Bhimsen Budhathoki

Design of an Electric Guitar Tuner

Metropolia University of Applied Sciences

Bachelor of Engineering

Degree Programme

Thesis

21 September 2018

Author Title Number of Pages Date	Bhimsen Budhathoki Design of an Electric Guitar Tuner 20 pages + 1 appendices 18 September 2018
Degree	Bachelor of Engineering
Degree Programme	Electronics
Professional Major	-
Instructors	Kai Lindgren, Senior Lecturer

The objective of the thesis was to design an electric guitar tuner. The other purpose was to investigate a frequency detection method other than the common pitch detection techniques such as Zero-Crossings, Peak Detection and Slope Detection. Those techniques are usually fast and efficient but do not work well in a noisy environment.

Arduino uno was used in the design. Audio signal from the guitar after amplification is fed in to the Arduino. It, then, detects the frequency and guides the user to tune the string using an LCD screen. Autocorrelation method proved to be an efficient method for our purpose. This method also proved to be comparatively comprehensible and easy to implement on Arduino uno.

The resulting design fulfils the set objectives. It presents a comprehensible algorithm to calculate the frequency of an incoming signal and guides the user to tune the guitar string.

Keywords	DSP, Autocorrelation, Arduino, Guitar Tuner

Contents

Appendix 1. Source code of the tuner

List of Abbreviations

FFT	Fast Fourier Transform
USB	Universal Serial Bus
SPI	Serial Peripheral Interface
TWI	Two Wire Interface
I2C	Inter-Integrated Circuit
UART	Universal Asynchronous Receiver/Transmitter
PWM	Pulse Width Modulation
DSP	Digital Signal Processing

1 Introduction

Music is a form of art. Its medium is sound organised in time. Pitch, rhythm and dynamics are some elements of music. Vast range of vocal techniques and musical instruments are used to perform music. Music has been an integral part of human society since time immemorial. It plays a key role in religious rituals, cultural activities and social activities. People may make music as a hobby or work as a professional musician. The music industry consists of individuals who create music, individuals who perform music and individuals who record music.

Musical instruments are the most crucial element of music and to play beautiful music, a perfectly tuned musical instrument is a must. For an untrained ear, it is almost impossible to tune a musical instrument without the aid of a tuning equipment. This thesis focuses on designing an electric guitar tuner.

A musical note is the pitch and duration of the sound. Frequency may be referred to as quantitative measure of pitch. Pitch and frequency are related but these two cannot be used synonymously. A guitar note is a complex waveform consisting not just of the fundamental frequency but also of several harmonics. It is the harmonics that make the same note sound different on various stringed instruments.

2 Guitar

2.1 Introduction

The guitar is one of the most popular stringed musical instruments. It is comprised of a neck, which has a fretted fingerboard and a body. A normal guitar has six strings. The strings are strung along the neck to the bridge on the guitar body. Both hands are required to be used to play a guitar. One hand is used for strumming or plucking the strings, while simultaneously, the other hand is used for fretting.

All the strings have certain thickness. Fretting onto the different position on the fretboard changes the length as well the tension of the string. This alters the note being played. As the strings of the guitar are of a certain length and tension, when played, each of the strings corresponds to a certain note and thus can be tuned to a certain note. Each musical note corresponds to a certain frequency of sound.

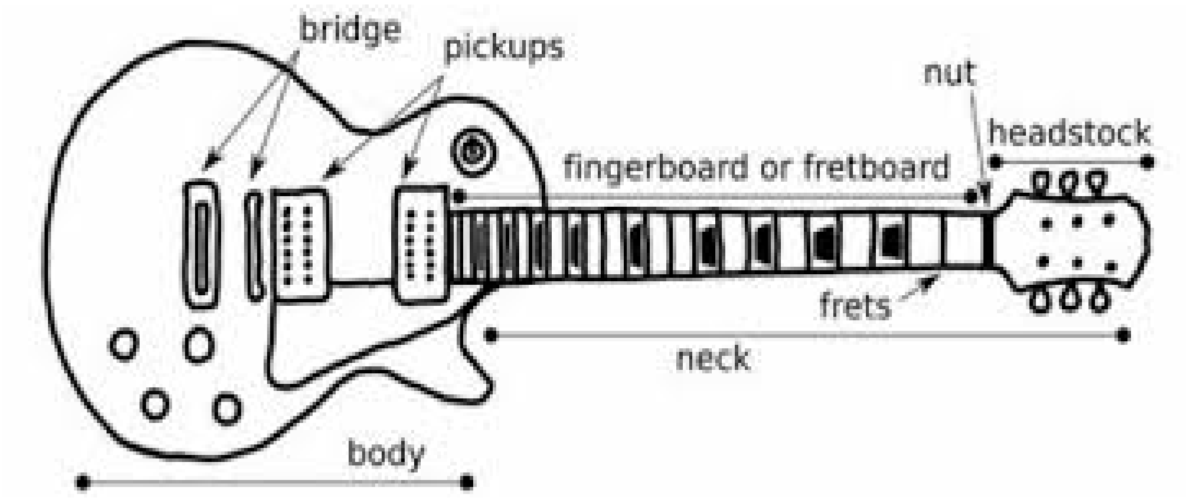


Figure 1. Parts of an Electric Guitar

Figure 1 shows parts of an electric guitar. A musical scale is any set of musical notes arranged by pitch. Traditional western music is based on 12 notes, denoted A, A#, B, C, C#, D, D#, E, F, F#, G and G#. The set of these 12 notes make an octave.E, B, G, D, A and E are respectively the guitar strings going upward from the bottom.

Table 1. Fundamental frequencies of the six guitar strings

String	Frequency (in Hz)
E (1st)	329.6
B (2nd)	246.9
G (3rd)	196.0
D (4th)	146.8
A (5th)	110.6
E (6th)	82.4

Table 1 shows the fundamental frequencies of the six guitar strings. The guitar is said to be out of tune, when the frequency of the string does not correspond to that of the expected note. A basic guitar tuner detects the pitch of the notes played and gives the visual feedback of how far way the note being played is away than the standard tuning. Along with the fundamental frequency, a guitar note is made up of many harmonics as well. The difference in the harmonics is what makes the same note sound different on different stringed instruments.

2.2 The electric guitar

An acoustic guitar makes sound by vibration alone. When a string is vibrated, it transmits sound energy into the hollow wooden body of the guitar, which resonates and amplifies the sound.

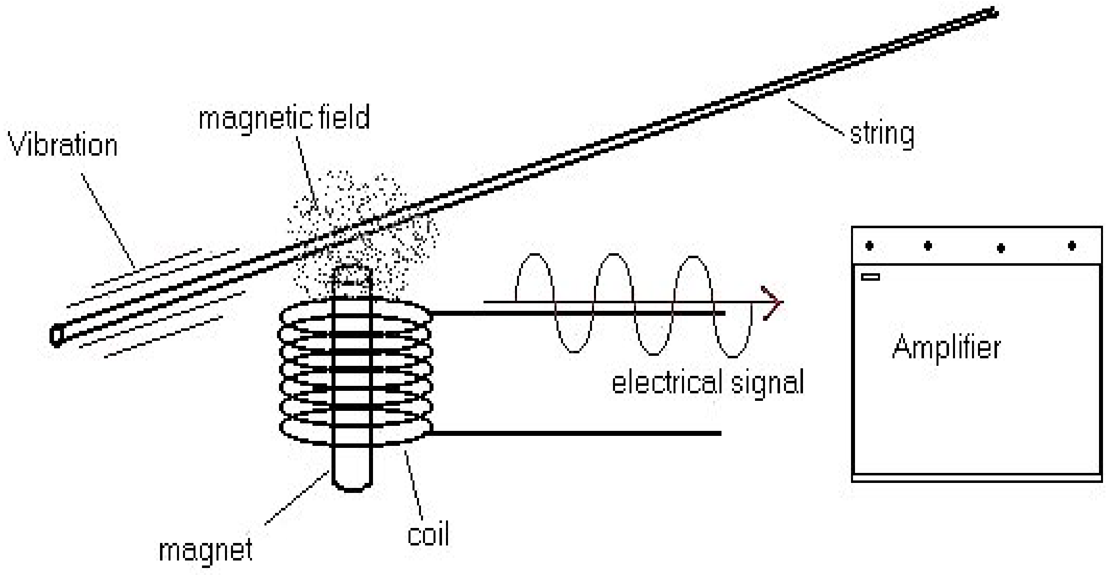


Figure 2. Working of an electric guitar pick-up

Figure 2 shows the working of an electric guitar pick up. Faraday’s law of electromagnetic induction explains the working of an electric guitar. The vibrations of the strings are converted into electrical signals by the electric guitar. The electro-magnetic pickups mounted under the strings on the guitar's body can generate electricity. These pickups are basically one or more bar magnets wrapped thousands of turns of fine wire. The magnetic field generated by these magnets passes up through the strings as well. When the string of an electric guitar, made of metal, is plucked, its vibration in the pickup’s magnetic field induces a current in the wire of the pickup coil. The electrical signal generated is miniscule so it needs some amplification. Therefore, this electrical signal is fed into the amplifier and the amplified signal is sent to the speaker which converts it into audible sound. The amplifier and loudspeaker are built into a single unit basically referred to as an "amp."

3 Frequency or pitch detection theory

3.1 Introduction

Pitch is the human perception of sound. It is that quality of sound which makes it possible to judge sound as “higher” and “lower”. And frequency on the other hand is a measurable scientific attribute. Often pitch is considered the same as frequency but that is not the case. In fact, pitch does not even have a linear or even logarithmic relationship with frequency. Pitch and fundamental frequency of sound are considered equivalent for the sake of simplicity for this project. (Openlearn, 2018)

The primary objective of this thesis is concerned with the pitch detection of guitar sound. Guitar music corresponds to a quasiperiodic or oscillating signal. Therefore, its pitch detection can be done in the time domain, frequency domain, or both. The frequency of a wave is the inverse of its time period. So, most pitch detection algorithms first calculate the time period of the signal than inverse the value to calculate the fundamental frequency. (Gdańsk University of Technology, 2007)

Pitch detection methods or algorithm can be grouped into several categories depending upon the domain they operate in. It is either in time domain or frequency domain or the combination of both. Of all these algorithms, the ones that work in the time domain are of most importance to us. And among these algorithms, zero crossing and autocorrelation methods have been used most commonly for pitch detection of musical instruments.

3.2 Zero Crossing

In electronics, the zero crossing of an alternating current is the point at which the voltage is zero. Zero crossing is a very popular pitch detection technique in use.

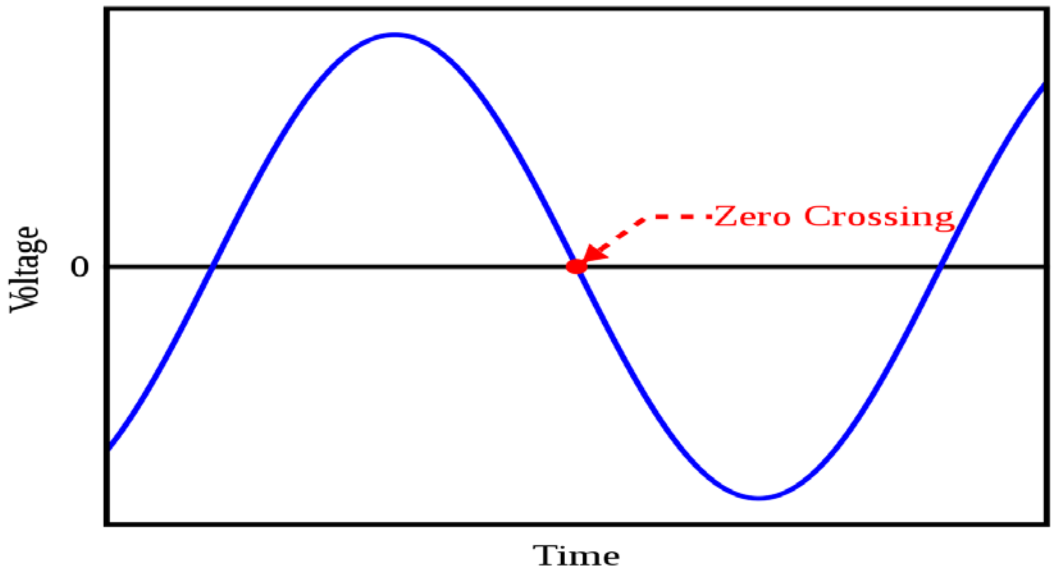


Figure 3. Zero crossing of a waveform

Figure 3 shows zero crossing of a sinusoidal waveform. During one whole cycle of a simple waveform, zero crossing occurs twice. Thus, counting the number of zero crossings in a waveform gives the period of the waveform which in turn can be used to calculate the fundamental frequency of the waveform. Where speech processing is concerned, counting the number of zero crossings is a common method to estimate the fundamental frequency. Since this is a time domain feature detection method, the signal is usually processed beforehand to accentuate the time domain feature. Zero crossing is a very simple technique. It is comparatively faster and inexpensive to implement as well. But it is not so accurate. Furthermore, in a noisy environment where the fundamentals can be weaker than the partials, this method proves to ineffective.

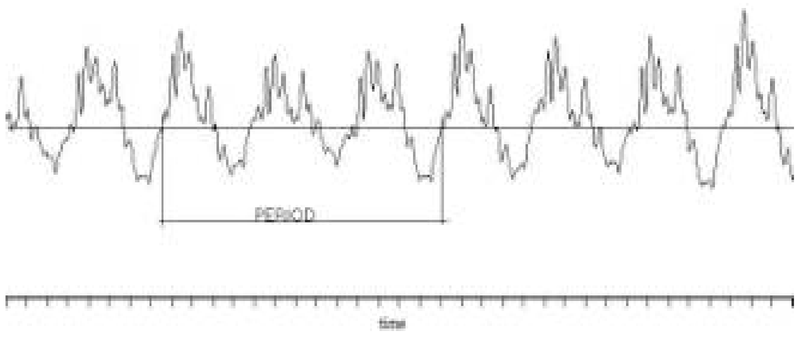


Figure 4. Harmonic signal in time domain

Figure 4 shows that zero crossing is not so effective in detecting the period when a harmonic signal is being analysed. Also, the diagram above illustrates that this method does not give clear results when harmonic signal is being analysed. The pre-processing of the signal using filters may increase the accuracy of the technique. **(Stanford University, 2012)**

3.3 Autocorrelation

Association, in statistics, is the relationship between two random variables whereas correlation is a statistical association. Correlation gives an idea of how close two variables are from each other from having a linear relationship.

In digital signal processing, autocorrelation is the correlation of a signal with delayed copy of itself as a function of delay. Informally, it is the similarity between observations as a function of lag between them. Autocorrelation is like a search procedure in frequency detection. The signal is stepped through sample by sample and a correlation between reference window and the lagged window is performed. The correlation at "lag 0" will be the global maximum because the reference is being compared to the exact copy of the signal, at some point it will begin to increase again, then reach a local maximum again. The pitch is then estimated from the distance between "lag 0" and that first peak.

Autocorrelation is a very powerful tool in signal processing. It comes very handy in finding repeating patterns such as a periodic signal interfered by noise, fundamental frequency hidden in the harmonics etc. Therefore, it is the perfect pitch detection algorithm for guitar sound. (Wikipedia, 2018)

3.3.1 Mathematics of Autocorrelation

Suppose 'x_t' is discrete time signal. Then autocorrelation function can be defined as:

$$r_t(\tau) = \sum_{j=t+1}^{t+W}(x_j x_j + \tau) \qquad (1)$$

where, $r_t(\tau)$ is the autocorrelation function of lag τ calculated at the time index t and W is the integration window size. (Cheveigne´, 2002)

4 The Design

4.1 Block Diagram Representation

The block diagram representation of our electric guitar tuner is illustrated below.

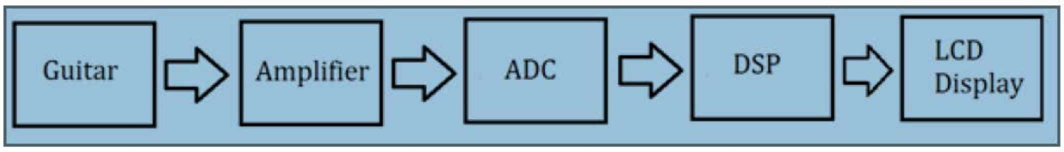


Figure 5. Block diagram of the design

Figure 5 shows the block diagram of the design. The design takes input signal from the guitar and the output it generates guides the user in tuning the guitar. Various stages of the design are described below.

4.1.1 Amplifier stage

The incoming signal from the guitar is too small to be processed by our pitch detection algorithm. So, it is amplified at this stage. An audio amplifier is designed using an operational amplifier for this purpose. It amplifies the signals coming from the guitar pickups.

4.1.2 Analog to Digital Converter

At this stage the incoming analog signal is converted into digital signal. Working with digital signals is much easier and gives much accurate result as compared to analog signal. The digital signal converted must have a suitable resolution and sampling frequency for further processing.

4.1.3 Digital Signal Processor

At this stage the pitch detection algorithm of our choice i.e. the autocorrelation method is used to detect the pitch or frequency of the incoming signal.

4.1.4 LCD Display

This is the final stage of our design. This is where the output of the design is produced as well. It gives the user a visual feedback of whether the guitar is in tune or not. If the guitar is not in tune it also guides the user in tuning it.

5 Components used

5.1 Operational Amplifier

LM358 was the choice of operational amplifier for designing the audio amplifier. The low harmonic distortion and low noise make the LM358 ideally suited for audio preamplifier applications. It also operates on a single power supply making it an ideal choice as compared to majority other Op-Amps which operate on a dual power supply.

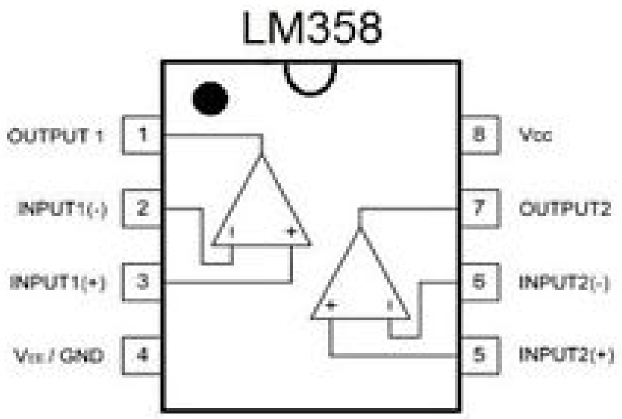


Figure 6. Illustrative pin diagram of LM358

Figure 6 shows the pin diagram of LM358. Only one part of the LM358 was used in our design.

5.2 Signal Amplifier Circuit

The sound output from an electric guitar corresponds to very low peak to peak voltage (approximately 400mV). It also varies in the zero reference because Arduino’s analog input only reads value between 0 and 5 Voltage. Therefore, we must amplify the audio signal so that it has 2.5 V peak to peak value and provide DC offset to the signal's reference so that it falls within the 0-5V range. That is where the signal amplifier circuit comes in to play.

Following non-inverting amplifier configuration was used in the design.

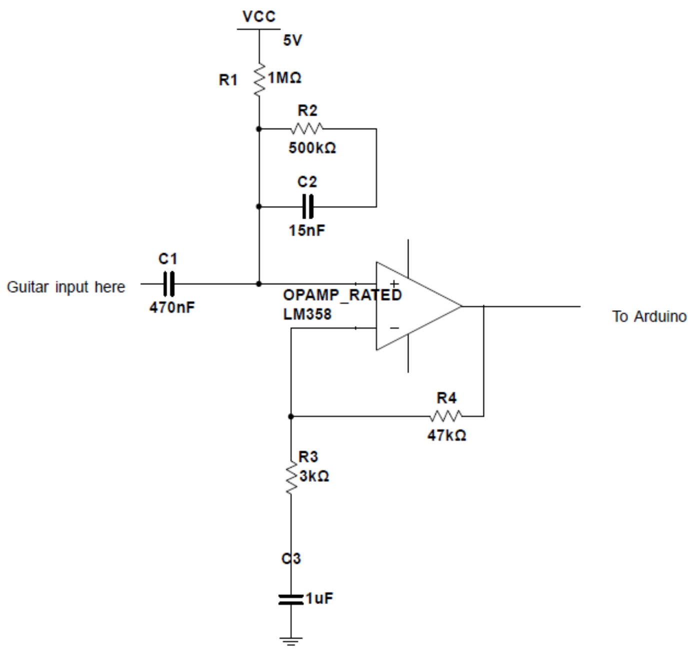


Figure 7. Schematic of the signal amplifier

Figure 7 shows the schematic of the signal amplifier. LM358 operates of a single 5V power supply. Non-inverting configuration of LM358 was used in the design. The op-amp has a AC gain of 16 and DC gain of 1. The capacitor at the input acts as a low pass filter. It only allows signal of frequency greater than 1 Hz.

5.3 Microcontroller

The analog to digital conversion part as well the digital signal processing part takes place within the microcontroller. It is, therefore, the core of the project. Arduino uno was the first choice of microcontroller for this project.

5.3.1 Arduino Development Platform

Arduino is a programmable electronic platform based on open source hardware and software. The ease of use and flexibility of the Arduino hardware and software enables developers and enthusiasts to create interactive devices.

The team, that developed Arduino, was formed in 2005 in Italy. The team members included Hernando Barragan, Massimo Banzi, David Cuartielles, Dave Mellis, Gianluca Marino and Nicholas Zambetti. Their goal was to create simple, low cost electronics development platform to design digital systems for non-engineers. The result was Arduino, whose features included a programming environment based on the Processing Programming Language, USB Connection for use and programming, and low price.

Arduino is a tool for making small computers that are expanded with switches, sensors, servers, or other similar physical devices. From these devices, Arduino receives or controls information with the microcontroller. Arduino platforms can work independently as embedded systems or communicate with another computer software. Arduino's operations are based on ATmega8 and ATmega168 microcontrollers, which are published under the Creative Commons license. This license allows designers to customize and expand Arduino's microcontrollers.

ATmega microcontrollers belong to the Atmel AVR product family, which includes a wide range of 8-bit microcontrollers.

Figure 8. ATmega8 microcontroller

Figure 8 shows an ATmega8 microcontroller. Circuit boards used by Arduino platforms can be manually assembled according to ready wiring diagrams or purchased pre-assembled. The openness of the Arduino project has led to the creation of similar platforms based on Arduino. However, Arduino's developers have stressed that Arduino should only be used for official products.

5.3.2 Arduino uno R3

The Arduino Uno R3 is the third development version of the Arduino Uno circuit board. It is the latest Arduino USB bus, and is also a reference model for other Arduino platform products. The physical size of the printed circuit board is 6.9 cm long and 5.3 cm wide.

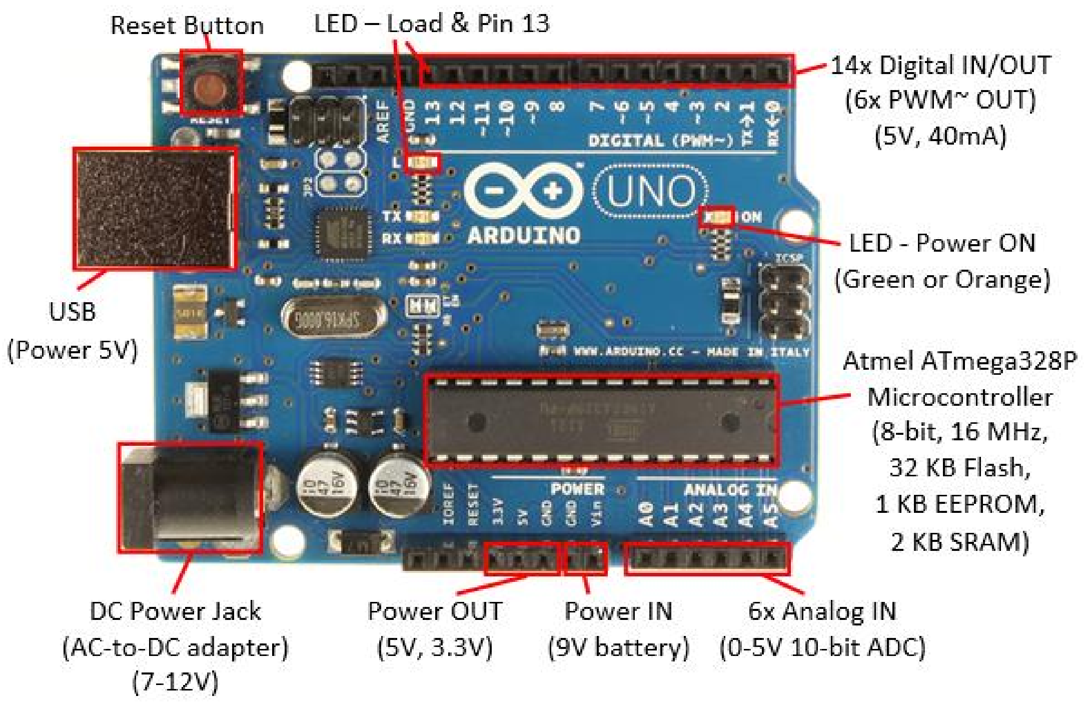


Figure 9. Illustrative diagram of the Arduino uno board

Figure 9 shows an Arduino uno board. The Arduino Uno R3 is different from the previous Uno board. Instead of the FTDI USB serial interface driver circuit, it uses an ATmega16U2 microprocessor programmed as a USB serial connection converter. In addition, new pins have been added to the R3 and the reset circuit has been changed as well.

The major features of the Arduino Uno R3 are as follows:

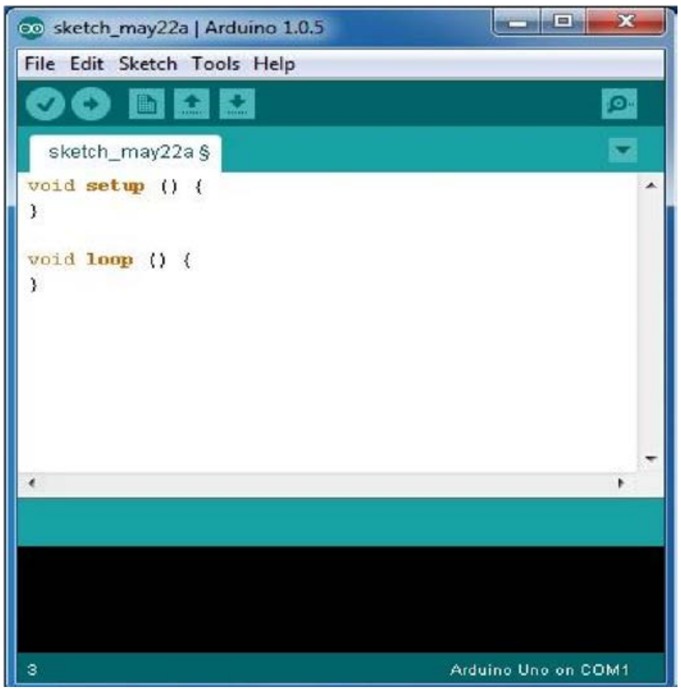
- ATmega328 microcontroller
- 5V supply voltage
- Recommended supply voltage: 7-12V
- 14 digital I / O pins, 6 of which are PWM-controlled outputs
- 6 analog inputs
- single I / O pins DC: 40mA
- 3.3V DC current: 50mA
- 32 kB of flash memory
- 2 kB of SRAM memory
- 1 kB programmable EEPROM memory
- 16 MHz clock frequency (Arduino Uno.)

The Arduino Uno R3 circuit board is powered by a USB bus, external power supply, or batteries. The voltage sources used are VIN, 5V, 3.3V, GND and IOREF. A VIN pin can supply voltage to a circuit board if a separate power supply is used. GND is a grounding pin. The 5V and 3.3V pins give the corresponding voltages to their output according to their name. The IOREF pin is used with some shield attachments, so the shield can read the voltages used by the IOREF pin and send them forward. [1]

Each 5V digital I / O pin can be used either as input or output. They can also be controlled by a microcontroller using pinMode, digitalWrite and digitalRead programming commands. In addition, some of the pins have special functions. Pins 0 and 1 are used to receive and send serial data. These pins are connected to the corresponding ends of the ATmega16U2 USB serial interface. Pins 2 and 3 can be used to create an interrupt signal. The pins 3, 5, 6, 9, 10 and 11 function as 8-bit PWM outputs controlled by an analogWrite programming command. In the pins 10, 11, 12 and 13, SPI connection can be used. The pin 13 also has a built-in LED light. The Arduino Uno R3 analog pins operate at a 10-bit resolution of 05 V. Analog pins A4 and A5 support TWI connection. In addition, there are still AREF and Reset pins on the circuit board. The AREF surface can be used to change the upper limit of the voltage of all analog pins. The reset pin name sets the circuit board start state, but the pin itself is only used when the Reset button is not possible to press.

5.3.3 Arduino complier

Arduino IDE is a programming environment for writing a code that the Arduino Microcontroller performs. The program can be downloaded for free and runs on Windows, Macintosh OSX and Linux. The latest version of the program at the writing time is 1.8.7.



```
void setup () {
}

void loop () {
}
```

Figure 10. Arduino IDE window

Figure 10 shows the Arduino IDE windows. The Arduino IDE is based on the Processing Programming Language and comes with a Wiring Software Library that facilitates general operations. The code that Arduino understands is written in the C ++ language. The whole program code is called Sketch.

The programming environment's code editor features auto-indentation, code coloring, and syntax highlighting. Additionally, sending a code to the Arduino PCB is done with one touch of a button. The programming environment automatically makes the necessary changes to the code by adding the top header and ending the main function. Following, the program code is reversed and finally transmitted via USB to the circuit board. Because of its simplicity, the Arduino IDE has some shortcomings, especially in detecting errors. The programming environment does not tell the user about incomplete commands or error states. Code simulation is also unavailable.

Arduino's program code consists of two main functions. The setup is run when the program starts. It initializes the required variables, pins, and program libraries. The loop is run after the setup and is repeated until the Arduino power is switched off. The function of the loop is to control the circuit board and the connected devices**. (Arduino, 2018)**

5.4 Output Display

After the microprocessor runs our pitch detection algorithm, the fundamental frequency is calculated. Then a display is used to guide the user in tuning the guitar. Funduino LCM1602 LCD module was used for our design.

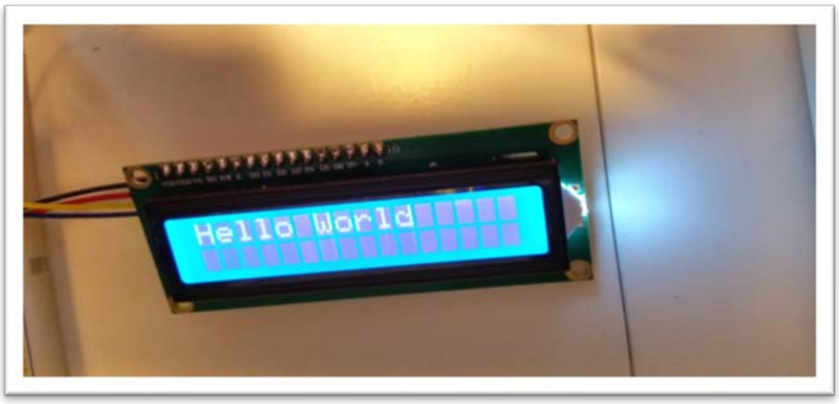


Figure 11. Funduino LCD module

Figure 11 shows the funduino LCD module. It is available for a very cheap price and the wiring is really simplified. It usually comes together with Arduino learning kits widely available online.

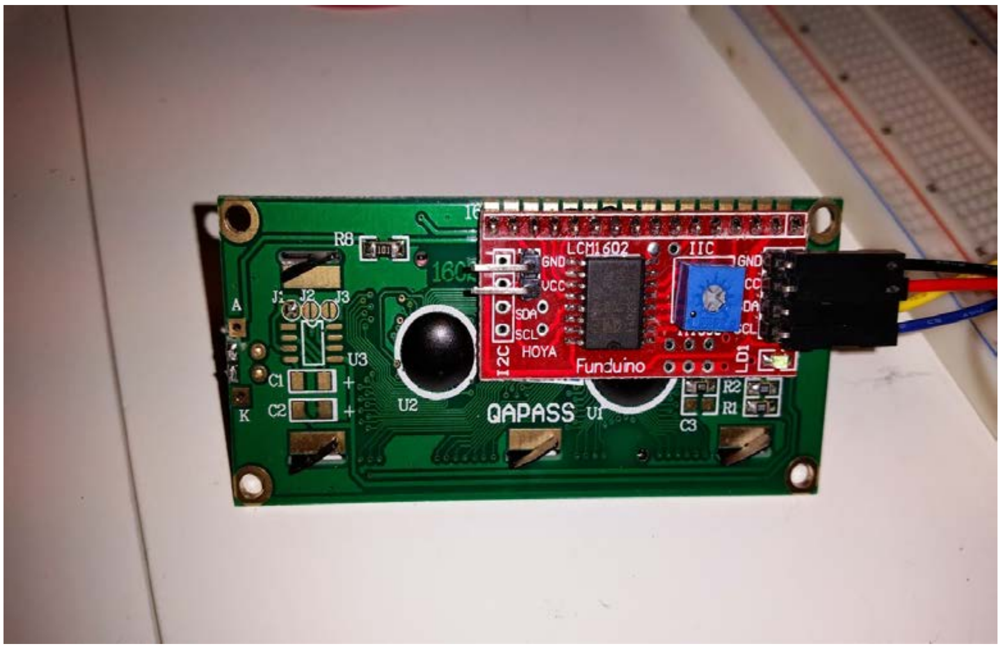


Figure 12. Back side of the funduino module

Figure 12 shows the back side of the funduino module. The usual Standardcrystal library does not work with the funduino module. After some research LiquidCrystal_I2C library was found which works with this module and has the same parameters.

6 Conclusion

The resulting design fulfills the set objective. The autocorrelation algorithm computes the frequency by detecting the relative level of the fundamental with respect to the other frequencies. This ensures accurate result even at very high noise levels as the only requirement is that the maximum energy must be carried by the fundamental.

The goal of this project was also to implement the design that recognizes the signal coming from the guitar, analyses it and then guides the user in tuning the guitar. The construction of the prototype did not go as expected. The design was first implemented on a breadboard using all the components as mentioned in the design. The prototype did not work as expected.

When troubleshooted, it was found that the amplifier circuit was not working well. An alternative audio amplifier circuit was also tested using the TL082 Operational Amplifier, the problem persisted.

With some time and dedication, the design can produce a powerful guitar tuner. The algorithm presented is fast, works well in noisy environment and easy to implement on microcontrollers. Further improvements on the design could be the use of a servo motor to rotate the tuners or even implementing the algorithm on a smartphone application which controls the servo motors to tune the guitar with wireless technologies like Bluetooth.

7 References

1. Arduino. (September 2018). *Introduction to Arduino*. Accessed 14 March Arduino: https://www.arduino.cc/en/Guide/Introduction

2. Cheveigne´, K. (2002). YIN, a fundamental frequency estimator for speech and music. 2.

3. Gdańsk University of Technology. (2007). *Pitch Detection Methods*. Accessed 17 April https://sound.eti.pg.gda.pl/student/eim/synteza/leszczyna/index_ang.htm

4. Openlearn. (23. 02 2018). The Perception of Frequency.

5. Stanford University. (2012). *Pitch Detection Method Review*. Accessed 17 May https://ccrma.stanford.edu/~pdelac/154/m154paper.htm

6. Wikipedia. (2018). *Autocorrelation*. Accessed 17 June Wikipedia: https://en.wikipedia.org/wiki/Autocorrelation

Source code of the tuner

//Code is based on reliable frequency detection techniques by akkeliryl on instructables//

```
#define LENGTH 512 //Maximum size of input data declared to be 512 bytes
byte rawData[LENGTH]; //rawData array is defined as byte. A byte stores 8 bit
unsigned integer, from 0 to 255
int count; // variable count is declared as an integer
char noteName; // variable noteName is defined as character
const float sample_freq = 8919; //sample_freq is defined as a float of con-
stant value
int len = sizeof(rawData); //size of rawData array
int i,k;
long sum, sum_old; //variable sum is declared. sum is the value of each auto-
correlation calculation
int thresh = 0;
float freq_per = 0;
byte pd_state = 0;
void setup(){
  analogReference(EXTERNAL); // Analog reference of Arduino is 3.3V
  analogRead(A0); // Pin A0 reads the amplified guitar signal
  Serial.begin(115200); // Sets the data rate in bps for serial transmission
  count = 0;
}


void loop()
{
  if (count < LENGTH)
 {
    count++;
    rawData[count] = analogRead(A0)>>2; //Input signal data is stored in raw-
Data array, of values more than 2
  }
  else {
    sum = 0;
    pd_state = 0;
    int period = 0;
    for(i=0; i < len; i++) //value of i is increased until it equals the value
of LENGTH
    {
      // Arduino code for Autocorrelation
      sum_old = sum;
      sum = 0;
      for(k=0; k < len-i; k++) sum += (rawData[k]-128)*(rawData[k+i]-128)/256;

// The raw data array has 8 bit unsigned values so we subtract 128 from it, so
we get signed values//

       Serial.println(sum);

      // Peak Detect State Machine (courtesy of akkeliryl)//

// The state machine moves from one state to another//
//Example STATE 0 thresh, the threshold under which we ignore the data
NEWSTATE is 1. Then we look for threshold above thresh and where slope of the
signal is positive. NEWSTATE is 2. Then we look for the slope of the signal to
be negative or zero. Then the peak is determined //

      if (pd_state == 2 && (sum-sum_old) <=0)
      {
        period = i;
        pd_state = 3;
      }
      if (pd_state == 1 && (sum > thresh) && (sum-sum_old) > 0) pd_state = 2;
      if (!i)
       {
        thresh = sum * 0.5;
        pd_state = 1;
      }
    }
    for(i=0; i < len; i++)
Serial.println(rawData[i]);

    // Frequency identified in Hz
    if (thresh >100) {
      freq_per = sample_freq/period;
      Serial.println(freq_per); // prints calculated frequency on serial
screen of IDE
    }
    count = 0;
  }
}

//Filters out too high frequencies
 }
else
{ freq = -1;
 }

}
count = 0;
 Serial.println(freq); // prints the frequency on serial screen on IDE
 displayToLCD(freq); // displays frequency on LCD
 delay(400); // creates 400 ms delay
}
}

void displayToLCD(float freq)
{ if(freq == -1) { return; }

if(freq >= 15.89){ // checks if the frequency is above minimum C note fre-
quency
octave_counter = -1;
Ffreq=getFfreq(freq);


//Categorizes frequency values to corresponding note range

if((15.89<=Ffreq) & (Ffreq<=17.34)){ Note = C; noteName = ‘C’; }
 else if((17.35<=Ffreq) & (Ffreq<19.475)){ Note = D; noteName = ‘D’; }
 else if((19.475<=Ffreq) & (Ffreq<21.215)){ Note = E; noteName = ‘E’; }
 else if((21.215<=Ffreq) & (Ffreq<23.185)){ Note = F; noteName = ‘F’; }
 else if((23.185<=Ffreq) & (Ffreq<26.00)){ Note = G; noteName = ‘G’; }
 else if((26.00<=Ffreq) & (Ffreq<29.185)){ Note = A; noteName = ‘A’; }
 else if((29.185<=Ffreq) & (Ffreq<31.785)){ Note = B; noteName = ‘B’; }

float closeness0 = (Ffreq/Note);
int cl1 = 0;
cl1 = int((closeness0-1)*100); // round to nearest whole number for accuracy

//means the string is in tune//
if(Ffreq==Note){out = “b—-[[ ]]—-#”;}
 else if(cl1==-1) out = “b—-[[ ]]—-#”;
 else if(cl1==1) out = “b—-[[ ]]—-#”;
```

```
//means the string is out tune//
else if(cl1==-2) out = “b–<<< >––#”;
 else if(cl1==2) out = “b––< >>>–#”;
 else if(cl1==-3) out = “b-<<<< >––#”;
 else if(cl1==3) out = “b––< >>>>-#”;
 else if(cl1==-4) out = “b-<<<<< >––#”;
 else if(cl1==4) out = “b––< >>>>>-#”;
 else if(cl1==-5) out = “b<<<<<< >––#”;
 else if(cl1==5) out = “b––< >>>>>>#”;
}
 else{ Ffreq = -1; }

 // -1 one for too low //

 count++;


// Output text is positioned on the LCD display
lcd.setCursor(3,0);
lcd.print(freq);
lcd.setCursor(11,0);
lcd.print(“Hz”);
lcd.setCursor(0,1);
lcd.print(out);
lcd.setCursor(7,1);
lcd.print(noteName);
lcd.setCursor(8,1);
lcd.print(octave_counter);
 }

float getFfreq(float freq)
{ octave_counter++;
 if(freq > B){ return getFfreq(freq/2);
}
 else return freq;
 }

}
```