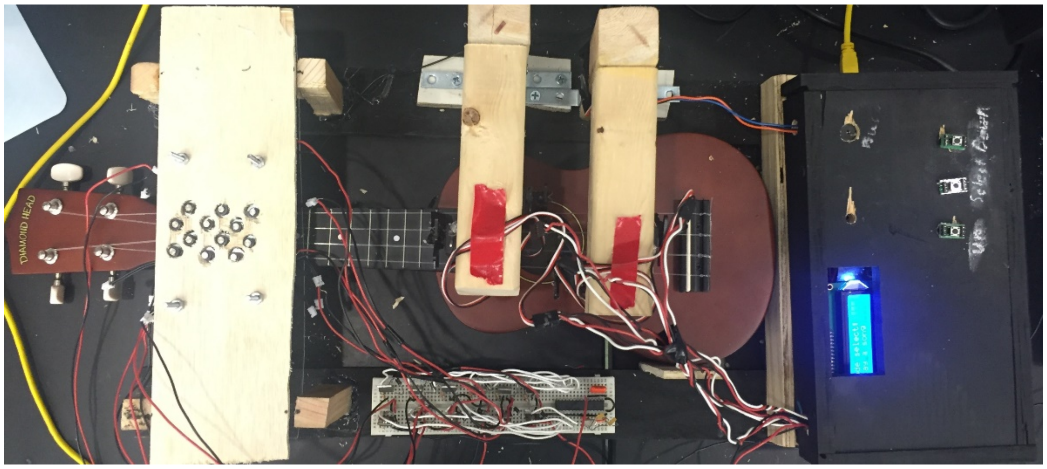
5-2016

Mechalele (Self-Playing Ukulele)

Timothy F. Davison
Trinity University, tdavison@trinity.edu

Arsenio Gonzalez
Trinity University, agonza10@trinity.edu

Taylor Piske
Trinity University, tpiske@trinity.edu



Mechalele (Self-Playing Ukulele)

TIMOTHY F. DAVISON, ARSENIO GONZALEZ, TAYLOR PISKE (*PLEDGED)*

ENGR 4397 FINAL DESIGN REPORT
05/02/16

Table of Contents

Design Summary

This project involves the design and implementation of a self-playing ukulele. The ukulele plays chords and plucks open strings based on a song or chord progression selected by the user. Each string on the ukulele has a designated hobby servo motor that plucks the string with pre-programmed timing as is shown in Figure 1 (3). The plucking is achieved by attaching a flexible piece of cardboard to the shaft of the servo to act as the guitar pick. By moving the motors together at once, the action of strumming can be approximated. The design uses the last three frets on the ukulele for pressing different strings and allows the ukulele to play different notes and chords. The strings are pressed by small push/pull solenoids (2) that are spring loaded to push the string when active and not touch the string when passive. The output of the system is controlled with user selection buttons that selects song or tuning mode for the ukulele from an LCD display (6). The user can either select tuning or song playing mode using three pushbuttons. In tuning mode, a speaker plays the desired note for a string and the LCD display prompts the user to pluck that string. The frequency of the note is detected from a microphone and the display tells the user if the string is tuned, too high, or too low.

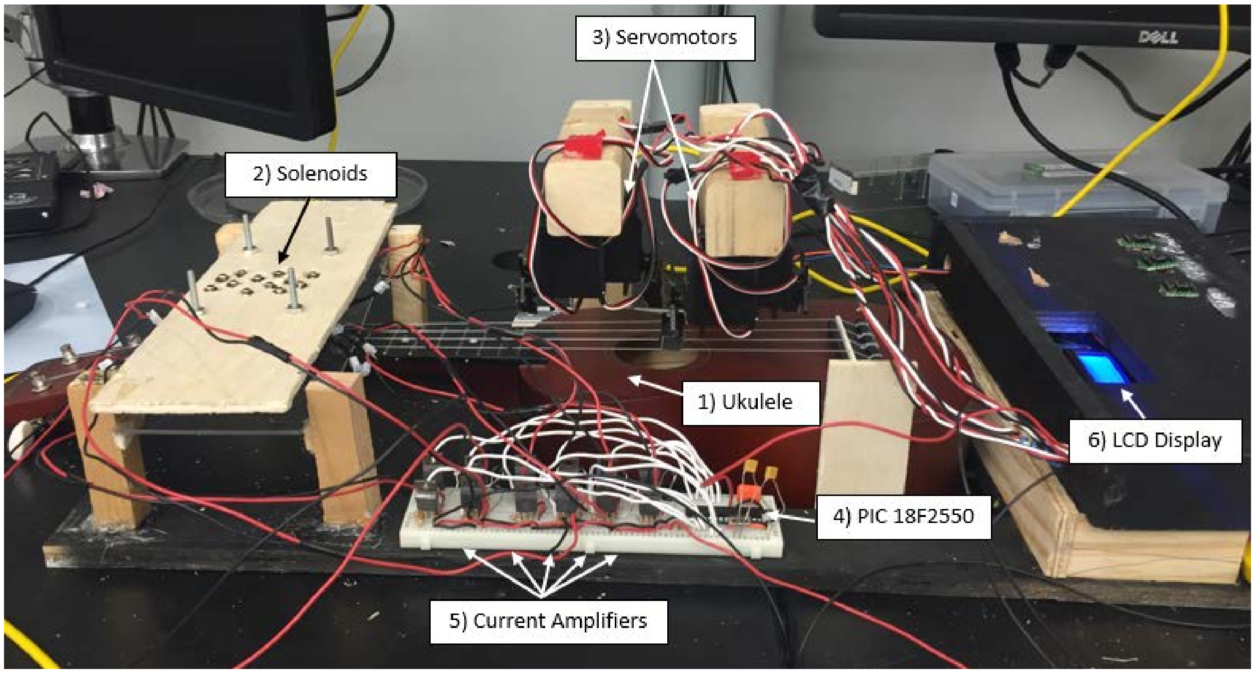


Figure 1: Complete Mechalele System

System Details

This group successfully designed a ukulele that can play a chord procession that is common in popular songs and aid the user in tuning the instrument. The ukulele is contained within a wooden frame along which the solenoid and motor supports are connected. A small black box is used at the end of this frame to contain the Arduinos and complex LCD wiring.

A small, 16x2 LCD screen is used to display information to the user as he or she tunes or plays music with the ukulele. The LCD is programmed to display several things depending on user input: a welcome message at startup, a selection menu where users choose between song and tuning modes, when a song is currently being played, which string is being tuned during tuning mode, whether that string is too high, too low, or tuned, and when the user is exiting tuning mode and entering the selection screen again. Figure 2 was used and adapted to wire the LCD to an Arduino Mega (rather than an Uno).

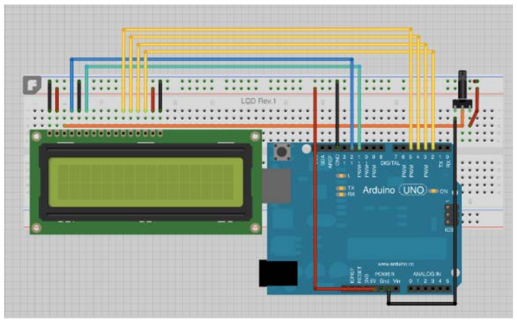


Figure 2: Arduino LCD wiring

In order to navigate the menus displayed on the LCD, three buttons were implemented into the system: up, down and select/cancel. The internal pullup resistors in the Arduinos were used in tandem with the buttons to provide active low manual inputs to users for mode and string selection. The LCD display (6) and push buttons (8) are shown mounted on the black wiring box in Figure 3.

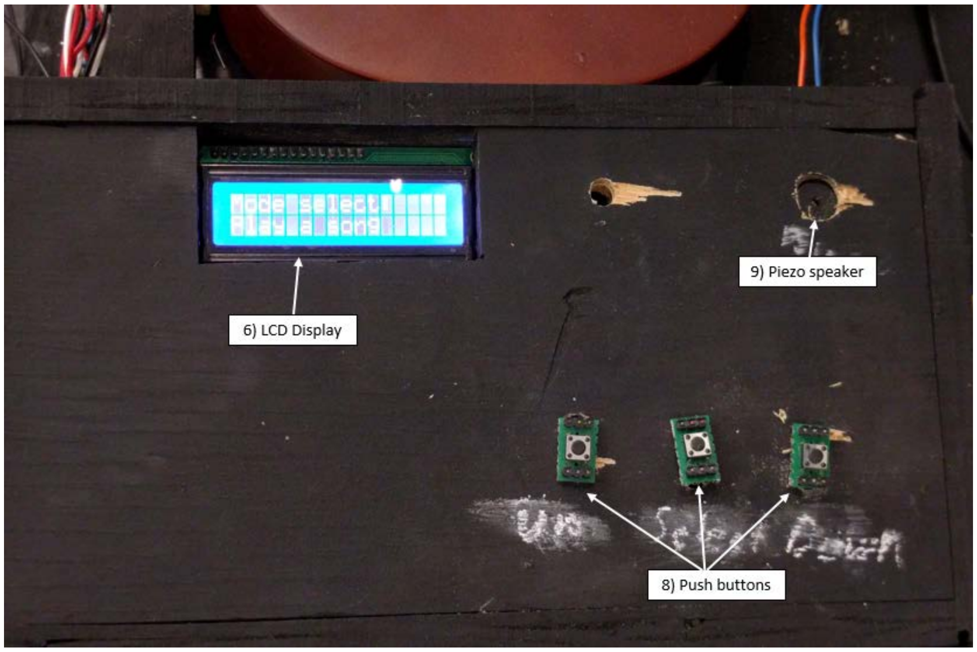


Figure 3: LCD Display, Push Buttons, and Piezo Speaker

When the system is turned on, the Arduino Mega waits for the user to use the push buttons to select either the tuning or song playing option. This highest level procedure is shown in the flowchart in Figure 5. The system is controlled by this basic master program run on an Arduino Mega. It polls for user input at 100 Hz and provides a menu based system of navigation with the LCD. A finite state machine design is implemented to achieve this end - button presses send the controller between LCD display, song playing, and microphone interfacing states. Controller-to-controller communication is implemented to provide the master Arduino Mega with frequency information collected by the slave Arduino Uno and its microphone. The Mega also interfaces with a PIC18F2550, which is wired to the twelve solenoids and programmed to actuate them in a timed routine coordinated with the timed servo routine programmed onto the servo controller, the Mega. The Mega provides the PIC power when the song routine begins and removes power when the song is ended.

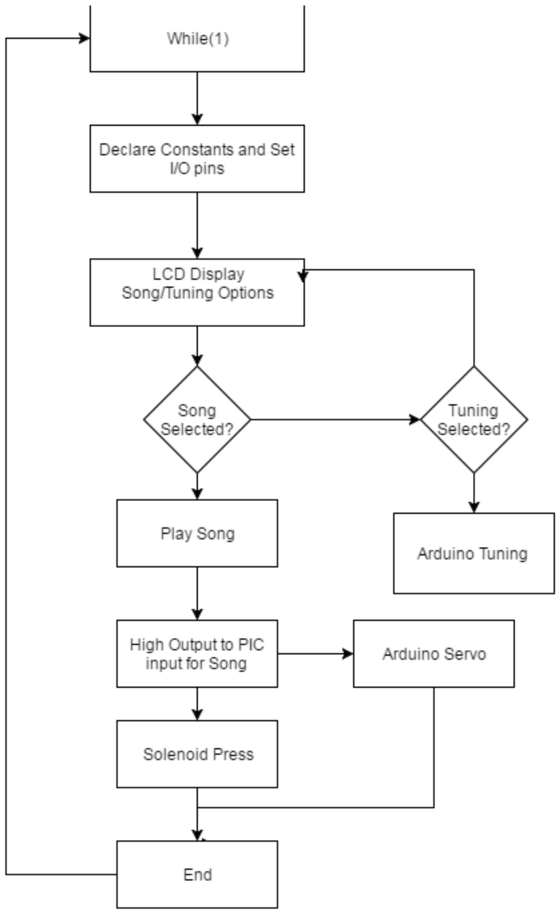

While(1)
Declare Constants and Set I/O pins
LCD Display Song/Tuning Options
Song Selected?
Tuning Selected?
Play Song
Arduino Tuning
High Output to PIC input for Song
Arduino Servo
Solenoid Press
End

Figure 4: Highest level Arduino Mega Flowchart

Tuning Mode

If tuning mode is selected the LCD displays which string is currently being tuned and the user cycles through strings using the left and right pushbuttons, with the center pushbutton taking the user back to the menu. As each string is selected, the system also provides a tuned note for comparison using the piezo speaker (9) in Figure 3. When a string is selected for tuning, the speaker will play the note of that string (either G4, C4, E4, or A4) for five tenths of a second. This both helps the user tune the guitar and trains them to recognize in and out of tune notes. To provide the user with feedback as to whether strings are in tune or not during tuning mode, a microphone is implemented as the system's automatic sensor. The microphone (7) is shown in Figure 5 below.

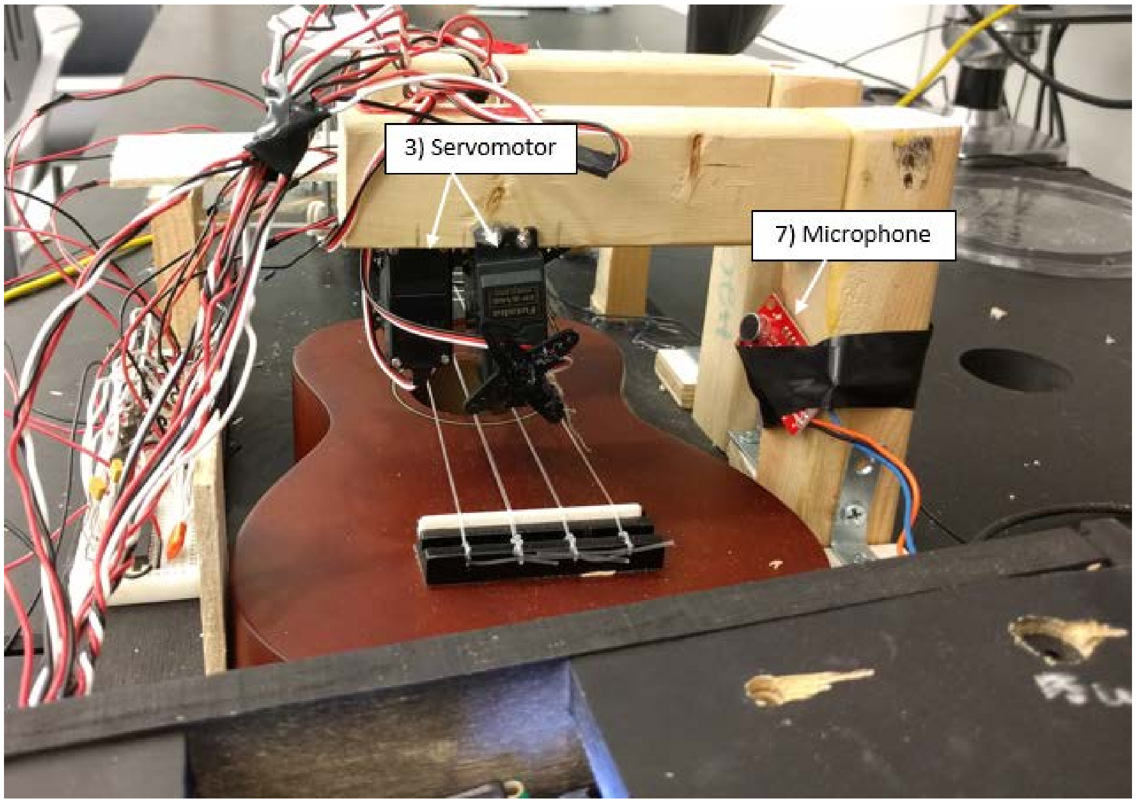


Figure 5: Servomotors and Frequency Detecting Microphone

The microphone provides the slave Arduino with an analog signal representing the audio data captured. The slave Arduino then measures the elapsed time during a single cycle and outputs the frequency of the signal in Hz. This method first filters the input signal frequency before comparing it to some threshold value around the desired frequency for that string. If the note the microphone detects is outside this threshold it signals the Arduino to display "too high" or "too low" on the LCD display. Though the method worked on a bare Arduino Uno, it began to fail when added to the Arduino Mega that had already been wired to buttons, an LCD display, and servo motors. The microphone seemed to be affected by some sort of electrical noise generated by one of the servos. As the servo jerked and whirred randomly in system standby, the microphone would show that it was detecting a note being played at each motor movement. Even after the motors had been disconnected for testing, the microphone provided poor frequency data through the Arduino Mega. The problem was resolved by implementing it onto the bare Uno again, at which point the microphone began receiving excellent data and allowing for a satisfyingly responsive and consistent tuning mode. A flowchart representation of the method used in this tune assisting mode is given in Figure 6.

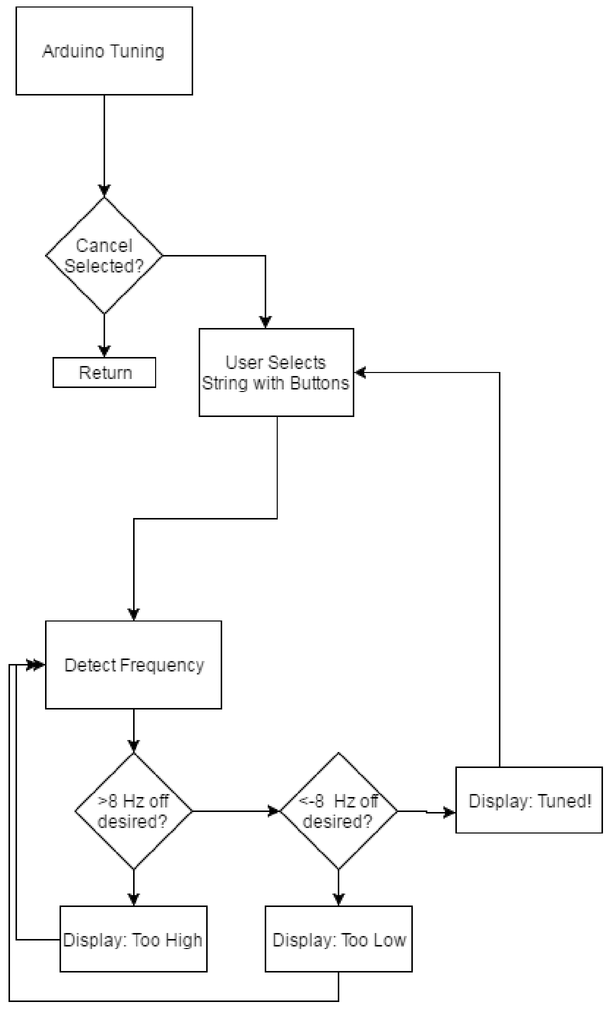


Figure 6: Tuning mode flowchart

Song Playing Mode

If the user selects “Play a Song from the LCD menu”, the motors are sequenced to pluck the strings by the Arduino Mega and the solenoids are sequenced by the PIC to press the strings to produce various notes. A close-up of the solenoids (2) and motor (3) is given in Figure 7.

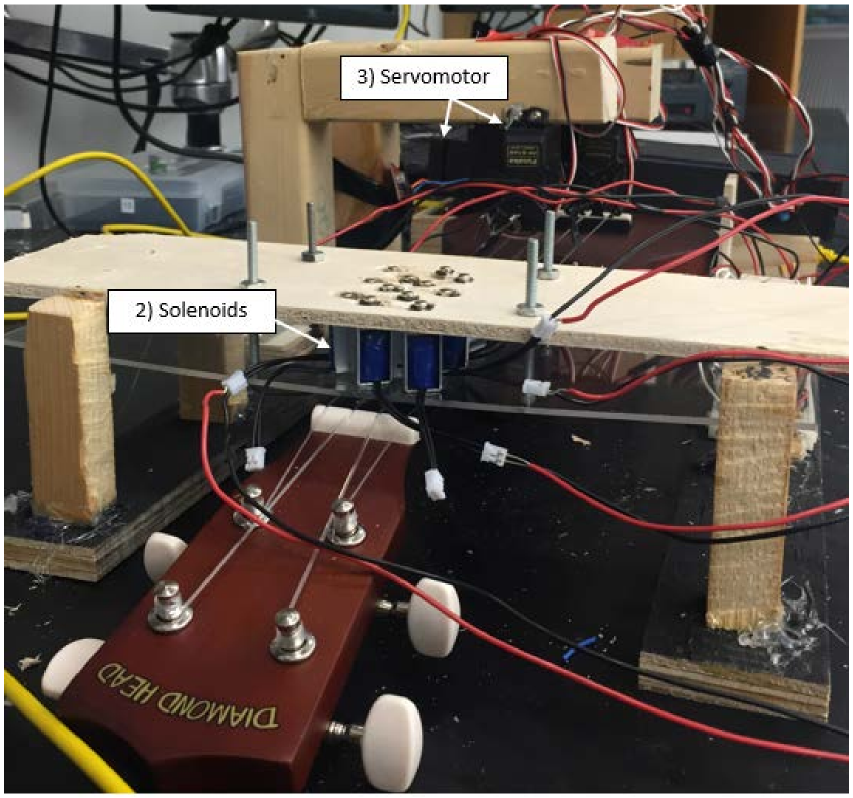


Figure 7: Servomotor and Solenoids

The servos are rigged with ukulele picks made of cardboard that have enough flex to allow for strumming without putting significant stress on the strings or moving the ukulele. At the same time, they are not so flexible that they make a weak sound when plucking strings. The servos are mounted on small wooden beams connected to the ukulele housing and are carefully spaced across the strings. After initially trying to control the servos using pulse-width modulation on a PIC16F88, the servos were switched to Arduino control. With the limited data available on these servos, the Servo library built into the Arduino IDE allowed for more precise and robust control. Some clever programming was used to keep track of servo positions and command them to pluck either left or right in accordance to that position. In the song playing mode the servos alternate between strumming (playing all at once) and sequencing one after another to give the sound of an arpeggio. After pin initialization and setting the initial position, the Arduino subroutine for string plucking is given in Figure 8.

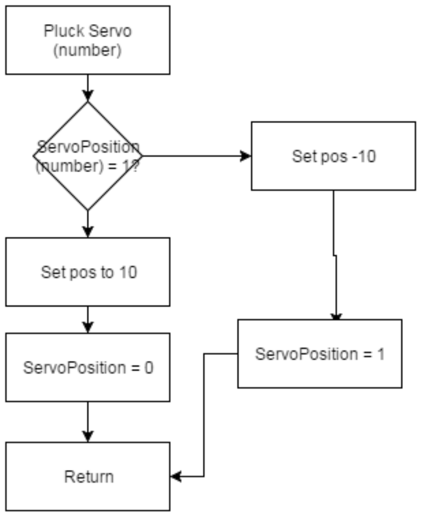


Figure 8: Arduino plucking subroutine

The solenoids are mounted over a piece of plexiglass dotted with holes that were carefully punched in the material to align with the frets and strings intended to be pressed by the solenoids. The solenoids are secured above using a small piece of wood and then connected to the ukulele frame with supports. Each solenoid is supplied current through the use of BJT current amplifiers (5) signaled by a PIC 18F2550 (4) from Figure 1. The solenoids are rated at 5 V, 1.1 A so in order for the PIC to power the solenoids, the PIC signal was used as the input to a power transistor with the solenoids acting as a collector output. In order to supply the solenoids with their required current, a 5V protoboard rated at 1A was used to power them. A fly back diode was used in parallel with the solenoid to provide protection for the circuit when the current is switched on and off. This PIC output circuit is shown in Figure 9.

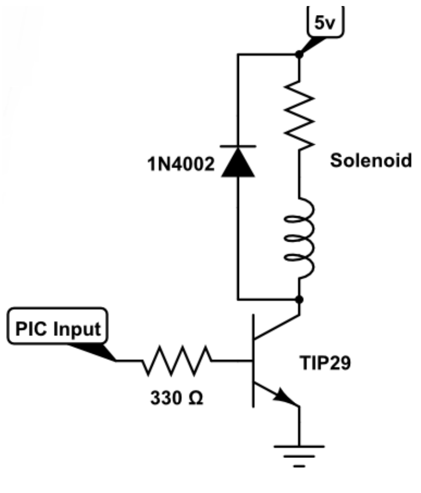


Figure 9: Solenoid collector output circuit

The wiring and Arduinos under the black box are shown in Figure 10 with a potentiometer used to control the brightness of the LCD screen. Each Arduino is powered with a wall socket via USB. The functional diagram for the complete system is given in Figure 11.

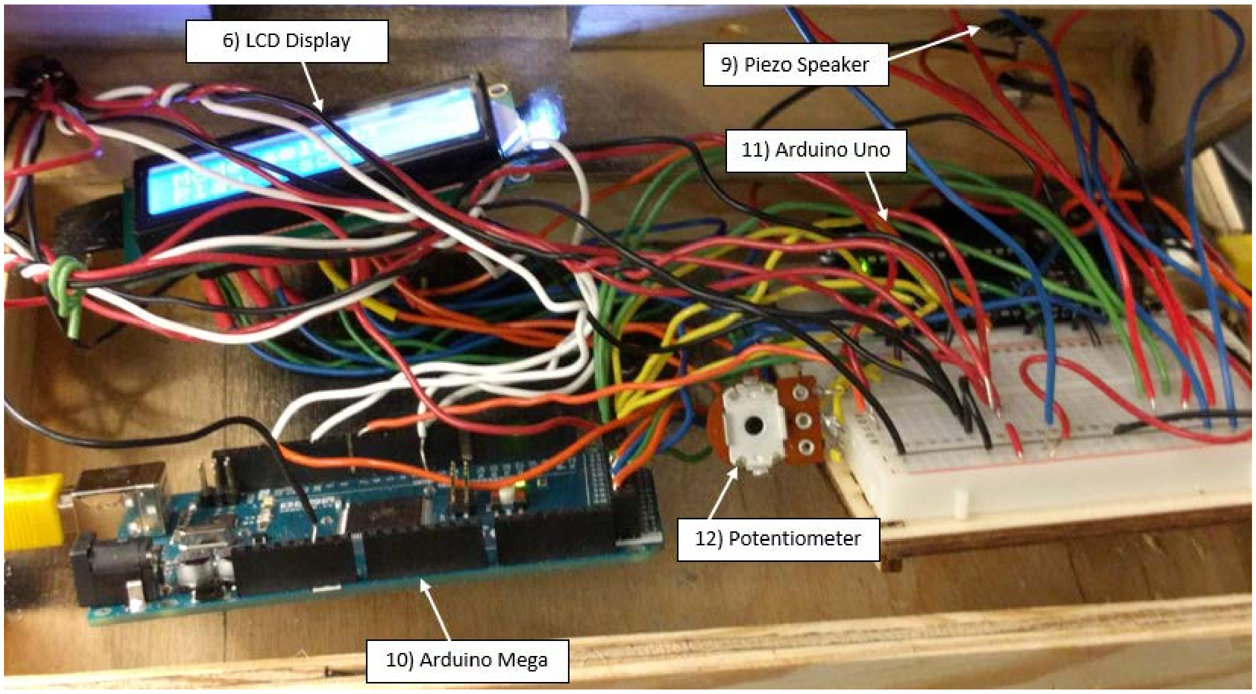


Figure 10: Inside of Ukulele Casing

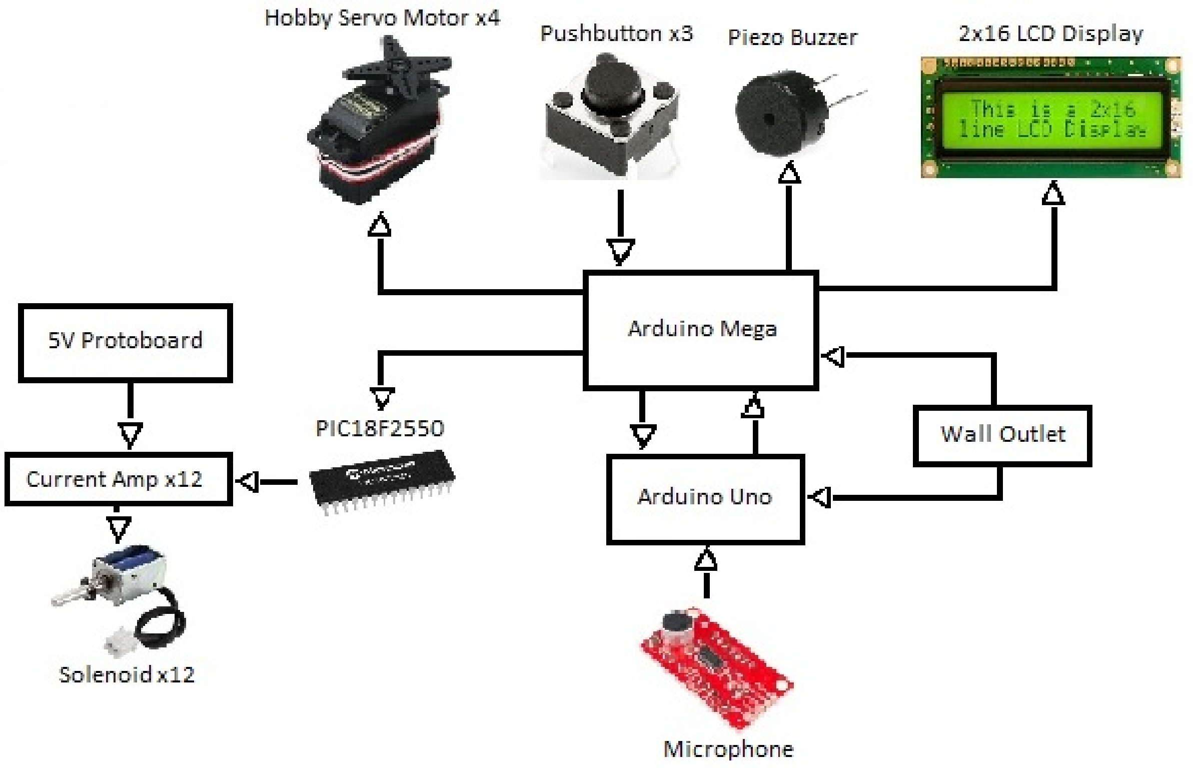


Figure 11: Mechalele complete system functional diagram

Design Evaluation

A. Output Display

- 16x2 LCD Display

The LCD was not introduced in class, so some research was required in order to wire it correctly to the system's master Arduino and to program it to display the required text in the desired circumstance. The Arduino IDE's built-in LiquidCrystal library made coding for the LCD fairly straightforward. Switching between text on the LCD depending on the user input and tuning output required some additional programming knowledge beyond what was covered in this class. Besides maintaining its messy wiring, there were no issues in implementing the display, and it functions reliably.

B. Audio Output Device

- Piezo speaker

The implementation of this speaker was not covered in class, so it took some research to discover how to wire and code it correctly. The Arduino IDE has useful built-in libraries that handle playing perfect pitches with piezo speakers and helped to expedite the coding process. The speaker is strident, but it works well and reliably.

C. Manual User Input

- Pushbuttons

The buttons are decidedly basic, but to implement a more complex control scheme for UI navigation would have been a needless use of resources and could have potentially resulted in a counterintuitive system for the user, which is certainly not the intent. The buttons are simple, streamlined, and completely reliable.

D. Automatic Sensor

- Microphone

The use of a microphone to detect sound frequency was not touched on in class, so a solution was found online at www.prjc.com, written by Paul Stoffregen, and tested until working satisfyingly enough for implementation into the final design. This solution was not designed for the Arduino so some adaptation was needed to alter it for this project's purposes.

E. Actuators, Mechanisms & Hardware

- Solenoids and plexiglass mount
- Hobby servos and wooden beam mounts

The hobby servos were occasionally too weak to overcome the string tension when mounted too low, but when properly placed, they were strong enough to consistently make a clean, firm pluck. The solenoids were also adequately strong so that notes played clearly when the solenoid connected with its string. The wooden beams provide solid support, but the solenoid board was not completely secured in place. It can move slightly, resulting in the solenoids missing their strings when actuated. Fixing them often requires only a slight adjustment in board placing, but messing them up also often requires only a slight adjustment in board placing. This is one aspect of the design that could be easily improved with a more sturdy structure. Overall, this is the aspect of our design that went above and beyond the requirements for this project. Mounting and simultaneous controlling all of these actuators was an arduous task.

F. Logic, Processing & Control

- Finite state machine
- Servo, LiquidCrystal, FreqMeasure libraries
- Arduino-to-Arduino and Arduino-to-PIC digital communication

Extensive research was needed to understand Arduino programming and the various libraries used in this project. Although the use of two Arduino's was not an efficient use of resources, it was necessary to allow both the playing and tuning modes to function properly. Other than this slight design drawback, overall the programming complexity and simultaneous control of numerous actuators in this project is commensurate with additional grading adjustments. Finite state machines and controller-controller communication were also not a part of this course's curriculum, yet were instrumental in this project's functionality. Many hours were invested researching and testing various methods of inter-controller communication before a successful one was finally found. Serial communication and the Wire library yielded no success. It took ingenuity and improvisation to find a working solution to the Arduino-to-PIC communication dilemma.

Partial Parts Lists

Mini Push-Pull Solenoid – 5V

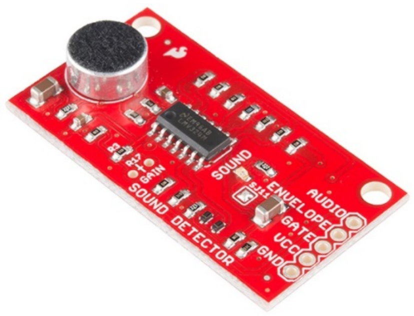
(Adafruit 2776 - $4.95)

This is a very small and powerful solenoid with a 20 mm long body and an armature with a return spring. The solenoid is activated with 5VDC and springs back to its original position when the voltage is removed. Each solenoid required up to 1.1 Amps (measured 0.6 Amps), so the solenoid needed to be placed at the collector of a BJT power transistor in parallel with a fly back diode for protection.

Standard Precision Servo

(Futsaba S148 - $13.99)

High precision servomotor with various connectors for load applications. Operates between 4.8 and 6 V and is controlled with pulse width modulation. Mounting holes provided for easy mounting with wood.



Sound Detector

(SparkFun Sen-12642 - $10.95)

This `microphone' detects sounds and produces output voltages based on the amplitude of sound. This signal is then processed in the Arduino to determine the frequency and tune the ukulele.

Piezo Speaker

(SparkFun COM-07959 - $1.95)

This is a small 12 mm round speaker that operates in the audible range. Operates with 3.5-5V and max current of 35 mA with a sound output of 95 dBA.

Lessons Learned

Problem	**Solution**	**Lesson Learned**
Controlling the servomotors using PIC. There was little documentation on the servomotors and no procedure or specifications on pwm. This made control with the PIC very difficult and led to the motors often missing a pluck and moving in the wrong direction.	We decided to use the Arduino to control the servomotors with its helpful servo library that allowed us to easily input an angle for the servo and proceed with little error.	Do not be afraid to switch your control techniques if it is not working. At some point, the time put in to make something work is not worth it if there is an easier solution. Also, find motors with good datasheets.
Frequency detection using the microphone only worked on the Arduino Uno and was affected by some noise from the servomotors when implemented on the master Arduino Mega.	We originally had the tuning working perfectly on the Uno so with little time left we moved it back to the Uno and used the Uno for frequency detection and calculations with power and signaling from the Mega.	When everything is working individually is might not work when you put it together. Try to put it all together early so you have time to debug. If the debugging does not work do not be afraid to take a step back and use all of your resources.
LCD Display would sometimes cycle through random characters when the user was prompted to select a mode	The wiring in the box was complex so we had to separate some wires and ensure none would touch that would result in this erroneous signal to the LCD display from the Arduino.	Carefully wire everything and make sure that there is plenty of separation and securing between wires.

With guitar picks attached to the motors the strings could not be plucked because the motor was too weak and the picks too rigid.	We attached more flexible, but still strong, pieces of cardboard to the motors that moved over the strings easily while still making a loud enough note.	We originally were set on using guitar picks and thought it was a major road block when we could not pluck strings. It is important to be open to adapting your design and think outside the box.
Communication between microcontrollers was a difficult thing to achieve. We coded our PIC to begin its routine when a listener port was sent a high. It worked when one solenoid was connected, but not when multiple were. Serial communication from Arduino-to-Arduino or Arduino-to-PIC was also a non-cooperator.	We eliminated the need for communication between the Mega and a servo PIC by moving servo control to the Mega, and we made it so that the solenoid PIC was supplied power by the Mega when it were to begin its routine. The PIC was programmed to start the routine as soon as it turned on, fixing the issue.	Simple solutions are valuable in a pinch, so keep them in your back pocket when the sophisticated ones don't pan out in time.

Beyond these five specific lessons we learned during this project, there are several overarching lessons we would like to pass forward to future students. The first is: start early. Once you have an idea for the project there is no time that is too soon to start. We found ourselves plodding along and just meeting the project deliverable deadlines throughout the semester. When it finally came time to put our ideas and components together into the self-playing system we spent significantly more time than we thought we would which led to a few long nights. If we had started building and testing parts earlier in the project, we would have been able to spread out our time more evenly. In addition, we recommend that you try to have a near finished project before the first early bird deadline. Inevitably, something will go wrong and you will have to come up with new ideas and debug until the final deadline. If not, you get 10 extra points, so it is a win-win.

When ordering components, if you have the budget, get yourself a spare to whatever it is you are buying. Not only will it save you valuable time in the case that you misplace or inadvertently destroy one of the critical pieces to your design and have to replace it, but it will afford you the flexibility to expand or alter your design as the project progresses. Mechatronics projects are dynamic, shifting creatures, and you must be capable of adapting to any new discovery or pitfall that is inevitably waiting for you in the design process. As inexperience undergrads, we are often travelling into unknown territory when we commit to a project like this. Make sure you have a little more than you think you need, because anything can happen.

And it goes without saying, but back up your code, documents, and schematics on Google Drive or something similar whenever you get the chance.

Another important lesson we learned from this project is that you should select something you are interested in and will want to spend many hours working on. We nearly selected an automatic dog feeder as our project because it was a less complex design, but ultimately chose the ukulele because our group enjoys music and would enjoy creating something that plays it autonomously. You will spend a lot of time on this project no matter what you pick, so it is essential that you pick something that you think is fun and are willing to put in a little extra work for to create an awesome final product.

Appendix

Materials and Wiring Diagrams

Table 1: Purchased Materials

Item	Company	Quantity	Price
PIC 18F2550	Digikey	1	$4.30
SEN-12642 (Sound Detector)	SparkFun	1	$10.95
Mini Push-Pull Solenoid	Adafruit	12	$4.46
		Total	$68.77

Table 2: Borrowed Materials (not including wires and wood)

Item	Company	Quantity
Servomotors (S148)	Futsaba	4
Arduino Uno	Arduino	1
Arduino Mega	Arduino	1
LCD Display	Arduino	1
Piezo Speaker	Arduino	1
NO Pushbutton	Arduino	3

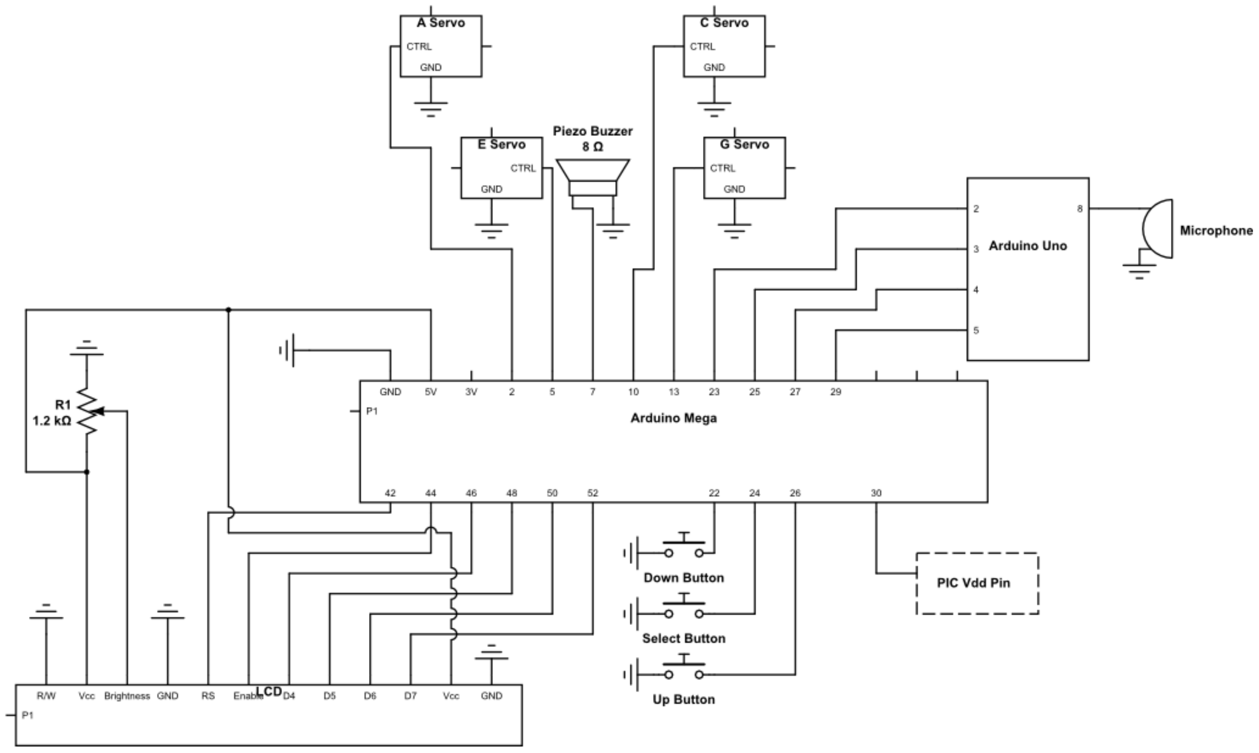


Figure 12: Arduino Wiring Diagram

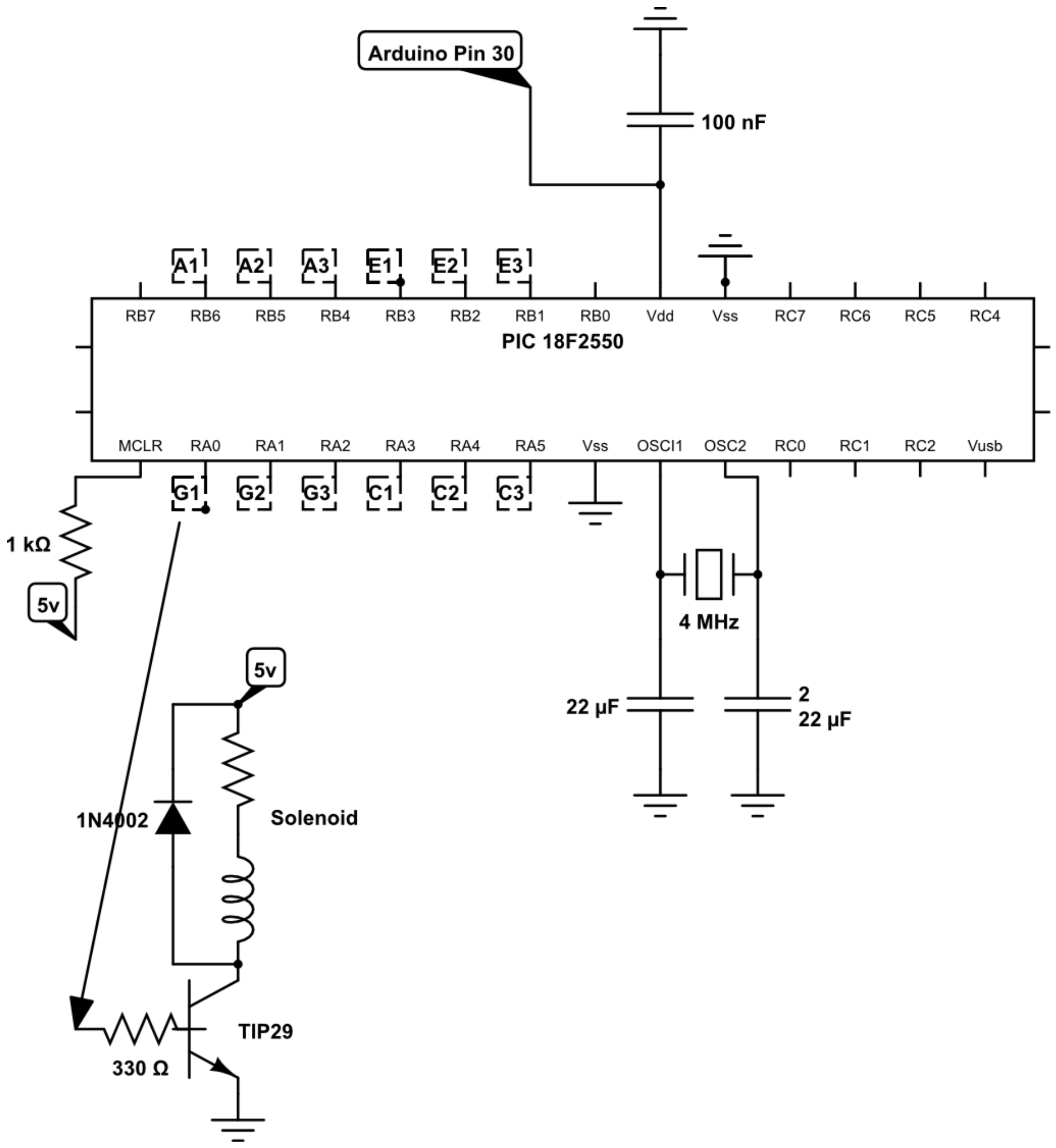


Figure 13: Wiring Diagram for PIC 18F2550

Arduino and PIC Code

I. Arduino Mega Master Sketch

```
/* ukuleleMaster.ino
  By Taylor Piske, Arsenio Gonzales, and Tim Davidson
  Written for Arduino Mega interfacing with a PIC18F2550, an Uno,
  an LCD display, buttons, hobby servos, and a piezo buzzer.

  This sketch is designed to control self-playing ukulele system. Users
  will be able to choose between two modes: song play or tuning. In
  song play, the Mega will begin a timed picking routine with the
  servos. At the same time, it will power the PIC18F2550, which will
  actuate solenoids to finger frets in time with the servos' picking.

  In tuning mode, the Mega will communicate to the Uno the string being
  tuned with two digital pinouts. The Uno will listen to its microphone
  for the frequency of the plucked note and determine whether the note
  is too low, too high, or out of tune in relation to the string's target
  pitch. It will communicate that in-tuneness with two digital pinouts,
  and the Mega will display that result on the LCD for the user.

  An LCD display will display mode choices as well, and three buttons will
  be used to move up, move down, and select/cancel.

  The LCD circuit:
 * RS pin to digital pin 42
 * Enable pin to digital pin 44
 * D4 pin to digital pin 46
 * D5 pin to digital pin 48
 * D6 pin to digital pin 50
 * D7 pin to digital pin 52
 * R/W pin to ground
 * VSS pin to ground
 * VCC pin to 5V
 * 10K resistor:
 * ends to +5V and ground
 * wiper to LCD VO pin (pin 3)
*/

#include <LiquidCrystal.h>
#include <Servo.h>
#include "pitches.h"
#include <SoftwareSerial.h>

// initialize the library with the numbers of the interface pins
LiquidCrystal lcd(42,44,46,48,50,52);
//LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
```

```
// initialize all servos
Servo Gservo;
Servo Cservo;
Servo Eservo;
Servo Aservo;

// set button, microphone, speaker, and PIC pins
const int downBut = 22;
const int selectBut = 24;
const int upBut = 26;
const int speaker = 7;

// frequency communication with Arduino Uno.
// StringSig is two bits telling the Uno which string is being tuned,
// 1 (00), 2 (01), 3 (10), or 4 (11).
// inTune signals tell the Mega whether the string is in tune, too low, or too high.
// Too low is highOrLow = 0, too high is highOrLow = 1
const int msdStringSig = 23;
const int lsdStringSig = 25;
const int inTune = 27;
const int highOrLow = 29;

// Song command pins talking to the solenoid PIC. High when playing song, low when not
const int beatlesSolCmd = 30;
//const int zeldaSolCmd = 32;
int beatlesSerCmd = 0;
int zeldaSerCmd = 0;

// String tuning command pins talking to the PICS
int tuneGCmd = 0;
int tuneCCmd = 0;
int tuneECmd = 0;
int tuneACmd = 0;

/* states, assigned to variable 'option' */
// 1.0: on the select screen for 'Here Comes the Sun'
// 1.1: wait state going from state 2.0 -> 1.0
// 1.9: wait state going from state 1.0 -> 2.0
//
// 2.0: on the select screen for 'Zelda's Lullaby'
// 2.1: wait state going from state 3.0 -> 2.0
// 2.9: wait state going from state 2.0 -> 3.0
//
// 3.0: on the select screen for the tuner
// 0.1: wait state going from state 1.0 -> 3.0
// 0.9: wait state going from state 3.0 -> 1.0
//
```

```
// 3.5: wait state going from state 3.0 -> 3.51
// 3.511: wait state going from state 3.52 -> 3.51
// 3.519: wait state going from state 3.51 -> 3.52
// 3.521: wait state going from state 3.53 -> 3.52
// 3.529: wait state going from state 3.52 -> 3.53
// 3.531: wait state going from state 3.54 -> 3.53
// 3.539: wait state going from state 3.53 -> 3.54
// 3.501: wait state going from state 3.51 -> 3.54
// 3.509: wait state going from state 3.54 -> 3.51
//
// 1.4: wait state going from state 1.0 -> 1.5
// 2.4: wait state going from state 2.0 -> 2.5
//
// 1.5: playing 'Here Comes the Sun'
// 2.5: playing 'Zelda's Lullaby'
// 3.51: in tuning mode -> tuning first string, G (392.0 Hz)
// 3.52: in tuning mode -> tuning second string, C (261.6 Hz)
// 3.53: in tuning mode -> tuning third string, E (329.6 Hz)
// 3.54: in tuning mode -> tuning fourth string, A (440.0 Hz)
//
// 0.0: wait state going from song play or tuning mode -> 1.0 when select/cancel is pressed
//
// -1.0: intermediate state for going from the intro screen to song/mode select
float option;

// Servo positions in degrees and position trackers, 0 being left, 1 being right
// Gservo lPos = 95, rPos = 140
// Cservo lPos = 85, rPos = 135
// Eservo lPos = 110, rPos = 160
// Aservo lPos = 90, rPos = 135
int Gpos = 0;
int Cpos = 0;
int Epos = 0;
int Apos = 0;

// Button readers
int upState;
int selectState;
int downState;

// Frequency holder
float frequency;

// Used to silence the speaker after 2 seconds of playing.
// Also used to send the system back to State 1.0 when a song has finished playing.
int tonePlayed = 0;
double count = 0.0;
```

```
void setup() {
  Serial.begin(9600);
  lcd.begin(16,2);
  pinMode(upBut,INPUT_PULLUP);
  pinMode(selectBut,INPUT_PULLUP);
  pinMode(downBut,INPUT_PULLUP);
  pinMode(beatlesSolCmd,OUTPUT);
  pinMode(msdStringSig,OUTPUT);
  pinMode(lsdStringSig,OUTPUT);
  pinMode(inTune,INPUT);
  pinMode(highOrLow,INPUT);
  lcd.setCursor(0,0);
  lcd.print("THE MECHALELE");
  lcd.setCursor(0,1);
  lcd.print("Play or tune!");
  delay(2500);
  lcd.clear();
  delay(500);
  option = -1.0;
  resetServos();
}

void loop() {
  upState = digitalRead(upBut);
  selectState = digitalRead(selectBut);
  downState = digitalRead(downBut);

  /////// song and tuning mode select screens ////////
  if (option == 1.0) {
    if (upState == 0) {
      option = 2.9;
    }
    if (downState == 0) {
      option = 0.1;
    }
    if (selectState == 0) {
      option = 1.4;
      lcd.clear();
      lcd.setCursor(0,0);
      lcd.print("Playing song...     ");
      digitalWrite(beatlesSolCmd,HIGH);
      beatlesSerCmd = 1;
      count = 0.0;
    }
  }

  else if (option == 2.0) {
    if (upState == 0) {
      option = 2.9;
    }
    if (downState == 0) {
      option = 1.1;
    }

    if (selectState == 0) {
      option = 2.4;
      lcd.clear();
      lcd.setCursor(0,0);
      lcd.print("Enjoy Z's Lullaby!");
      lcd.setCursor(0,1);
      lcd.print("Cancel to quit");
      //digitalWrite(zeldaSolCmd,HIGH);
      //zeldaSerCmd = 1;
      count = 0.0;
    }
  }

  else if (option == 3.0) {
    if (upState == 0) {
      option = 0.9;
    }
    if (downState == 0) {
      //option = 2.1;  // for multiple songs
      option = 1.1;
    }

    if (selectState == 0) {
      count = 0;
      option = 3.50;
      lcd.clear();
    }
  }

  else if (option == 3.51 && selectState == 1) {
    lcd.setCursor(0,0);
    lcd.print("Tuning Mode");
    lcd.setCursor(0,1);
    if ((digitalRead(inTune) == 1) && (digitalRead(highOrLow) == 1)) {
      //lcd.print("String 1  ...          ");
    }
    else if (digitalRead(inTune) == 1 && (digitalRead(highOrLow) == 0)) {
      lcd.print("String 1 Tuned! ");
    }
    else if ((digitalRead(inTune) == 0) && (digitalRead(highOrLow)) == 0) { //(frequency > 360 &&
frequency < 410) {
//      if (frequency < 392-8) {
      lcd.print("String 1 Too Low");
    }
    else if ((digitalRead(inTune) == 0) && (digitalRead(highOrLow) == 1)) { //(frequency > 392+8) {
      lcd.print("String 1 Too Hi ");
    }
//    }
//    lcd.print(frequency);
//    tuningMode();
    if (upState == 0) {
    option = 3.519;
    }
    if (downState == 0) {
      option = 3.501;
    }
    if (count <= 5.0 && tonePlayed == 0) {
      tone(speaker,NOTE_G4);
    }
    else {
      noTone(speaker);
      tonePlayed = 1;
    }
    count += 1.0;
  }

  else if (option == 3.52 && selectState == 1) {
    lcd.setCursor(0,0);
    lcd.print("Tuning Mode");
    lcd.setCursor(0,1);
    if ((digitalRead(inTune) == 1) && (digitalRead(highOrLow) == 1)) {
      //lcd.print("String 2  ...          ");
    }
    else if ((digitalRead(inTune) == 1) && (digitalRead(highOrLow) == 0)) {
      lcd.print("String 2 Tuned! ");
    }
    else if ((digitalRead(inTune) == 0) && (digitalRead(highOrLow) == 0)) { //(frequency > 210 &&
frequency < 350) {
      //if (frequency < 262-8) {
      lcd.print("String 2 Too Low");
    }
    else if ((digitalRead(inTune)) == 0 && (digitalRead(highOrLow) == 1)) { //(frequency > 262+8) {
      lcd.print("String 2 Too Hi ");
    }
//    }
//    lcd.print(frequency);
//    tuningMode();
    if (upState == 0) {
      option = 3.529;
    }
    if (downState == 0) {
      option = 3.511;
    }
    if (count <= 5.0 && tonePlayed == 0) {
      tone(speaker,NOTE_C4);
    }
    else {
      noTone(speaker);
      tonePlayed = 1;
    }
    count += 1.0;
  }

  else if (option == 3.53 && selectState == 1) {
    lcd.setCursor(0,0);
    lcd.print("Tuning Mode");
    lcd.setCursor(0,1);
    if ((digitalRead(inTune) == 1) && (digitalRead(highOrLow) == 1)) {
      //lcd.print("String 3  ...          ");
    }
    else if ((digitalRead(inTune) == 1) && (digitalRead(highOrLow) == 0)) {
      lcd.print("String 3 Tuned! ");
    }
    else if ((digitalRead(inTune) == 0) && (digitalRead(highOrLow) == 0)) { //(frequency > 290 &&
frequency < 410) {
      //if (frequency < 330-8) {
      lcd.print("String 3 Too Low");
    }
    else if ((digitalRead(inTune) == 0) && (digitalRead(highOrLow) == 1)) { //(frequency > 330+8) {
      lcd.print("String 3 Too Hi ");
    }
//    }
//    lcd.print(frequency);
//    tuningMode();
    if (upState == 0) {
      option = 3.539;
    }
    if (downState == 0) {
      option = 3.521;
    }
    if (count <= 5.0 && tonePlayed == 0) {
      tone(speaker,NOTE_E4);
    }
    else {
      noTone(speaker);
      tonePlayed = 1;
    }
    count += 1.0;
```

```
 }

 else if (option == 3.54 && selectState == 1) {
  lcd.setCursor(0,0);
  lcd.print("Tuning Mode");
  lcd.setCursor(0,1);
  if ((digitalRead(inTune) == 1) && (digitalRead(highOrLow) == 1)) {
   //lcd.print("String 4  ...         ");
  }
  else if ((digitalRead(inTune) == 1) && (digitalRead(highOrLow) == 0)) {
   lcd.print("String 4 Tuned! ");
  }
  else if ((digitalRead(inTune) == 0) && (digitalRead(highOrLow) == 0)) { //(frequency > 380 &&
frequency < 500) {
//     if (frequency < 440-8) {
   lcd.print("String 4 Too Low");
  }
  else if ((digitalRead(inTune) == 0) && (digitalRead(highOrLow) == 1)) { //(frequency > 440+8) {
   lcd.print("String 4 Too Hi ");
  }
//   }
//   lcd.print(frequency);
//   tuningMode();
  if (upState == 0) {
   option = 3.509;
  }
  if (downState == 0) {
   option = 3.531;
  }
  if (count <= 5.0 && tonePlayed == 0) {
   tone(speaker,NOTE_A4);
  }
  else {
   noTone(speaker);
   tonePlayed = 1;
  }
  count += 1.0;
 }

 // song playing states. When cancel is pressed, interupt PICs and move back to song select states
 else if ((option == 1.5 || option == 2.5 || (option >= 3.51 && option <= 3.54)) && selectState == 0) {
  beatlesSerCmd = 0;
  zeldaSerCmd = 0;
  digitalWrite(beatlesSolCmd,LOW);
  //digitalWrite(zeldaSolCmd,LOW);
  digitalWrite(tuneGCmd,LOW);
  digitalWrite(tuneCCmd,LOW);
  digitalWrite(tuneECmd,LOW);
```

```
    digitalWrite(tuneACmd,LOW);
    Gservo.attach(13);
    Cservo.attach(10);
    Eservo.attach(5);
    Aservo.attach(2);
    resetServos();
    lcd.clear();
    lcd.setCursor(0,0);
//    lcd.print("Cancelled...");
    lcd.print("Done tuning!");
    lcd.setCursor(0,1);
    lcd.print("Back to menu...");
    delay(2000);
    lcd.clear();
    option = 0.0;
  }

  ////////// wait states //////////
  else if (option == 0.1 && downState == 1) {
    lcd.clear();
    lcd.print("Mode select:");
    lcd.setCursor(0,1);
    lcd.print("Tune the uke");
    option = 3.0;
  }
  else if (option == 0.9 && upState == 1) {
    lcd.clear();
//    lcd.print("Song select:");
    lcd.print("Mode select:          ");
    lcd.setCursor(0,1);
//    lcd.print("The Beatles      ");
    lcd.print("Play a song          ");
    option = 1.0;
  }
  else if (option == 1.1 && downState == 1) {
    lcd.clear();
//    lcd.print("Song select:");
    lcd.print("Mode select:          ");
    lcd.setCursor(0,1);
//    lcd.print("The Beatles      ");
    lcd.print("Play a song          ");
    option = 1.0;
  }
  else if (option == 1.9 && upState == 1) {
    lcd.clear();
    lcd.print("Song select:");
    lcd.setCursor(0,1);
    lcd.print("Zelda's Lullaby      ");
    option = 2.0;
  }
  else if (option == 2.1 && downState == 1) {
    lcd.clear();
    lcd.print("Song select:");
    lcd.setCursor(0,1);
    lcd.print("Zelda's Lullaby       ");
    option = 2.0;
  }
  else if (option == 2.9 && upState == 1) {
    lcd.clear();
    lcd.print("Mode select:");
    lcd.setCursor(0,1);
    lcd.print("Tune the uke");
    option = 3.0;
  }
  else if (option == 3.511 && downState == 1) {
    count = 0;
    tonePlayed = 0;
    tuneGCmd = 1;
    tuneCCmd = 0;
    digitalWrite(lsdStringSig,LOW);
    digitalWrite(msdStringSig,LOW);
    lcd.clear();
    lcd.setCursor(0,1);
    lcd.print("String 1");
    option = 3.51;
  }
  else if (option == 3.519 && upState == 1) {
    count = 0;
    tonePlayed = 0;
    tuneCCmd = 1;
    tuneGCmd = 0;
    digitalWrite(lsdStringSig,HIGH);
    digitalWrite(msdStringSig,LOW);
    lcd.clear();
    lcd.setCursor(0,1);
    lcd.print("String 2");
    option = 3.52;
  }
  else if (option == 3.521 && downState == 1) {
    count = 0;
    tonePlayed = 0;
    tuneCCmd = 1;
    tuneECmd = 0;
    digitalWrite(lsdStringSig,HIGH);
    digitalWrite(msdStringSig,LOW);
    lcd.clear();
    lcd.setCursor(0,1);
    lcd.print("String 2");
    option = 3.52;
  }
  else if (option == 3.529 && upState == 1) {
    count = 0;
    tonePlayed = 0;
    tuneECmd = 1;
    tuneCCmd = 0;
    digitalWrite(lsdStringSig,LOW);
    digitalWrite(msdStringSig,HIGH);
    lcd.clear();
    lcd.setCursor(0,1);
    lcd.print("String 3");
    option = 3.53;
  }
  else if (option == 3.531 && downState == 1) {
    count = 0;
    tonePlayed = 0;
    tuneECmd = 1;
    tuneACmd = 0;
    digitalWrite(lsdStringSig,LOW);
    digitalWrite(msdStringSig,HIGH);
    lcd.clear();
    lcd.setCursor(0,1);
    lcd.print("String 3");
    option = 3.53;
  }
  else if (option == 3.539 && upState == 1) {
    count = 0;
    tonePlayed = 0;
    tuneACmd = 1;
    tuneECmd = 0;
    digitalWrite(lsdStringSig,HIGH);
    digitalWrite(msdStringSig,HIGH);
    lcd.clear();
    lcd.setCursor(0,1);
    lcd.print("String 4");
    option = 3.54;
  }
  else if (option == 3.501 && downState == 1) {
    count = 0;
    tonePlayed = 0;
    tuneACmd = 1;
    tuneGCmd = 0;
    digitalWrite(lsdStringSig,HIGH);
    digitalWrite(msdStringSig,HIGH);
    lcd.clear();
```

```
    lcd.setCursor(0,1);
    lcd.print("String 4");
    option = 3.54;
  }
  else if (option == 3.509 && upState == 1) {
    count = 0;
    tonePlayed = 0;
    tuneGCmd = 1;
    tuneACmd = 0;
    digitalWrite(lsdStringSig,LOW);
    digitalWrite(msdStringSig,LOW);
    lcd.clear();
    lcd.setCursor(0,1);
    lcd.print("String 1");
    option = 3.51;
  }
  else if (option == 3.5 && selectState == 1) {
    count = 0;
    tonePlayed = 0;
    tuneGCmd = 1;
    Gservo.detach();
    Cservo.detach();
    Eservo.detach();
    Aservo.detach();
    digitalWrite(lsdStringSig,LOW);
    digitalWrite(msdStringSig,LOW);
    lcd.clear();
    lcd.setCursor(0,1);
    lcd.print("String 1");
    option = 3.51;
  }
  else if (option == 0.0 && selectState == 1) {
    lcd.clear();
//    lcd.print("Song select:");
    lcd.print("Mode select:            ");
    lcd.setCursor(0,1);
//    lcd.print("The Beatles       ");
    lcd.print("Play a song           ");
    option = 1.0;
  }
  else if (option == 1.4 && selectState == 1) {
    option = 1.5;
  }
  else if (option == 2.4 && selectState == 1) {
    option = 2.5;
  }
  else if (option == -1.0) {
//    lcd.print("Song select:");
```

```
    lcd.print("Mode select:              ");
    lcd.setCursor(0,1);
//    lcd.print("The Beatles       ");
    lcd.print("Play a song            ");
    Gservo.attach(13);
    Cservo.attach(10);
    Eservo.attach(5);
    Aservo.attach(2);
    option = 1;
  }


  //////// Check servo song and tuning commands for initialization /////////
  if (beatlesSerCmd == 1) {
    delay(100);
    // C major chord
    sweep();
    delay(700);
    sweep();
    delay(350);
    pluckC();
    delay(350);
    pluckE();
    delay(350);
    pluckA();
    delay(350);
    pluckC();
    delay(350);
    pluckE();
    delay(350);
    // G major chord
    sweep();
    delay(700);
    sweep();
    delay(350);
    pluckC();
    delay(350);
    pluckE();
    delay(350);
    pluckA();
    delay(350);
    pluckC();
    delay(350);
    pluckE();
    delay(350);
    // A major chord
    sweep();
    delay(700);
    sweep();
    delay(350);
    pluckC();
    delay(350);
    pluckE();
    delay(350);
    pluckA();
    delay(350);
    pluckC();
    delay(350);
    pluckE();
    delay(350);
    // C major chord
    sweep();
    delay(2800);
    beatlesSerCmd = 0;
    digitalWrite(beatlesSolCmd,LOW);
    lcd.clear();
//    lcd.print("Song select:");
    lcd.print("Mode select:            ");
    lcd.setCursor(0,1);
//    lcd.print("The Beatles       ");
    lcd.print("Play a song           ");
    option = 1.0;
  }

//  if (zeldaSerCmd == 1) {
//    if (count == 1.0) {
//      pluckG();
//    }
//    else if (count == 2.0) {
//      sweep();
//    }
//    else if (count > 100.0) {
//      zeldaSerCmd = 0;
//    }
//    count += 0.01;
//  }

  //////// Checking states and counts via the Serial Monitor ////////
  Serial.print("Option: ");
  Serial.print(option);
  Serial.print(", StringSig: ");
  Serial.print(digitalRead(msdStringSig));
  Serial.print(digitalRead(lsdStringSig));
  Serial.print(", Too High: ");
  Serial.print(digitalRead(highOrLow));
  Serial.print(", Tuned: ");
  Serial.println(digitalRead(inTune));
  delay(10);
}

///// Servo control functions for each string /////
void pluckG() {
  Serial.println("Plucking G");
  if (Gpos == 0) {
    Gservo.write(140);
    delay(10);
    Gpos = 1;
  }
  else {
    Gservo.write(90);
    delay(10);
    Gpos = 0;
  }
}

void pluckC() {
  Serial.println("Plucking C");
  if (Cpos == 0) {
    Cservo.write(122);
    delay(10);
    Cpos = 1;
  }
  else {
    Cservo.write(75);
    delay(10);
    Cpos = 0;
  }
}

void pluckE() {
  Serial.println("Plucking E");
  if (Epos == 0) {
    Eservo.write(115);
    delay(10);
    Epos = 1;
  }
  else {
    Eservo.write(160);
    delay(10);
    Epos = 0;
  }
}

void pluckA() {
  Serial.println("Plucking A");
  if (Apos == 0) {
    Aservo.write(100);
    delay(10);
    Apos = 1;
  }
  else {
    Aservo.write(140);
    delay(10);
    Apos = 0;
  }
}

void sweep() {
  Serial.println("Sweeping strings");
  pluckG();
  delay(10);
  pluckC();
  delay(10);
  pluckE();
  delay(10);
  pluckA();
  delay(10);
}

void resetServos() {
  Serial.println("Resetting servos");
  //if (Gpos == 1) {
    Gservo.write(90);
    delay(10);
    Gpos = 0;
  //}
  //if (Cpos == 1) {
    Cservo.write(75);
    delay(10);
    Cpos = 0;
  //}
  //if (Epos == 1) {
    Eservo.write(160);
    delay(10);
    Epos = 0;
  //}
  //if (Apos == 1) {
    Aservo.write(140);
    delay(10);
    Apos = 0;
  //}
}
```

II. Arduino Uno Frequency Measurement Sketch

```
/* FreqMeasureUno.ino
 *  by Taylor Piske, Arsenio Gonzales, and Tim Davison
 *
 *  For the Mechalele. Reads two digital pinouts of the
 *  master Mega controller to determine which string
 *  is being tuned, reads frequency from Sparkfun
 *  microphone, and signals to the Mega whether the played
 *  string is too high, too low, or tuned by communicating
 *  via its own two digital pinouts.
 */

#include <FreqMeasure.h>

/* Most significant and least significant digits of
 * string signal from the Mega. 00 means string 1,
 * 01 string 2, 10 string 3, 11 string 4 */
const int msdStringSig = 2;
const int lsdStringSig = 3;

/* Result signal sent to the Mega for display on the
 * LCD. inTune is msd, highOrLow is lsd.
 * 00 is string too low, 01 is string too high
 * 10 is string in tune, and 11 is bad frequency
 * measurement, ignore. */
const int inTune = 4;
const int highOrLow = 5;

void setup() {
  Serial.begin(9600);
  FreqMeasure.begin();
  pinMode(msdStringSig,INPUT);
  pinMode(lsdStringSig,INPUT);
  pinMode(inTune,OUTPUT);
  pinMode(highOrLow,OUTPUT);
}

double sum=0;
int count=0;
float frequency;

void loop() {
  if (FreqMeasure.available()) {
    // average several reading together
```

```
sum = sum + FreqMeasure.read();
count = count + 1;
if (count > 60) {
  frequency = FreqMeasure.countToFrequency(sum / count);
  sum = 0;
  count = 0;

  // Tuning string 1
  if (digitalRead(msdStringSig) == 0 && digitalRead(lsdStringSig) == 0) {
    if (frequency > 320 && frequency < 450) {
      if (frequency < 392-8) {
        digitalWrite(inTune,LOW);
        digitalWrite(highOrLow,LOW);
      }
      else if (frequency > 392+8) {
        digitalWrite(inTune,LOW);
        digitalWrite(highOrLow,HIGH);
      }
      else {
        digitalWrite(highOrLow,LOW);
        digitalWrite(inTune,HIGH);
      }
    }
    else {
      digitalWrite(highOrLow,HIGH);
      digitalWrite(inTune,HIGH);
    }
  }
  // Tuning string 2
  if (digitalRead(msdStringSig) == 0 && digitalRead(lsdStringSig) == 1) {
    if (frequency > 210 && frequency < 320) {
      if (frequency < 262-8) {
        digitalWrite(inTune,LOW);
        digitalWrite(highOrLow,LOW);
      }
      else if (frequency > 262+8) {
        digitalWrite(inTune,LOW);
        digitalWrite(highOrLow,HIGH);
      }
      else {
        digitalWrite(highOrLow,LOW);
        digitalWrite(inTune,HIGH);
      }
    }
    else {
      digitalWrite(highOrLow,HIGH);
      digitalWrite(inTune,HIGH);
    }
}
// Tuning string 3
if (digitalRead(msdStringSig) == 1 && digitalRead(lsdStringSig) == 0) {
  if (frequency > 290 && frequency < 380) {
    if (frequency < 330-8) {
      digitalWrite(inTune,LOW);
      digitalWrite(highOrLow,LOW);
    }
    else if (frequency > 330+8) {
      digitalWrite(inTune,LOW);
      digitalWrite(highOrLow,HIGH);
    }
    else {
      digitalWrite(highOrLow,LOW);
      digitalWrite(inTune,HIGH);
    }
  }
  else {
    digitalWrite(highOrLow,HIGH);
    digitalWrite(inTune,HIGH);
  }
}
// Tuning string 4
if (digitalRead(msdStringSig) == 1 && digitalRead(lsdStringSig) == 1) {
  if (frequency > 390 && frequency < 480) {
    if (frequency < 440-8) {
      digitalWrite(inTune,LOW);
      digitalWrite(highOrLow,LOW);
    }
    else if (frequency > 440+8) {
      digitalWrite(inTune,LOW);
      digitalWrite(highOrLow,HIGH);
    }
    else {
      digitalWrite(highOrLow,LOW);
      digitalWrite(inTune,HIGH);
    }
  }
  else {
    digitalWrite(highOrLow,HIGH);
    digitalWrite(inTune,HIGH);
  }
}
// Monitoring inputs and outputs
Serial.print("Frequency: ");
Serial.print(frequency);
Serial.print(", String: ");
Serial.print(digitalRead(msdStringSig));
```

```
      Serial.print(digitalRead(lsdStringSig));
      Serial.print(", highOrLow: ");
      Serial.print(digitalRead(highOrLow));
      Serial.print(", inTune: ");
      Serial.println(digitalRead(inTune));
    }
  }
}
```

III. PIC18F2550 Solenoid Chord Progression Routine

```
'****************************************************************
'*  Name    : ukuleleSolenoidPICSimple.BAS
'*  Author  : Taylor Piske, Tim Davison, and Arsenio Gonzales
'*  Date    : 3/19/2016
'*  Notes   : Written for a PIC18F2550.
'*          Waits for the master to signal a song choice and
'*          carries out the song routine.
'****************************************************************

' Define I/O pins for solenoids and Arduino communication
Gfret1 var porta.0
Gfret2 var porta.1
Gfret3 var porta.2
Cfret1 var porta.3
Cfret2 var porta.4
Cfret3 var porta.5
Efret1 var portb.3
Efret2 var portb.2
Efret3 var portb.1
Afret1 var portb.6
Afret2 var portb.5
Afret3 var portb.4

' Main loop
' Note that pause 500 correlates to about 0.2 seconds of pause
while(1)
   'high Cfret1
   'pause 500
   'low Cfret1
   'pause 500
   'high Gfret2
   'pause 500
   'low Gfret2
   'pause 500
   'high Cfret1
   'high Gfret2
   'pause 500
   'low Cfret1
   'low Gfret2
   'pause 500
   gosub CmajChord
   pause 7000
   gosub GmajChord
   pause 7000
   gosub release
   pause 7000
   gosub CmajChord
   pause 7000
   gosub release
   pause 10000
wend

AmajChord:
   LOW Gfret1
          high Gfret2
          LOW Gfret3
          high Cfret1
          low Cfret2
          LOW Cfret3
          low Efret1
          low Efret2
          low Efret3
          LOW Afret1
          low Afret2
          LOW Afret3
return

AminChord:
   LOW Gfret1
          high Gfret2
          LOW Gfret3
          LOW Cfret1
          low Cfret2
          LOW Cfret3
          low Efret1
          LOW Efret2
          low Efret3
          LOW Afret1
          low Afret2
          LOW Afret3
return

CmajChord:
   LOW Gfret1
          LOW Gfret2
          LOW Gfret3
          LOW Cfret1
          low Cfret2
          LOW Cfret3
          LOW Efret1
          LOW Efret2
```

```
            low Efret3
            LOW Afret1
            LOW Afret2
            high Afret3
return

DmajChord:
   LOW Gfret1
            high Gfret2
            LOW Gfret3
            LOW Cfret1
            high Cfret2
            LOW Cfret3
            LOW Efret1
            high Efret2
            low Efret3
            LOW Afret1
            LOW Afret2
            low Afret3
return

FmajChord:
   LOW Gfret1
            high Gfret2
            LOW Gfret3
            LOW Cfret1
            low Cfret2
            LOW Cfret3
            high Efret1
            LOW Efret2
            low Efret3
            LOW Afret1
            low Afret2
            LOW Afret3
return

GmajChord:
   LOW Gfret1
            LOW Gfret2
            LOW Gfret3
            LOW Cfret1
            high Cfret2
            LOW Cfret3
            LOW Efret1
            LOW Efret2
            high Efret3
            LOW Afret1
            high Afret2
            LOW Afret3
return

Gmin7Chord:
   LOW Gfret1
            LOW Gfret2
            LOW Gfret3
            LOW Cfret1
            high Cfret2
            LOW Cfret3
            high Efret1
            LOW Efret2
            low Efret3
            LOW Afret1
            high Afret2
            LOW Afret3
return

release:
   LOW Gfret1
            LOW Gfret2
            LOW Gfret3
            LOW Cfret1
            LOW Cfret2
            LOW Cfret3
            LOW Efret1
            LOW Efret2
            LOW Efret3
            LOW Afret1
            LOW Afret2
            LOW Afret3
return
```