lcd.print(Notes[pitchValue]);
    lcd.print(" - ");
    lcd.print(locationValue + 1);
    lcd.print(" - ");
    lcd.print(Notes[findFreqIndex(pattern[locationValue])]);

    //Move cursor to the second line of the LCD.
    lcd.setCursor(0, 1);
    lcd.print("T: ");
    lcd.print(map(delayTime, minDelayValue, maxDelayValue, 100, 1));
    lcd.print(" L: ");
    lcd.print(patternLength);
    //Move cursor back to the first line of the LCD.
    lcd.setCursor(0, 0);


    // Reference values are now changed to what they currently are.
    // This way, the LCD will only update when one of the values
    // is changed.
    referencePitchValue = pitchValue;
    referenceLocationValue = locationValue;
    referenceLocationFrequencyValue = pattern[locationValue];
  }

  // This short delay is included to alleviate some
  // stress on the Arduino.
  delay(50);
}


void playPattern(){

  // Traverses through the pattern array and plays the corresponding notes.
  // If there is a "0" in the array, no note is played, but the proper amount
  // of time is waited.

  for (int i = 0; i < patternLength; i++)
  {
    lcd.clear();
    if (pattern[i] != 0)
    {
    tone(soundPiezo, pattern[i], delayTime);
    }

    // These three lines display th current location and note
    // being played in the pattern.  i + 1 is printed because
    // this value would make more sense to the user.  For instance,
    // for a four note pattern, the function will play the frequencies
    // in indexes 0, 1, 2, and 3 of the pattern array, but to a user/musician,
    // it would make more sense to see 1, 2, 3, 4.
    lcd.print(i + 1);
    lcd.print(" - ");
    lcd.print(Notes[findFreqIndex(pattern[i])]);

    // This delay call is how the tempo functionality works.  The
    // changeTempo function changes the value of delayTime, and in turn
    // changes how fast the notes will play.
    delay(delayTime);
  }

  displayCurrentValues();
}



void changeTempo(){
  lcd.clear();
  int tempoValue;

  // This referenceTempo variable is again used so that the
  // LCD only updates when a value is changed.
  int referenceTempo = 0;

  // The user can scroll through all possible tempo values.
  // The playButton is used to exit the changeTempo mode.
  while (digitalRead(playButton) == LOW)
  {
    int tempoVoltage = analogRead(adjustPot);
    // Again, map and if statement written to avoid observed screen
    // flickering issue.
    tempoValue = map(tempoVoltage, 0, 1023, maxDelayValue, minDelayValue - 1);
    if (tempoValue == minDelayValue - 1)
    {
      tempoValue = minDelayValue;
    }

    // The LED should be at its brightest when tempoValue is at its lowest.
    int LEDBrightness = map(tempoValue, minDelayValue, maxDelayValue, 255, 0);
    analogWrite(tempoLED, LEDBrightness);

    // Screen is only updated if tempoValue is changed.
    if (tempoValue != referenceTempo)
    {

      // When displaying to tempo for the user, 1 will be the minimum tempo and
      // 100 will be the maximum tempo.  This  inutivitvely makes more sense to the user compared
      // to displaying to them a delay time.
      int displayTempoValue = map(tempoValue, minDelayValue, maxDelayValue, 100, 1);
      lcd.clear();
      lcd.print("Tempo: ");
      lcd.print(displayTempoValue);
      referenceTempo = tempoValue;

      // playPattern uses delayTime, so changing delayTime will affect
      // how the pattern is played.  The result is the desired effect of
      // changing the tempo.
      delayTime = tempoValue;
    }
  }

  // When leaving the function, occasionally only a blank
  // screen would be displayed, so these lines were included.
  lcd.clear();
  displayCurrentValues();
```

```
}


void changeLength(){
  lcd.clear();
  int lengthVoltage;
  int lengthValue;
  int referenceLengthValue = 0;

  // The user can scroll through all possible pattern length values.
  // The playButton is used to exit the changeLength mode.
  while (digitalRead(playButton) == LOW)
  {
    lengthVoltage = analogRead(adjustPot);
    lengthValue = map(lengthVoltage, 0, 1023, minPatternLength, maxPatternLength + 1);
    if (lengthValue == maxPatternLength + 1)
    {
      lengthValue = maxPatternLength;
    }
    if (lengthValue != referenceLengthValue)
    {
      lcd.clear();
      lcd.print("Length: ");
      lcd.print(lengthValue);
      referenceLengthValue = lengthValue;
    }
  }

  // When the play button is hit, the pattern is played up
  // to the index patternLength, so changing this index changes
  // the length of the array.
  patternLength = lengthValue;

  // When leaving the function, occasionally only a blank
  // screen would be displayed, so these lines were included.
  lcd.clear();
  displayCurrentValues();
}


// Sets all indexes of an array icluding and
// after x equal to 0.
void setToZeros(int x)
{
  for (int i = x; i < (((sizeof(pattern)) / (sizeof(int)))); i++)
  {
    pattern[i] = 0;
  }
}





// When leaving the changeTempo and changeLength functions,
// the screen would occasionally stay blank until a potentiometer
// was adjusted.  This function ensures that the this "blank screen"
// does not occur.

void displayCurrentValues()
{
  int pitchVoltage = analogRead(pitchPot);
  int pitchValue = map(pitchVoltage, 0, 1023, 0, ((sizeof(Frequencies)) / (sizeof(int))));

  if (pitchValue == (sizeof(Frequencies)) / (sizeof(int)))
  {
      pitchValue = ((sizeof(Frequencies)) / (sizeof(int)) - 1);
  }


  int locationVoltage = analogRead(adjustPot);
  int locationValue = map(locationVoltage, 0, 1023, 0, patternLength);

  if (locationValue == patternLength)
  {
      locationValue = patternLength - 1;
  }

  //The LCD first displays the current Note that will
  // be heard if the soundButton is hit.  Then " - " is
  // displayed.  Then the current location in the pattern is
  // displayed.  Then " - " is displayed.  Then the current note
  // stored in that location is displayed.  On the next line of the
  // LCD, "T: " is displayed.  Then the current tempo is displayed.
  // Then " L: " is displayed.  Then the current pattern length is displayed.

   lcd.clear();
   lcd.print(Notes[pitchValue]);
   lcd.print(" - ");
   lcd.print(locationValue + 1);
   lcd.print(" - ");
   lcd.print(Notes[findFreqIndex(pattern[locationValue])]);
   //Move cursor to the second line of the LCD.
    lcd.setCursor(0, 1);
    lcd.print("T: ");
    lcd.print(map(delayTime, minDelayValue, maxDelayValue, 100, 1));
    lcd.print(" L: ");
    lcd.print(patternLength);
    //Move cursor back to the first line of the LCD.
    lcd.setCursor(0, 0);

}



// This function returns the index of where a given
// frequency is in the Frequency array.  This function
// is useful in getting the name of a note given its frequency,
// since the Frequencies and Notes arrrays are parallel.
int findFreqIndex(int x)
{
  boolean found = false;
  int ret = 0;
  for(int i = 0; i < ((sizeof(Frequencies)) / (sizeof(int))); i++)
  {
    if (Frequencies[i] == x)
    {
    found = true;
    ret = i;
    }
  }

  // If the variable found is never set to true, then
  // the given frequency is not in the Frequency array,
  // and -1 is returned.
  if (found == true)
  {
  return ret;
  }
  else
  {
    // The program should only reach this else statement if an
    // invalid frequency is given to it (which should never happen),
    // but if this else block is triggered 1 is returned.  This 1
    // corresponds to REST, which is a reasonable value to return if
    // an invalid frequnecy is given to the function.  (-1 could be
    // returned, but that would surely cause the program to crash,
    // since the number being returned is supposed to be an index in
    // the Frequency array.)
    return 1;
  }
}
```

Conventional Circuit Diagram:


ARDUINO
Duemilanove
Diecimila
UNO R1/R2
HELLO WORLD !
LCD 16 X 2
U2
10K
10K
10K
10K
220
10K
PIEZO1
S1
S2
S3
S4
S5
R1
R2
R3
R8
R9
D1