ELECTRIC AIR UKULELE

By

Ivan Setiawan

Satyo Iswara

Final Report for ECE 445, Senior Design, Spring 2012

TA: Jane Tu

28th April 2012

Project No. 32

Abstract

The purpose of this report is to describe the design and the verifications of the electric air ukulele project. The purpose of this project is to build a wireless device that senses user's left hand movements and distances to play notes in ukulele. This paper will first introduce the project then describe the design for each module and as a whole. The design verification will then be described. Finally, the cost will be evaluated and the success and possible future work of the project will be discussed.

Contents

1. Introduction

The motivation behind this project is to design and build an electric air musical instrument where the user can play music instruments with glove on the right hand and a portable and lightweight box on the left hand. This design can replace a musical instrument, such as a ukulele or an electric guitar, and can be cheaper than the actual ukulele or electric guitar.

1.1 Purpose

The purpose of this project is to design an electric air ukulele where the sound can be generated based on the hands' positions. This design includes microcontroller, proximity sensor, fret potentiometer, and the RF transmitter on the left hand. On the right hand, there will be a glove with flex sensors in each finger, except the thumb, another microcontroller, and RF transmitter. The sound can be generated depending on the distance between the left and the right hands. On the left hand, the potentiometers have functionalities like frets on the real ukulele. Then, depending on the distance between the left and the right hands, the input will be processed through the microcontroller. Then, those data will be sent into the right hand with the transmitter. The right hand will receive and send those data into the microcontroller. On the right hand, the flex sensors have functionalities like strings in the real ukulele. The inputs from the flex sensors will be sent into the microcontroller on the right hand to for further processing. Finally, the speaker is connected into the microcontroller so that the sound can be generated.

Benefits of this product to the customer include:

- Simplification of music playing since the frets are determined by the distance between the left and right hands
- Compact and lightweight
- Pre-calibrated strings and that do not need adjustments

1.2 Project Function

The specifications for the design as a whole are given below. Individual modules have their own verification requirements and will be explained in detail in later section.

- The proximity sensor can sense the distance of the right hand approximately 15 – 45 cm away
- The RF can still transmit and receive data correctly up to 3 feet away
- No information loss. The microcontroller on the right hand will receive the exact data from the left hand microcontroller via RF transceiver
- The power consumption is low enough so that both the left and right hands device can be powered up using 9 volts battery each
- The delay time from a user input to the RF transceiver sending a signal to the microcontroller to generate sounds should be less than 1 second
- The device can recognize which strings is being picked based on which finger is bent on the right hand, the distance between the left and right hands, and the potentiometer at the left hand

- The sound generated by a speaker will mimic the sound of the real ukulele strings at particular frequencies

1.3 Block Diagram and Description

In this section we would present our block diagram and brief description of each block.

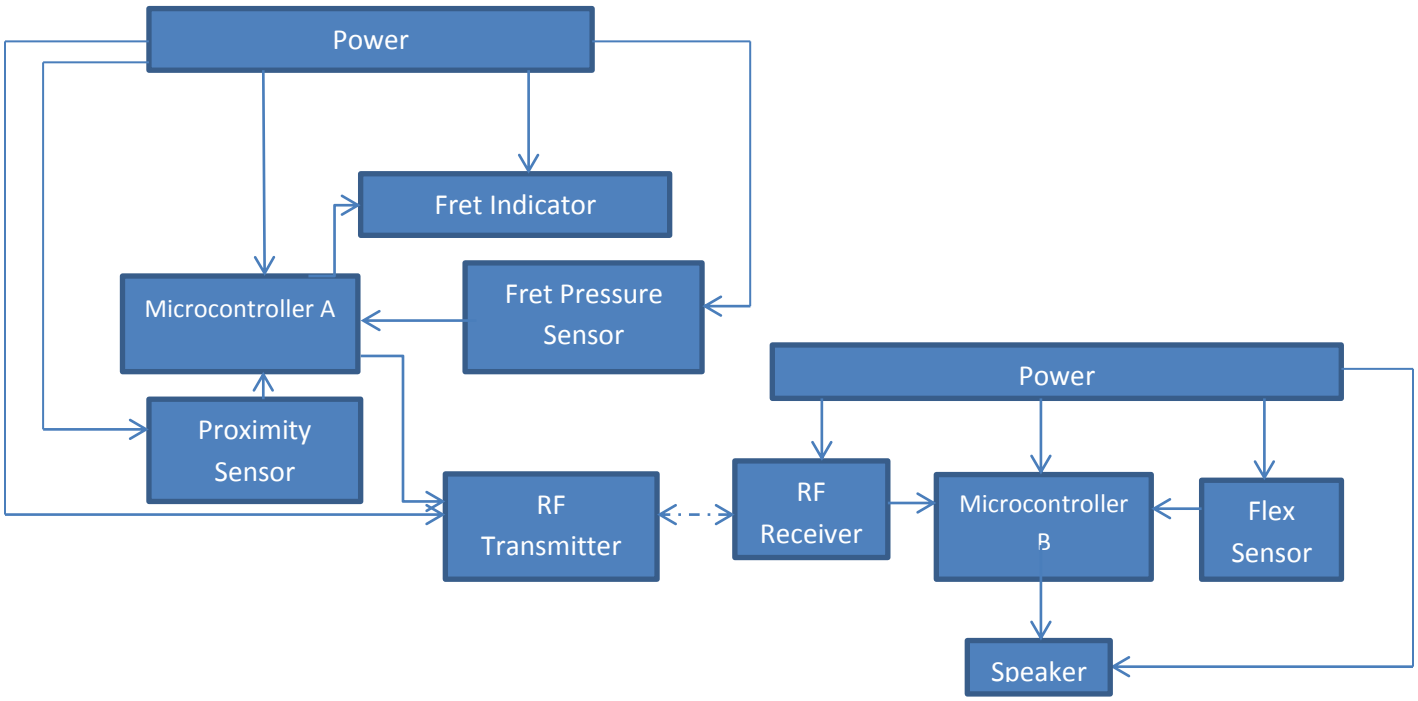


Figure 1.3.1 Block Diagram of left hand and the right hand

1.3.1 Left Hand Module

1.3.1.1 Microcontroller A

Arduino Uno is used for the microcontroller in this project. The microcontroller would be the overall control center of the left hand module. The microcontroller is responsible for taking the input data from the fret pressure sensor and the proximity sensor. The notes playing depending on the distance between the two hands and the fret pressed are hard coded inside the Arduino Uno platform. This microcontroller is also responsible for sending serial digital data into the right hand via wireless connection using RF transmitter.

1.3.1.2 Fret Pressure Sensor

The fret pressure sensors consist of four potentiometers. These potentiometers will detect which chords are being pressed, and the chords pressed will be sent into the microcontroller.

1.3.1.3 Proximity Sensor

The proximity sensor is used to detect the distance between the left and the right hands. Depending on the distance between the hands, different notes are played.

1.3.1.4 RF Transmitter

The RF transmitter has frequency of 434 MHz. This transmitter will be connected into the microcontroller, which sends serial digital data. Using the VirtualWire library that has been provided by Arduino Uno microcontroller, the data input will be sent to the right hand receiver for further processing.

1.3.1.5 Power Regulator

Nine volts battery is used to power up this module in the left and right hand design. The nine volts power supply can be directly connected into the microcontroller. For other parts, voltage regulator is needed since potentiometer, RF transmitter, and proximity sensor only require at most 5 volts supply.

1.3.2 Right Hand Module

1.3.2.1 Microcontroller B

For the right hand, Arduino Uno is once again used for the microcontroller. The microcontroller is the most important part for the right hand module. It is responsible for detecting the input from the flex sensors to determine which strings are being picked. The microcontroller is also responsible for receiving the serial digital data transmitted from the left hand via RF receiver. Then, after receiving the data, we use synthesis algorithm to mimic the sound of the real string. The speaker will be connected directly to the microcontroller to generate the sounds.

1.3.2.2 Flex Sensors

Four flex sensors are implemented on each finger, except for the thumb, which functions as a string in the ukulele instrument. Each flex sensor represents a string of real ukulele instrument, the string C represented by the first flex sensor, the string G by the second flex sensor and so forth. By bending a finger (thus bending a flex sensor), a string picking motion can be simulated.

1.3.2.3 RF Receiver

The RF receiver with 434 Mhz frequency is used to receive the digital serial data from the transmitter using VirtualWire library in Arduino platform.

1.3.2.4 Speaker

The speaker with internal amplifier is used. The PWM signal from microcontroller B is passed through low pass filter and blocking capacitor to generate analog sinusoidal wave. The blocking capacitor is needed to cut off unwanted dc offset which is 2.5 V.

2. Design

Before designing the air ukulele, we need to know the physical dimension of ukulele. There are four types of ukulele, but for this project we want to design the most common ukulele which is soprano ukulele. The tuning for soprano ukulele is G4-C4-E4-A4 with G4 at the top string. The dimension of typical ukulele is 53 cm for total length and 33 cm for scale length.

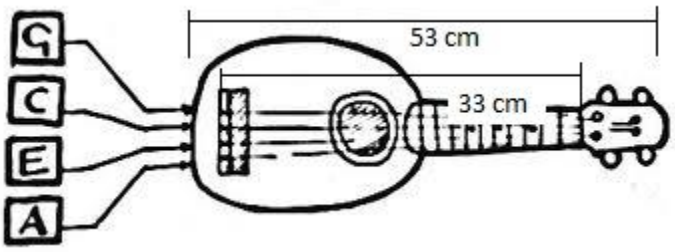


Figure 2.1 Soprano Ukulele

[cited from: http://www.uke-chords.com/ukulele-tunings.php]

The fret location and width then can be measured based on scale length. The location of fret is determined by the position nut to fret plus remainder of scale length divided by fret constant which is equal to 17.87.

$$B[n-1] = SL - N[n] \tag{2.1}$$

$$N[n] = \left(\frac{B[n-1]}{17.87}\right) + N[n-1] \tag{2.2}$$

Where B denotes bridge to fret, SL denotes scale length, N denotes nut to fret and n is integer greater of equal to 1. To calculate actual value for all frets position we first need to know N[1] which is easily SL divided by fret constant. In this case we are taking the actual physical data where SL equal to 33 cm.

$$N[1] = \left(\frac{SL}{17.87}\right) = \frac{33}{17.87} = 1.846\ cm \tag{2.3}$$

Then we can easily calculate the position of 24 frets by calculating N for every n less or equal to 24. Note that the fret distance is the distance from nut to the actual fret bridge. Results are tabulated below.

Table 2.1 Frets Position

Fret number	N(cm)	Fret number	N(cm)	Fret number	N(cm)
1	1.846	9	13.378	17	20.639
2	3.600	10	14.479	18	21.333
3	5.250	11	15.519	19	21.988
4	6.808	12	16.5	20	22.606
5	8.278	13	17.426	21	23.189
6	9.665	14	18.3	22	23.74
7	10.975	15	19.125	23	24.259
8	12.211	16	19.904	24	24.75

From this result by subtracting N[n+1] and N[n], we then can calculate the width of each fret. Results are tabulated below.

Table 2.2 Frets Width

Fret number	ΔN(cm)	Fret number	ΔN(cm)	Fret number	ΔN(cm)
1	1.846	9	1.167	17	0.735
2	1.754	10	1.101	18	0.694
3	1.65	11	1.04	19	0.655
4	1.558	12	0.981	20	0.618
5	1.47	13	0.926	21	0.583
6	1.387	14	0.874	22	0.551
7	1.31	15	0.825	23	0.519
8	1.236	16	0.779	24	0.491

For our design we are limiting to 11^{th} fret since width gets less than 1 cm after that. Also in terms of practicality, average adult male finger tips are about 1 cm. So, any frets with width smaller than 1 cm will be difficult to be pressed accurately. But for our Final design we decided to keep the fret width constant at 1.67 cm per fret. This way our potentiometer was limited to fit three frets at a time.

2.1 Design Procedure and Detail

The original design is not modified significantly from the design review. Most of the components in the final design are in accordance with the previous design, except for additional small components. The slight modification in this design is the speaker. Since the speaker in the original design, AS5008-32 manufactured by PUI Audio, was broken due to short circuit. In our final design we use commercial available speaker. The overall design procedure for each component including the software algorithm will be described below.

2.1.1 Arduino Uno Microcontroller

The main part of our project is to find a control platform, so the first step in designing our project is to find a microcontroller that meets the project requirement: PWM pins for outputting digital sound, four analog inputs for potentiometers, for flex sensors, one digital input/output for proximity sensor, ability to transmit (TX) and receive (RX) through serial data process. With this consideration we chose Arduino Uno with ATmega328 chip for our project. Both the left and the right hand use Arduino Uno as the microcontroller.

The Arduino Uno has 16 MHz clock speed which enable operation at very high speed. The first goal of our microcontroller is to generate an analog sinusoidal output. To generate analog output we use Arduino capabilities to generate PWM (pulse width modulation signal). The analog output amplitude is proportional to the duty cycle of PWM generated. To generate the desired PWM signal at lower frequency, we want to modify the clock speed of Arduino Uno and we do this by taking advantage of Timer2 register of AT Mega 328P-PU. First we write on TIMSK2 register which is Timer2 interrupt mask register. For normal operation, this register usually not changed but for our project time-sensitive purpose we want to modify this register. We first modify bit 0 in this register which is TOIE2, Timer2 overflow interrupt enable. By modifying this register, we directly command the program to perform the interrupt sub-routine when Timer2 overflow. Inside the interrupt subroutine we determine the PWM duty cycle by modifying OCR2A, output compare register A. The OCR2A value determines the duty cycle of PWM with 255 correspond 100% duty cycle. In order to avoid any timing distortion we disable Timer0 functionality which disables the delay function in Arduino software.

The last important step is to transmit and to receive digital data from one microcontroller to another microcontroller. VirtualWire library is used in this step because this function provides features that send a short message. For initial testing, the array of characters ‘Hello World’ is sent. And, the other microcontroller is able to receive this message without any loss of information. Therefore, two Arduino Uno as microcontrollers are used for this project.

2.1.1.1 Low Pass Filter

Before connecting the PWM output of the microcontroller into the speaker, we put it through low pass filter. This filter has a purpose to pass the desired frequencies signals and block the unwanted high frequencies signals.

2.1.2 Proximity Sensor

The Parallax’s PING))) ultrasonic proximity sensor is used for distance detection measurement. This sensor only requires one I/O pin. The activity status of the LED indicator indicates if the sensor is detecting an object. This sensor works by transmitting an ultrasonic burst and providing an output pulse that corresponds to the time required for the burst echo to return to the sensor. By measuring the echo pulse width, the distance to target can easily be determined. This device is chosen because it can detect an object within the range of 2 cm to 3 m with narrow acceptance beam angle. This device also provides reliability with 1cm resolution.

2.1.3 Potentiometer

Four soft membrane potentiometers manufactured by Spectra Symbol are used to determine which frets are being pressed. Each membrane has an effective length of 50 mm. By pressing down on various part of the strip, the resistance linearly changes from 1000Ohms to 10,000Ohms allowing the user to accurately calculate the relative position on the strip. These four potentiometers are connected into four analog inputs on the Arduino Uno microcontroller. One slight modification from the design review is that the addition of resistors for each inputs into the microcontroller. When the potentiometer was connected to digital multi-meter for testing, the resistance value of the potentiometer decreases slowly to 1000Ohms. This behavior was not accurate enough to determine the fret positions. So, an additional 33kOhms resistor is placed between the analog input of the microcontroller and the potentiometer so that the resistance value of the potentiometer decreases drastically to 1000Ohms when it is not pressed.

2.1.4 RF Transmitter and Receiver

For transmitter we use WRL-10534 with 434 MHz frequency manufactured by Wenshing Corporation. This transmitter is chosen due to low cost and easy to use. The initial design with data transmission is coded the Arduino platform with Serial.read() functionality. Binary data encoding is transmitted into the receiver. However, when the receiver is tested using the oscilloscope, there are some discrepancy between the data transmitted and data received. Therefore, VirtualWire library is used for both transmitting and receiving data. The VirtualWire library provides features to send short messages, without retransmit data.

For receiver module we use WRL-10532 with 434 MHz frequency which also manufactured by Wenshing Corporation. The transmitter and receiver must have the same frequency value to ensure correct data transmission. As same as the RF transmitter described in section 2.2.4, the VirtualWire library is used to ensure correct data reading.

2.1.5 Power

A 9 volts battery will be used to power up all of the components on the left and right parts of the projects. There are some modifications in this block. Four AA batteries were used for the initial design. However, a 9 volts battery with power regulator is enough to ensure that all of the components are powered up. The Arduino microcontroller needs 7 to 12 volts input voltage. So, it can be powered up directly. However, the rest of the components only require at most 5 volts power. So, the power regulator is used to ensure that the components won't blow up.

2.1.6 Flex Sensor

Four flex sensors manufactured by the Spectra Symbol are used to detect which strings are being picked. Each flex sensor has an effective length of 4.5 inch. The sensor changes in resistance depending on the amount of bend of the sensor. The sensor converts the change in bend to electrical resistance; the more the bend, the more the resistance value. The resistance can change from 60kOhms to 110kOhms depending on the amount of the bend. The flex sensors will be connected directly into four analog inputs of the Arduino Uno.

3. Design Verification

In order to make sure our components fulfill our requirements, we performed several tests on each individual module. Refer to the requirement and verification table in appendix A for testing focus and verification methodology.

3.1 Microcontroller

Our Microcontroller meets our requirement and specification. When connected with 5V input, Arduino uno was able to generate PWM signal. When the PWM output passed through low pass filter, it generated sinusoidal wave. The frequency of the sinusoidal wave fulfills the required range of operation, which is between 196 Hz to 830 Hz.

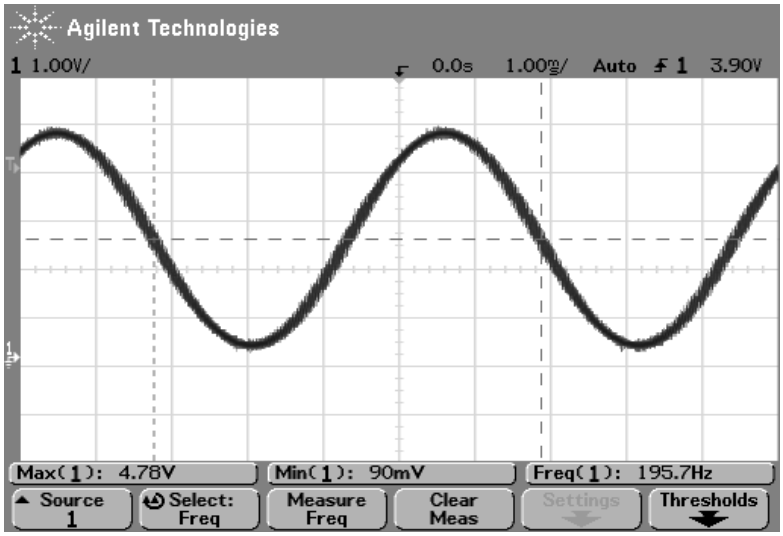


Figure 3.1.1 Analog Sinusoidal output at 195.7 Hz

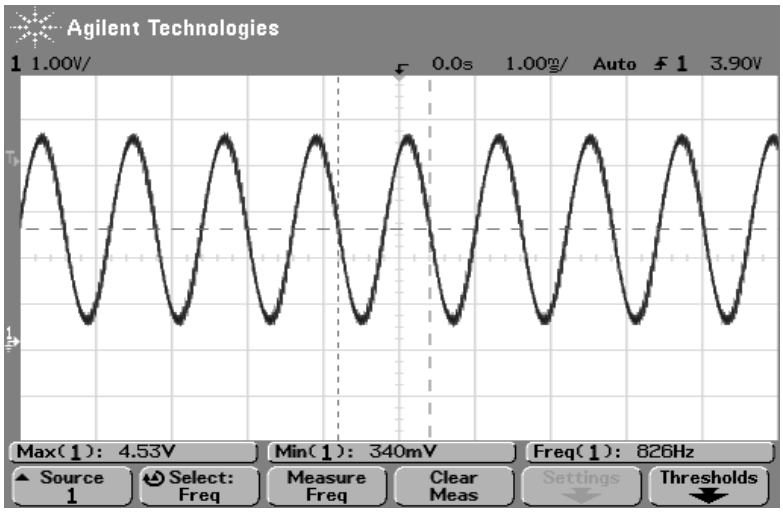


Figure 3.1.2 Analog Sinusoidal output at 826 Hz

3.2 Power

Our power module fulfills our requirement and work perfectly with our design. When connected with regular nine volt battery, the voltage output of the power module will be five volt. The output voltage does not show any peak spikes, which mean that power outputs at constant rate. The nine volt input is shown as channel 1 and the five volt output is shown as channel 2 in the oscilloscope graph.

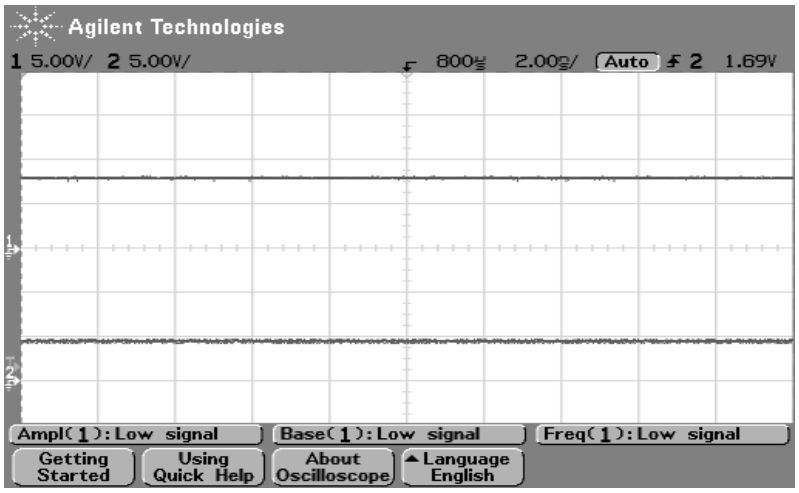


Figure 3.2.1 Power module input/output

3.3 Proximity Sensor

The proximity sensor meets the requirements and specifications for this project. The sensor was able to locate the object at a particular distance away from the sensor. Simple initial testing was conducted to measure the accuracy of the sensor. The output of the sensor was connected to the oscilloscope. The figure 3.3.1 below indicates a distance measurement. The red line below indicates if the proximity sensor is on. Then, the area blue line indicates a sensor detects an object. The sensor sends an ultrasonic pulse to the object, and it waits for the echo. The time between the pulse and the echo is used to determine the distance as seen in the green line in figure 3.3.1 below. The period of the measurement is 795 us. So, by taking the ultrasonic pulse velocity in the air to be 0.33 mm/us, the distance can be calculated as follow:

$$\text{Distance} = \text{Velocity} \times \text{Period} \qquad (3.3.1)$$

Using the formula 3.3.1 above, the distance measured is 26.2 cm away from the sensor.

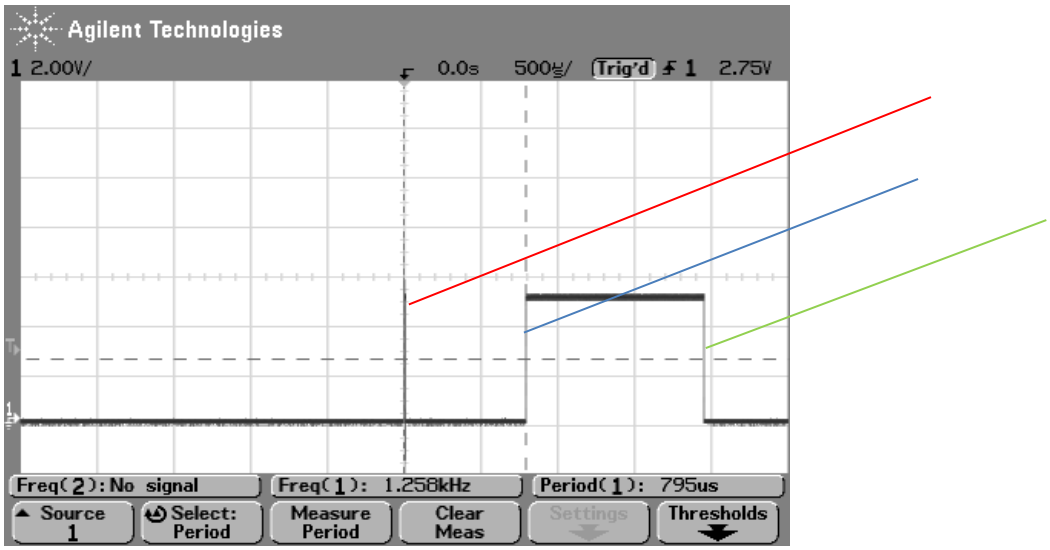


Figure 3.3.1 Proximity sensor module input/output

3.4 Fret potentiometer

The fret potentiometer fulfills our specification. The potentiometer works perfectly by reducing the output voltage linearly as a function of position. We recorded several data points.

Table 3.4.1 Measured voltage output

Distance Pressed(cm)	Vout (V)
0	4.96
1	3.72
2	2.68
3	1.85
4	1.81
5	5.10E-04

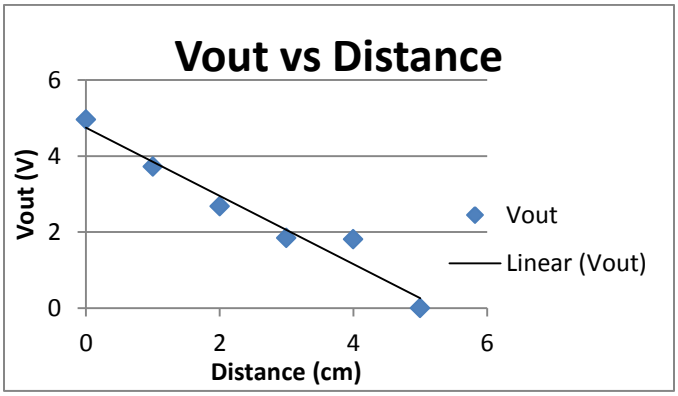


Figure 3.4.1 Linear function of voltage output

3.5 RF Trans-receiver

The RF transceiver generates bit errors when measured with high frequency value. For the initial testing of the transceiver, both transmitter and receiver were connected to the oscilloscope. Channel 1 indicates the data transmitted and channel 2 indicates the data received. The transmitter and receiver were set one meter apart. The transmitter was supplied with 35.7 Hz frequency from the pulse generator. At this frequency, the data transmitted and the data received are identical; there is no bit error in the received data as shown in figure 3.5.1 below.

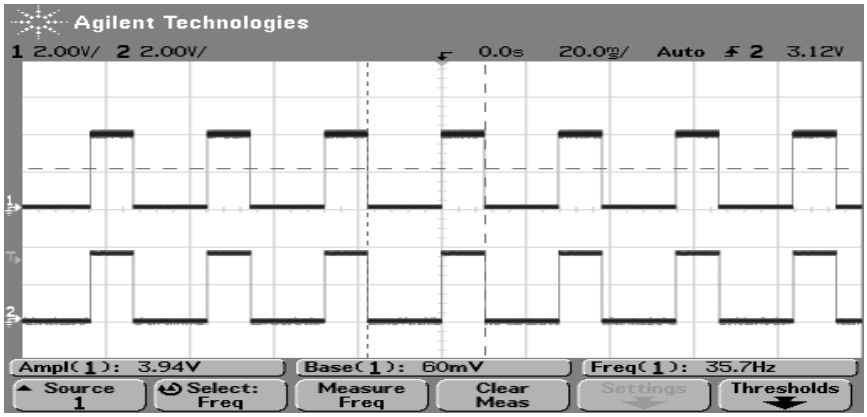


Figure 3.5.1 Data Transmission Recorded with Oscilloscope

However, as the frequency increased to 68.5 Hz, the RF trans-receiver generates error in the bits received. The red line in the figure 3.5.2 below indicates an error bit in the RF trans-receiver measurement due to the noise introduced by the environment.

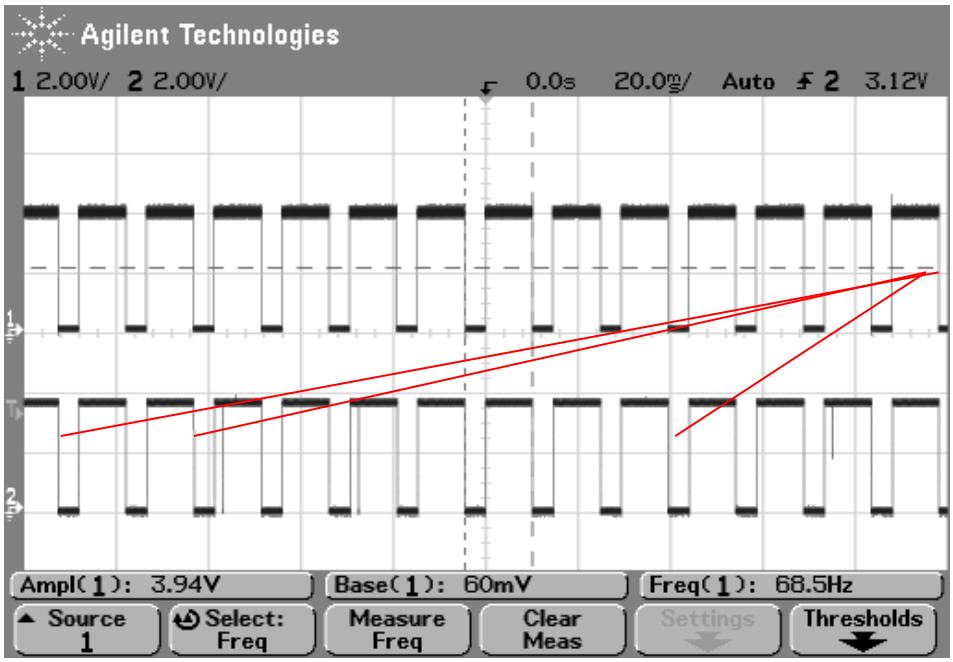


Figure 3.5.2 The RF Transceiver Generates Bits Errors

However, as we implement the RF transceiver in the system, the bits errors does not affect the device's performance. The RF trans-receiver is able to transmit and receive data correctly.

3.6 Flex Sensor

The flex sensor meets the requirements and specifications in this project. The flex sensor is sensitive enough so that the small amount of bending changes the resistance value. The sensor was connected to the digital multi-meter and the resistance values were measured. The flex sensor was bent down each 0.5 cm. The table below measured the flex sensor's resistance with digital multi-meter.

Table 3.5.1 Measurement of Flex Sensor Resistance

Effective Length (cm)	Resistance (kΩ)
0.5	25
1	25.54
1.5	23.93
2	24.79
2.5	23.14
3	23.53
3.5	23.56
4	24.2
4.5	26.07
5	26.4
5.5	28.92
6	28.4
6.5	27.74
7	25.39
7.5	24.83
8	24.88
8.5	23.93
9	25.7
9.5	23.8
10	21
10.5	19.2
11	19.9

From the table above, a plot was drawn using Excel. From the plot below, a quadratic interpolation was drawn. From the plot in figure 3.6.1 below, the peak point was observed when the effective length at 5.5 cm. Therefore, in order to get a sensitive response, the effective length of the flex sensor would be placed at the top of the finger joint.

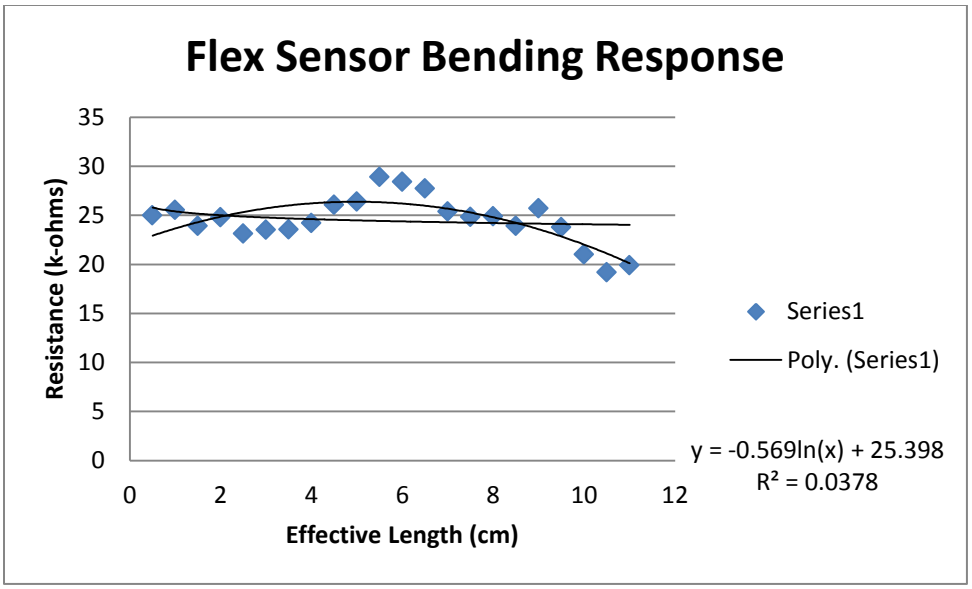


Figure 3.6.1 Flex Sensor Bending Response

4. Costs

4.1 Parts

The following table below is a table for part costs in the project.

Table 4.1.1 Cost of Parts

Part	Manufacturer	Quantity	Retail Cost ($)	Bulk Purchase Cost ($)	Actual Cost ($)
Proximity Sensor	Parallax	1	29.95	29.95	29.95
Battery	Duracell	1	2	2	2
Flex Sensor	Spectra Symbol	4	12.95	51.8	51.8
RF Transmitter	Wenshing	1	3.95	3.95	3.95
RF Receiver	Wenshing	1	4.95	4.95	4.95
Microcontrollers	Arduino	2	22.12	44.24	0
Resistors, Capacitors, Wire	Various		2	2	2
Potentiometers	Spectra Symbol	4	14.25	57	57
Voltage Regulator	Parts Express	2	1	2	2
Total Cost			93.17	197.89	153.65

4.2 Labor

The following table below is the cost of labors table.

Table 4.2.1 Cost of Labors

Person	Ideal Salary / Hour ($)	Actual Hours Spent (hours)	Labor (x2.5)($)
Ivan Setiawan	20	240	12000
Satyo Iswara	20	240	12000

We estimate the time spent in this project to be approximately 240 hours. Using 4.1, the labor cost for each person is $12,000. Combining the labor's cost and the parts' cost, the total cost of this project is $24,197.89.

5. Conclusion

5.1 Accomplishments

Majority of the individual components in this project were functioning as expected. The proximity sensor was able to sense the distance of the right hand accurately. The inputs from the potentiometers and from the proximity sensor were able to determine the frets positions of the ukulele. Then, the left hand was able to send the data to the right hand via RF transceiver. The right hand device was able to get the correct notes. When connecting the PWM output pin to the speaker, the ukulele like sounds were generated.

5.2 Uncertainties

There are some uncertainties we encounter in integrating the whole project. The proximity sensor of the left hand system is not accurate enough for detecting the object's distance. The sensor has 1 cm resolution. However, if the object is 24.7 cm away from the sensor, the distance measured will be 25 cm. This problem would cause an uncertainty in the note that is being played.

The RF trans-receiver has lot of noise as explained in section 3.5. So, the noise will cause incorrect data transfer to the right hand. Since the transmitted data is also calculated to determine the notes, the generated sound might not be the note that is intended to play.

5.3 Ethical considerations

There are several ethical issues to consider when designing our idea. Our group adheres to the IEEE Code of Conduct and Ethical Guidelines. Our project is intended to improve the understanding of technology, applications, and the consequences. For example, this project combines the flex sensor, proximity sensor, and as well as microcontroller to design an air-ukulele. Furthermore, all voltages will be safely regulated and all components will be shielded from user contact. So, this project accepts the responsibility in making decisions with the safety, health, and welfare of the public.

5.4 Future work

Below are the lists of improvements that would improve the performance of the device:

- Replace current RF trans-receiver with another RF trans-receiver with higher bps (bits per second). Our current trans-receiver has 4800 bps. Increasing the data rate transfer would allow more information to be sent.
- Replace current proximity sensor with more accurate distance sensing device. The proximity sensor in this project can only measure the object with the accuracy of 1 cm resolution. So, if the accuracy can be improved by 0.1 cm or maybe even less than 0.1 cm, then the sensor will be able to sense more accurately than the current proximity sensor.
- Code the microcontroller to get rid of the delay in the system.
- Remove the Arduino platform to reduce the budget cost significantly.

References

[1] "Arduino Uno SMD - SparkFun Electronics." *News - SparkFun Electronics*. N.p., n.d. Web. 12 Feb. 2012. Avalible at : http://www.sparkfun.com/products/10356

[2] "Basic LED - Red - SparkFun Electronics." *News - SparkFun Electronics*. N.p., n.d. Web. 21 Feb. 2012. Avalible at : http://www.sparkfun.com/products/533

[3] "Flex Sensor." *Sparkfun.com*. spectrasymbol.com, n.d. Web. 2 Nov. 2012. Avalible at : www.sparkfun.com/datasheets/Sensors/Flex/FLEXSENSOR(REVA1).pdf

[4] "LV-MaxSonar Â® -EZ." *maxbotix.com*. N.p., n.d. Web. 10 Feb. 2010. Avalible at : www.maxbotix.com/documents/MB1040_Datasheet.pdf

[5] "RF module datasheet." *Sparkfun.com*. Sparkfun.com, n.d. Web. 12 Feb. 2012. Avalible at : dlnmh9ip6v2uc.cloudfront.net/datasheets/Wireless/General/RWS-371-6_433.92MHz_ASK_RF_Receiver_Module_Data_Sheet.pdf

[6] "TWS-BS RF MODULE Series." *Sparkfun.com*. Sparkfun.com, n.d. Web. 2 Dec. 2012. Avalible at : dlnmh9ip6v2uc.cloudfront.net/datasheets/Wireless/General/TWS-BS-3_433.92MHz_ASK_RF_Transmitter_Module_Data_Sheet.pdf

[7] "Variable Potentiometer Membrane Strip, 50mm (SoftP." *MicroControllerShop.com Micro-Controller Development Tools*. N.p., n.d. Web. 20 Feb. 2012. Avalible at : http://microcontrollershop.com/product_info.php?products_id=3803

[8] “Calculating Fret Position.” *Lutherie Information Website.* Mon. 5 Mar. 2012. Avalible at : http://liutaiomottola.com/formulae/fret.htm

[9] “Arduino DDS Sinewave Generator.” Laboratory for Experimental Computer Science at the Academy of Media Arts Cologne. 7 April.2012. Available at: http://interface.khm.de/index.php/lab/experiments/arduino-dds-sinewave-generator/

[10] “Ping Ultrasonic Sensor Data Manual”, Parallax, Inc, 2012. Available at: http://www.parallax.com/Store/Sensors/AccelerationTilt/tabid/172/List/0/ProductID/543/Default.aspx?SortField=ProductName%2CProductName

[11] A. Outi, “From Technology to Art”. 19 July. 2011. Web. 10 Mar. 2012. http://fromtechnologytoart.blogspot.com/2011/07/how-to-connect-flex-sensor-to-arduino.html

[12] Y. Ning, and A. Beech. “Wearable Air Guitar”. Web. 3 Mar 2012. Available at: http://instruct1.cit.cornell.edu/courses/ee476/FinalProjects/s2004/ddb25/complete2.html

Appendix A Requirement and Verification Table

Requirement	Verification	Verification Status
Microcontroller (A)		
Testing Focus: For microcontroller A, the computation of the fret location is the most important part because the fret will be sent into the right hand for further decoding. In order to compute the fret location, we will code the microcontroller. The microcontroller functions correctly under voltage range of 7V – 12V Be able to send the signal at 4800 bps. Correctly decode one octave frequency range (262 Hz – 523 Hz) The delay time between the data input from the proximity sensor must be less than 200 ms Successfully output the analog signal into the fret indicator in the range of 0 – 5 volts with input above 2.5 V considered as high Be able to receive the signal from the fret potentiometer under 0 – 5 volts	**Acceptable Range of Operation:** The microcontroller A can send the data to the transmitter within one octave range (196 Hz – 830 Hz). Connect to the power supply at 7, 9, 12 V, and verify the correct input with function generator and correct output with LED and oscilloscope Code the microcontroller with 4800 bps data. Then, we connect the digital pin out of the microcontroller to the oscilloscope. Using the trigger function and sample the time in 10ms, we can see the data bits are generated. Code the microcontroller with one octave range every 1 second, and then connect the digital pin out to the oscilloscope. Using math function in the oscilloscope and convert the signal into frequency domain, we can measure the peak to peak voltage. Program the microcontroller with the delay phase less than 200ms and connect it to the oscilloscope to convert the signal into frequency domain. Then, we can use the delay sweep function to verify if the delay is less than 200ms. Wire the digital output pin of the microcontroller to the digital high (5V) and digital low (0V), and wire the LED at the output pin to check if it is on or off. Wire the digital output pin of the microcontroller to the digital high (5V) and digital low (0V), and wire the LED at the output pin to check if it is on or off	✓

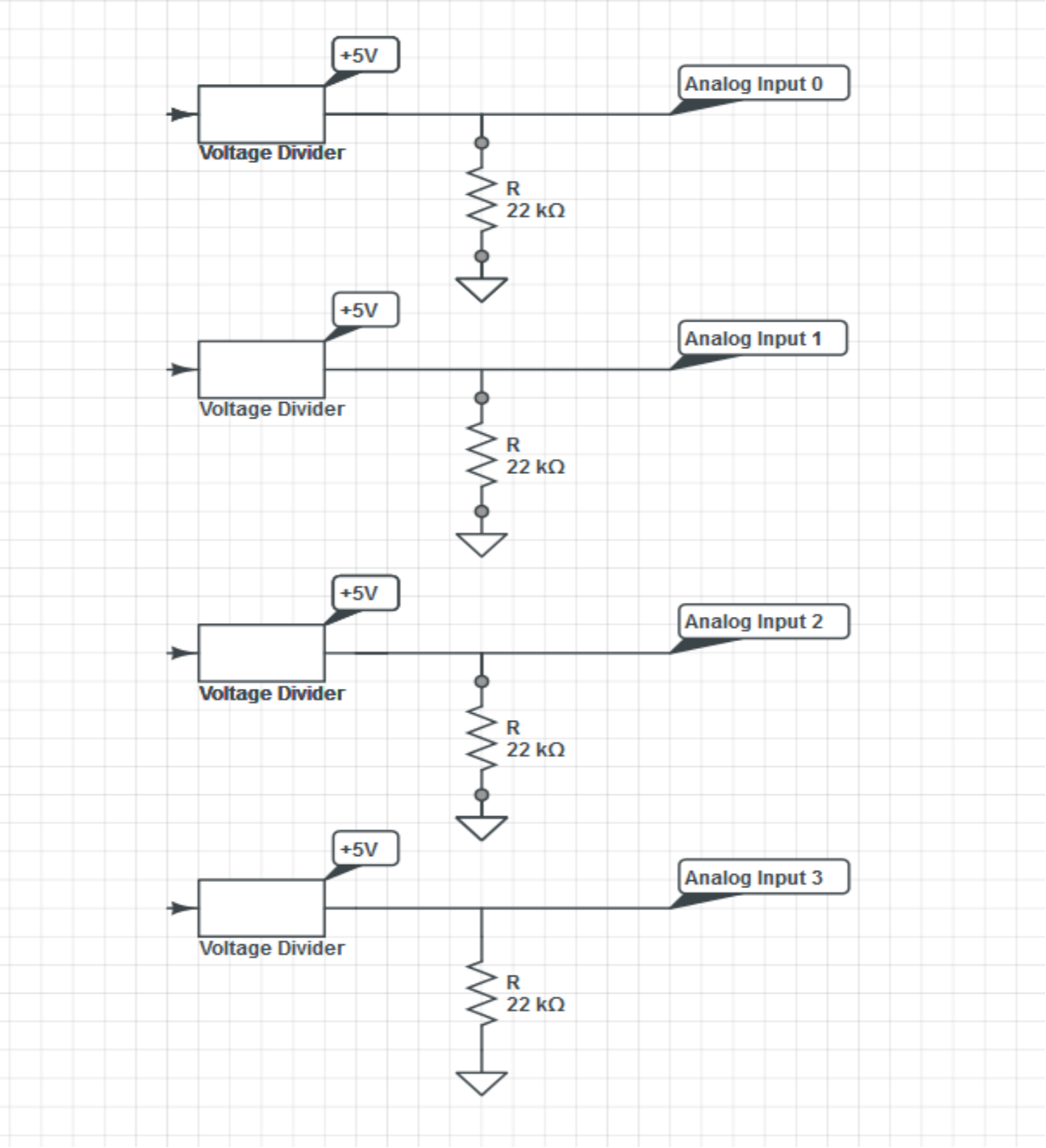
Microcontroller (B)		
Testing Focus: For microcontroller B, the decoding is the most important part because the microcontroller will output the generated the sound into the speaker. The microcontroller functions correctly under voltage range of 7V – 12V Be able to generate correct notes within one octave in frequency range of 262 Hz – 523 Hz Correctly generate note C (262 Hz) Correctly read the digital encoding of note C (262 Hz) The sounds must be damped after 10ms using the Karplus Strong synthesis The delay time between the input and the output must be less than 200ms Successfully receive analog signal from flex sensor in range 0 – 5 V with input above 2.5 V consider as high	**Acceptable Range of Operation:** The microcontroller B can generate analog sinusoidal signal with frequency between 196 Hz – 830 Hz. Connect to the power supply at 7, 9, 12 V, and verify the correct input with function generator and correct output with LED and oscilloscope Connect the output to the speaker and be able to hear the correct sound of the entire octave in a sequence and the expected output must be match to the tuner Code the microcontroller to generate note C every 1 second and connect it to the oscilloscope. Using the math function in the oscilloscope and convert the signal to frequency domain, we can measure the peak of the frequency pulse Wire the digital input pin of the microcontroller to the digital high (9V) and digital low (0V), and wire the LED at the output pin to check if it is on or off. Program the synthesizer to the microcontroller and connect it to the oscilloscope. If the amplitude of the note C is reduced by 50% of the initial and less than 10% after 10ms, then the synthesizer behaves correctly. Program the microcontroller with the delay phase less than 200ms and connect it to the oscilloscope to convert the signal into frequency domain. Then, we can use the delay sweep function to verify if the delay is less than 200ms. Wire up the analog input pin of the microcontroller with 0, 2.5, and 5V. Then, wire the LED at the analog pin output to check if it is on or off.	✓
Power		
Testing Focus: The most important for power are the voltage and the maximum current. The voltage and the current will determine	**Acceptable Range of Operation:** The voltage has 9V input and output constant output of 5 V.	✓

total power supplied. We need to ensure that each device is supplied with appropriate power in order to function properly. Four AA batteries will be used to generate 9V DC input power since most of the components must have voltage range of 5V – 9V	Connect battery to volt meter and check the output voltage with voltmeter and oscilloscope. The voltage input measure should stay constant at 9 V and output constant at 5V.	
Proximity Sensor		
Testing Focus: The proximity sensor must be able to detect object in the range of 30 cm – 2 meters. The output of the sensor is the voltage where the closer the object to the sensor, the voltage read will be smaller. Proximity sensor function correctly under 5 V input Be able to generate 2.5 – 5 volts output for distance between 30 cm to 2 m. Be able to generate correct output at fixed distance of 50 cm	**Acceptable Range of Operation:** The proximity sensor has .772V output for object detection at 2m and .116 V for 30 cm detection. So the expected range of proximity sensor should be between .116V and .772V with output increase as object detected farther. Connect to power supply and read output change indicating device working Connect to power supply and vary distance between proximity sensor and the object. The voltage will vary 0.98mV / inch	✓
Fret Potentiometer		
Testing Focus: Resistance is the most important parameter. The change of resistance on the sensor depends on whether the sensor is pressed or not. And, the change of resistance will determine which fret is being pressed. Fret potentiometer able to generate resistance ranging from 1 kΩ to 10 kΩ. Generate 1 kΩ when not pressed	**Acceptable Range of Operation:** The sensor has 0V – 5V output. The zero volt output correspond to potentiometer not being pressed and 5 volt output correspond to pressing the edge of potentiometer. Connect the potentiometer into the digital multi-meter, and then check the resistance output. Press the potentiometer then checks the output resistance with Digital Multimeter. The output resistance is 1 kΩ.	✓
RF Trans-receiver		
Testing Focus: For the transceiver, frequency is the most important parameter because the receiver must be able to receive the data from transmitter with specific frequency. For example, if note C (262 Hz) is sent	**Acceptable Range of Operation:** The transceiver can transmit and receive the data within 1 m radius. Digital signal will be sent from the microcontroller to the transmitter. A	✓

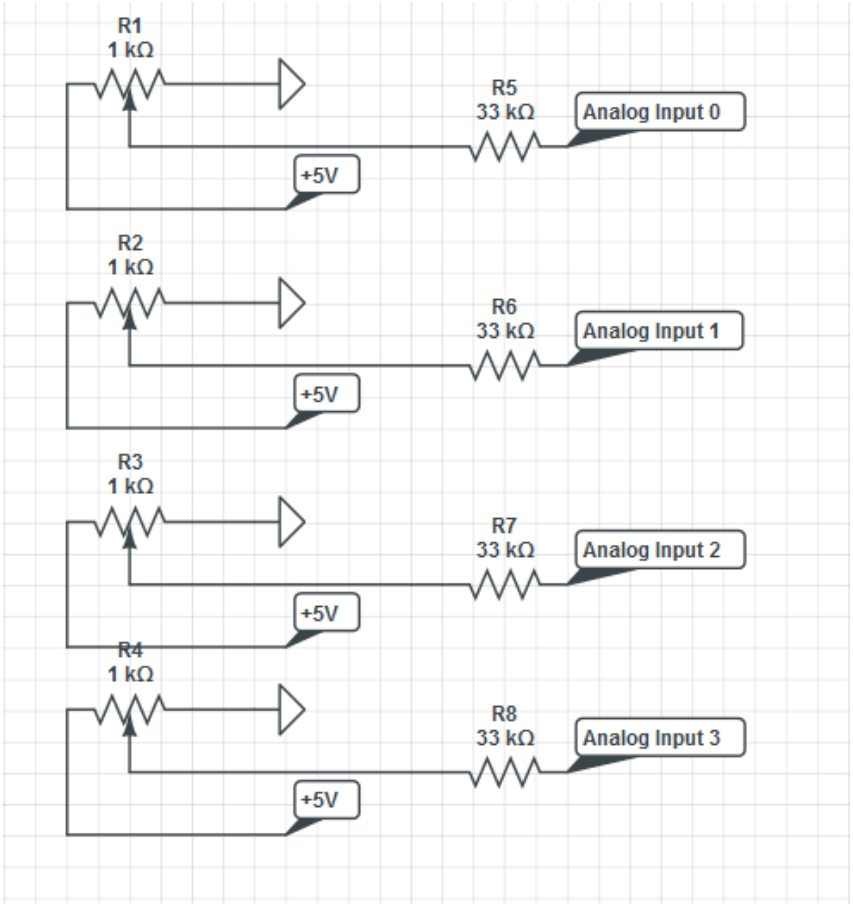
from transmitter, the receiver must be able to receive the data. RF transmitter will accept an input and transmit it through the antenna. Signal should be receivable at least 1 meter away. The turn on time for the transmitter after being powered up should be 1 ms. The RF receiver should receive signal from RF transmitter for at least 1 meter away.	distance test will measure the signal to noise ratio of the receiver to determine the maximum RF transmission distance. Additionally, the unit will be powered on and data will be inputted. Then, the antenna output will be measured to ensure it can transmit within 1ms power-on. We will continuously transmit data and then measure the output vs. distance. The point at which the data outputted cannot be recognized will be the maximum distance.	
Flex Sensor		
Testing Focus: Resistance is the most important parameter. The change of resistance on the sensor depends on the amount of bend on the sensor. And, the more the bend, the more the resistance value. The change in resistance will determine which string is being picked. Flex sensor able to generate resistance ranging from 10 kΩ to 30 kΩ. Generate 10kOhm when not bend	**Acceptable Range of Operation:** The flex sensor has 10kΩ - 30kΩ resistance range. When the sensor is not bent, the resistance is 10 kΩ. When the sensor is bent, the maximum resistance is 30 kΩ. Connect the flex sensor into the digital multi-meter, and then check the resistance. When the flex sensor is not bent down then checks the output resistance with the DMM. The output resistance is 10kΩ.	✓

Appendix B Requirement and Verification Table

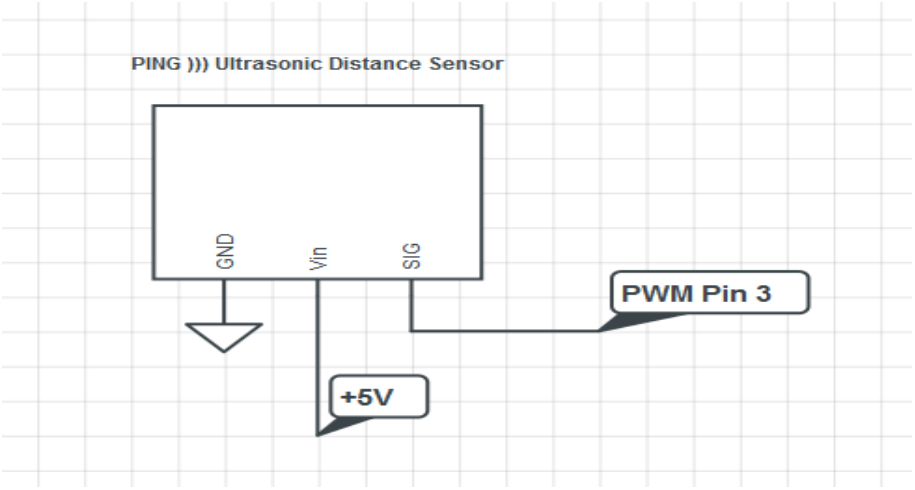
Flex Sensor



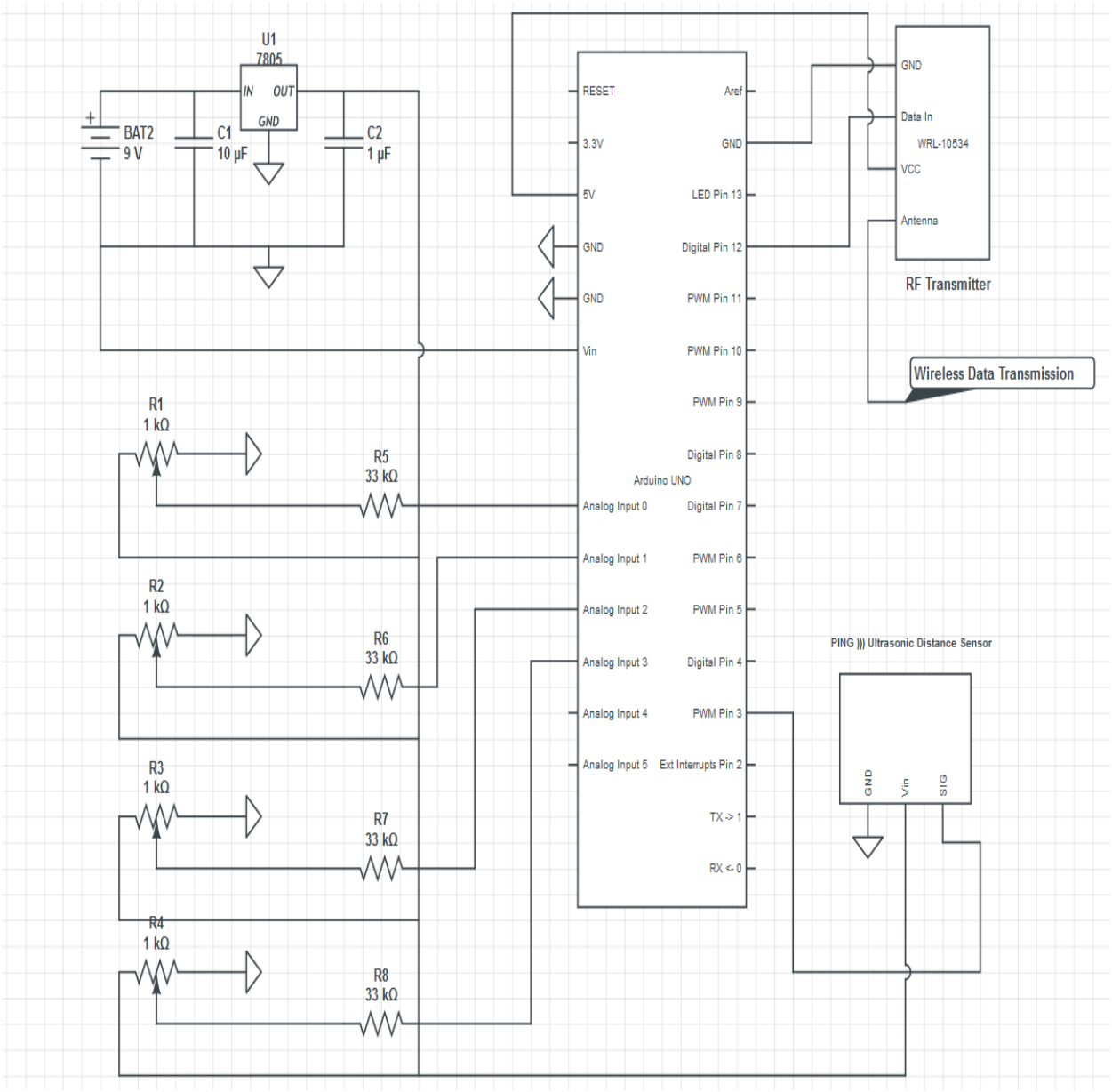
Potentiometers



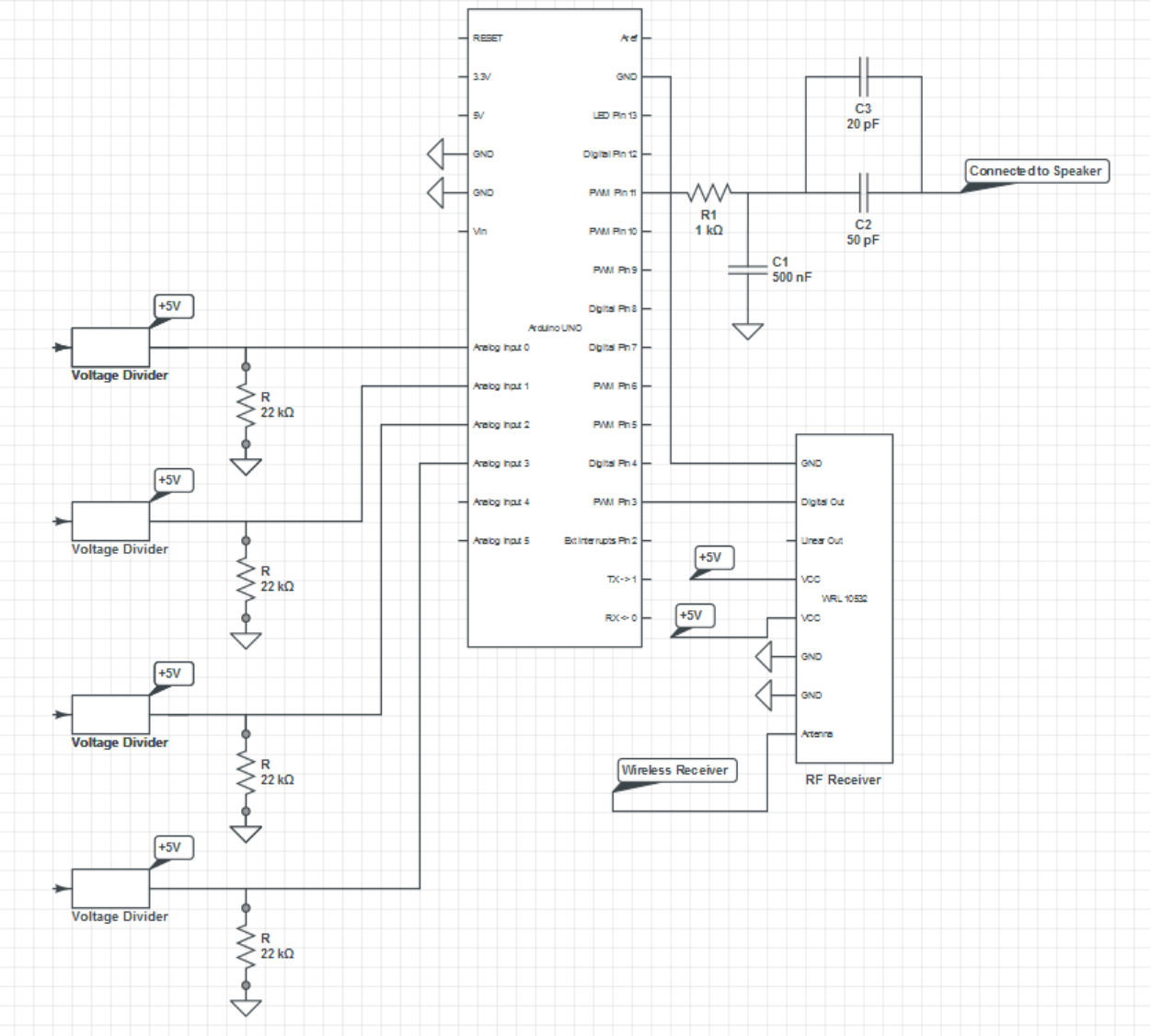
Proximity Sensor



Left Hand


U1
7805
IN
OUT
GND
BAT2
9 V
C1
10 µF
C2
1 µF
RESET
3.3V
5V
GND
GND
Vin
Aref
GND
LED Pin 13
Digital Pin 12
PWM Pin 11
PWM Pin 10
PWM Pin 9
Digital Pin 8
Arduino UNO
Analog Input 0
Analog Input 1
Analog Input 2
Analog Input 3
Analog Input 4
Analog Input 5
Digital Pin 7
PWM Pin 6
PWM Pin 5
Digital Pin 4
PWM Pin 3
Ext Interrupts Pin 2
TX -> 1
RX <- 0
GND
Data In
WRL-10534
VCC
Antenna
RF Transmitter
Wireless Data Transmission
R1
1 kΩ
R2
1 kΩ
R3
1 kΩ
R4
1 kΩ
R5
33 kΩ
R6
33 kΩ
R7
33 kΩ
R8
33 kΩ
PING))) Ultrasonic Distance Sensor
GND
Vin
SIG

Right Hand


RESET
3.3V
5V
GND
GND
Vin
Aref
GND
LED Pin 13
Digital Pin 12
PWM Pin 11
PWM Pin 10
PWM Pin 9
Digital Pin 8
Arduino UNO
Analog Input 0
Analog Input 1
Analog Input 2
Analog Input 3
Analog Input 4
Analog Input 5
Digital Pin 7
PWM Pin 6
PWM Pin 5
Digital Pin 4
PWM Pin 3
Ext Interrupts Pin 2
TX -> 1
RX <- 0
C3
20 pF
Connected to Speaker
R1
1 kΩ
C2
50 pF
C1
500 nF
+5V
Voltage Divider
R
22 kΩ
+5V
Voltage Divider
R
22 kΩ
+5V
Voltage Divider
R
22 kΩ
+5V
Voltage Divider
R
22 kΩ
GND
Digital Out
Linear Out
VCC
WRL 10532
VCC
GND
GND
Antenna
+5V
+5V
Wireless Receiver
RF Receiver

Appendix C Software Implementation

Left hand implementation

```
#include <VirtualWire.h>
//Poximity sensor pin assignment
const int pingPin = 2; //ping

//Potentiometer pin assignment
const int fret_in = 0; // potentiometer 1
const int fret2_in = 1; //potentiometer 2
const int fret3_in = 2; //potentiometer 3
const int fret4_in = 3; //potentiometer 4
//later might want to change to higher resolution indicator
//resolution .5 cm
//green LED
const int LEDG_1   = 12; // LED 1
const int LEDG_2   = 11; // LED 2
const int LEDG_3   = 10; // LED 3
const int LEDG_4   = 9;  // LED 4
//red LED
const int LEDR_1   = 8;  // LED 5
const int LEDR_2   = 7;  // LED 6
const int LEDR_3   = 6;  // LED 7
const int LEDR_4   = 5;  // LED 8
const int LEDR_5   = 4;  // LED 9
const int LEDR_6   = 3;  // LED 10

//mock demo const
const int speakerPin = 12;

//Variable used as transmission data
const int transmit_pin = 13;
const int transmit_en_pin = 1;
const int receive_pin = 2;

byte count = 1;


void setup() {
  //Integrating transmitter to left hand
  vw_set_tx_pin(transmit_pin);
  vw_set_rx_pin(receive_pin);
  vw_set_ptt_pin(transmit_en_pin);
  vw_setup(2000);
  Serial.begin(9600);
}
```

```
void loop()
{
  //universal variable for in range
  boolean on_neck;
  //var for prox sensor
  long duration, inches, cm;
  long cm_1;
  //cm for the closest edge to the right hand
  //cm_1 for the fathest edge to the  right hand
  //var for potentiometer
  int volt_in;
  int volt_in2;
  int volt_in3;
  int volt_in4;

  float fret_press_cm; //for potentiometer 1
  float fret_press_cm2; //for potentiometer 2
  float fret_press_cm3; //for potentiometer 3
  float fret_press_cm4; //for potentiometer 4

  float nut_to_fret_press;
  float nut_to_fret_press2;
  float nut_to_fret_press3;
  float nut_to_fret_press4;

  //ping process
  //ping active signal
  pinMode(pingPin, OUTPUT);
  digitalWrite(pingPin, LOW);
  delayMicroseconds(2);
  digitalWrite(pingPin, HIGH);
  delayMicroseconds(5);
  digitalWrite(pingPin, LOW);

  pinMode(pingPin, INPUT);
  duration = pulseIn(pingPin, HIGH);

  inches = microsecondsToInches(duration);
  cm = microsecondsToCentimeters(duration);
  cm_1 = cm + 5;

  Serial.print(cm);
  Serial.print("cm");
  Serial.println();

  //end of ping process

  //fret indicator process
```

```
pinMode(LEDG_1, OUTPUT);
pinMode(LEDG_2, OUTPUT);
pinMode(LEDG_3, OUTPUT);
pinMode(LEDG_4, OUTPUT);
pinMode(LEDR_1, OUTPUT);
pinMode(LEDR_2, OUTPUT);
pinMode(LEDR_3, OUTPUT);
pinMode(LEDR_4, OUTPUT);
pinMode(LEDR_5, OUTPUT);
pinMode(LEDR_6, OUTPUT);


if ((cm > 35)||(cm < 16))//outside range all LED blink
{
  on_neck = false;
  digitalWrite(LEDG_1, LOW);
  digitalWrite(LEDG_2, LOW);
  digitalWrite(LEDG_3, LOW);
  digitalWrite(LEDG_4, LOW);
  digitalWrite(LEDR_1, HIGH);
  digitalWrite(LEDR_2, HIGH);
  digitalWrite(LEDR_3, HIGH);
  digitalWrite(LEDR_4, HIGH);
  digitalWrite(LEDR_5, HIGH);
  digitalWrite(LEDR_6, HIGH);
  delay(50);
  digitalWrite(LEDR_1, LOW);
  digitalWrite(LEDR_2, LOW);
  digitalWrite(LEDR_3, LOW);
  digitalWrite(LEDR_4, LOW);
  digitalWrite(LEDR_5, LOW);
  digitalWrite(LEDR_6, LOW);
  delay(50);
}
else
{
  on_neck = true;
  digitalWrite(LEDG_1, HIGH);
  digitalWrite(LEDG_2, HIGH);
  digitalWrite(LEDG_3, HIGH);
  digitalWrite(LEDG_4, HIGH);
}

//end fret indicator process

//fret pressure process
volt_in = analogRead(fret_in);
```

```
//Adding more potentiometer input
volt_in2 = analogRead(fret2_in);
volt_in3 = analogRead(fret3_in);
volt_in3 = analogRead(fret4_in);

fret_press_cm = voltageToCentimeters(volt_in);
//Integrating for all potentiometers
fret_press_cm2 = voltageToCentimeters(volt_in2);
fret_press_cm3 = voltageToCentimeters(volt_in3);
fret_press_cm4 = voltageToCentimeters(volt_in4);

Serial.print(fret_press_cm);
Serial.print("cm");
Serial.println();
Serial.print(fret_press_cm2);
Serial.print("cm");
Serial.println();
Serial.print(fret_press_cm3);
Serial.print("cm");
Serial.println();
Serial.print(fret_press_cm4);
Serial.print("cm");
Serial.println();


//For fret 1 if it is being pressed
if ((fret_press_cm != 0) && (on_neck == true)) // fret 1 being press
{
  nut_to_fret_press = cm + fret_press_cm;
}
else // not pressed
{
  nut_to_fret_press = 0;
}


//Adding for more input
//If fret 2 is being pressed
if ((fret_press_cm2 != 0) && (on_neck == true)) // fret 2 being press
{
  nut_to_fret_press2 = cm + fret_press_cm2;
}
else // not pressed
{
  nut_to_fret_press2 = 0;
}
```

```
//If fret 3 is being pressed
if ((fret_press_cm3 != 0) && (on_neck == true)) // fret 3 being press
{
  nut_to_fret_press3 = cm + fret_press_cm3;
}
else // not pressed
{
  nut_to_fret_press3 = 0;
}


//If fret 4 is being pressed
if ((fret_press_cm4 != 0) && (on_neck == true)) // fret 4 being press
{
  nut_to_fret_press4 = cm + fret_press_cm4;
}
else // not pressed
{
  nut_to_fret_press4 = 0;
}

//Data to be sent
char msg[10] = {'g','0','c','0','e','0','a','0',' ','#'};
//int msg[4] = {392, 415, 440, 466};

//(lower_bound && high_bound)
// G4 string for mock demo purpose instead of send, we plays



//The Note Playing for string 1 (potentiometer 1)
if( nut_to_fret_press == 0 )
{
  //send G4 open
  tone(speakerPin,392);
  delay(150);
  msg[0] = 'g';
  msg[1] = '0';
}
else if( (nut_to_fret_press > 33.33) && (nut_to_fret_press < 35) )
{
  //send G#4 1st
  tone(speakerPin,415);
  delay(150);
  msg[0] = 'g';
  msg[1] = '1';

}
```

```
else if( (nut_to_fret_press > 31.67) && (nut_to_fret_press < 33.33) )
{
 //send A4 2nd
 tone(speakerPin,440);
 delay(150);
 msg[0] = 'a';
 msg[1] = '0';

}

else if( (nut_to_fret_press > 30) && (nut_to_fret_press < 31.67) )
{
 //send A#4 3rd
 tone(speakerPin,466);
 delay(150);
 msg[0] = 'a';
 msg[1] = '1';

}
else if( (nut_to_fret_press > 28.33) && (nut_to_fret_press < 30) )
{
 //send B4 4th
 tone(speakerPin,494);
 delay(150);
 msg[0] = 'b';
 msg[1] = '0';

}
else if( (nut_to_fret_press > 26.67) && (nut_to_fret_press < 28.33) )
{
 //send C5 5th
 tone(speakerPin,523);
 delay(150);
 msg[0] = 'c';
 msg[1] = '0';

}

else if( (nut_to_fret_press > 25) && (nut_to_fret_press < 26.67) )
{
 //send C#5 6th
 tone(speakerPin,554);
 delay(150);
 msg[0] = 'c';
 msg[1] = '1';

}
else if( (nut_to_fret_press > 23.33) && (nut_to_fret_press < 25) )
```

```
{
  //send D5 7th
  tone(speakerPin,587);
  delay(150);
  msg[0] = 'd';
  msg[1] = '0';

}

else if( (nut_to_fret_press > 21.67) && (nut_to_fret_press < 23.33) )
{
  //send D#5 8th
  tone(speakerPin,622);
  delay(150);
  msg[0] = 'd';
  msg[1] = '1';

}

else if( (nut_to_fret_press > 20) && (nut_to_fret_press < 21.67) )
{
  //send E5 9th
  tone(speakerPin,659);
  delay(150);
  msg[0] = 'e';
  msg[1] = '0';

}
else if( (nut_to_fret_press > 18.33) && (nut_to_fret_press < 20) )
{
  //send F5 10th
  tone(speakerPin,698);
  delay(150);
  msg[0] = 'f';
  msg[1] = '0';

}
else if( (nut_to_fret_press > 16.67) && (nut_to_fret_press < 18.33) )
{
  //send F#5 11th
  tone(speakerPin,740);
  delay(150);
  msg[0] = 'f';
  msg[1] = '1';

}
```

```
//If string 2 is being picked
if( nut_to_fret_press2 == 0 )
{
  //send G4 open
  tone(speakerPin,392);
  delay(150);
  msg[0] = 'c';
  msg[1] = '0';
}
else if( (nut_to_fret_press2 > 33.33) && (nut_to_fret_press2 < 35) )
{
  //send G#4 1st
  tone(speakerPin,415);
  delay(150);
  msg[0] = 'c';
  msg[1] = '1';

}
else if( (nut_to_fret_press2 > 31.67) && (nut_to_fret_press2 < 33.33) )
{
  //send A4 2nd
  tone(speakerPin,440);
  delay(150);
  msg[0] = 'd';
  msg[1] = '0';

}

else if( (nut_to_fret_press2 > 30) && (nut_to_fret_press2 < 31.67) )
{
  //send A#4 3rd
  tone(speakerPin,466);
  delay(150);
  msg[0] = 'd';
  msg[1] = '1';

}
else if( (nut_to_fret_press2 > 28.33) && (nut_to_fret_press2 < 30) )
{
  //send B4 4th
  tone(speakerPin,494);
  delay(150);
  msg[0] = 'e';
  msg[1] = '0';

}
else if( (nut_to_fret_press2 > 26.67) && (nut_to_fret_press2 < 28.33) )
```

```
{
  //send C5 5th
  tone(speakerPin,523);
  delay(150);
  msg[0] = 'f';
  msg[1] = '0';

}

else if( (nut_to_fret_press2 > 25) && (nut_to_fret_press2 < 26.67) )
{
  //send C#5 6th
  tone(speakerPin,554);
  delay(150);
  msg[0] = 'f';
  msg[1] = '1';

}
else if( (nut_to_fret_press2 > 23.33) && (nut_to_fret_press2 < 25) )
{
  //send D5 7th
  tone(speakerPin,587);
  delay(150);
  msg[0] = 'g';
  msg[1] = '0';

}

else if( (nut_to_fret_press2 > 21.67) && (nut_to_fret_press2 < 23.33) )
{
  //send D#5 8th
  tone(speakerPin,622);
  delay(150);
  msg[0] = 'g';
  msg[1] = '1';

}

else if( (nut_to_fret_press2 > 20) && (nut_to_fret_press2 < 21.67) )
{
  //send E5 9th
  tone(speakerPin,659);
  delay(150);
  msg[0] = 'a';
  msg[1] = '0';

}
else if( (nut_to_fret_press2 > 18.33) && (nut_to_fret_press2 < 20) )
```

```
{
  //send F5 10th
  tone(speakerPin,698);
  delay(150);
  msg[0] = 'a';
  msg[1] = '1';

 }
 else if( (nut_to_fret_press2 > 16.67) && (nut_to_fret_press2 < 18.33) )
 {
  //send F#5 11th
  tone(speakerPin,740);
  delay(150);
  msg[0] = 'b';
  msg[1] = '0';

 }




//If string 3 is being picked
if( nut_to_fret_press3 == 0 )
 {
  //send G4 open
  tone(speakerPin,392);
  delay(150);
  msg[0] = 'e';
  msg[1] = '0';
 }
 else if( (nut_to_fret_press3 > 33.33) && (nut_to_fret_press3 < 35) )
 {
  //send G#4 1st
  tone(speakerPin,415);
  delay(150);
  msg[0] = 'f';
  msg[1] = '0';

 }
 else if( (nut_to_fret_press3 > 31.67) && (nut_to_fret_press3 < 33.33) )
 {
  //send A4 2nd
  tone(speakerPin,440);
  delay(150);
  msg[0] = 'f';
  msg[1] = '1';

 }
```

```
else if( (nut_to_fret_press3 > 30) && (nut_to_fret_press3 < 31.67) )
{
  //send A#4 3rd
  tone(speakerPin,466);
  delay(150);
  msg[0] = 'g';
  msg[1] = '0';

}
else if( (nut_to_fret_press3 > 28.33) && (nut_to_fret_press3 < 30) )
{
  //send B4 4th
  tone(speakerPin,494);
  delay(150);
  msg[0] = 'g';
  msg[1] = '1';

}
else if( (nut_to_fret_press3 > 26.67) && (nut_to_fret_press3 < 28.33) )
{
  //send C5 5th
  tone(speakerPin,523);
  delay(150);
  msg[0] = 'a';
  msg[1] = '0';

}

else if( (nut_to_fret_press3 > 25) && (nut_to_fret_press3 < 26.67) )
{
  //send C#5 6th
  tone(speakerPin,554);
  delay(150);
  msg[0] = 'a';
  msg[1] = '1';

}
else if( (nut_to_fret_press3 > 23.33) && (nut_to_fret_press3 < 25) )
{
  //send D5 7th
  tone(speakerPin,587);
  delay(150);
  msg[0] = 'b';
  msg[1] = '0';

}
```

```
else if( (nut_to_fret_press3 > 21.67) && (nut_to_fret_press3 < 23.33) )
{
  //send D#5 8th
  tone(speakerPin,622);
  delay(150);
  msg[0] = 'c';
  msg[1] = '0';

}

else if( (nut_to_fret_press3 > 20) && (nut_to_fret_press3 < 21.67) )
{
  //send E5 9th
  tone(speakerPin,659);
  delay(150);
  msg[0] = 'c';
  msg[1] = '1';

}
else if( (nut_to_fret_press3 > 18.33) && (nut_to_fret_press3 < 20) )
{
  //send F5 10th
  tone(speakerPin,698);
  delay(150);
  msg[0] = 'd';
  msg[1] = '0';

}
else if( (nut_to_fret_press3 > 16.67) && (nut_to_fret_press3 < 18.33) )
{
  //send F#5 11th
  tone(speakerPin,740);
  delay(150);
  msg[0] = 'd';
  msg[1] = '1';

}




//If string 4 is being picked
if( nut_to_fret_press4 == 0 )
{
  //send G4 open
  //tone(speakerPin,392);
  //delay(150);
  msg[0] = 'a';
```

```
  msg[1] = '0';
}
else if( (nut_to_fret_press4 > 33.33) && (nut_to_fret_press4 < 35) )
{
  //send G#4 1st
  //tone(speakerPin,415);
  //delay(150);
  msg[0] = 'a';
  msg[1] = '1';

}
else if( (nut_to_fret_press4 > 31.67) && (nut_to_fret_press4 < 33.33) )
{
  //send A4 2nd
  //tone(speakerPin,440);
  //delay(150);
  msg[0] = 'b';
  msg[1] = '0';

}

else if( (nut_to_fret_press4 > 30) && (nut_to_fret_press4 < 31.67) )
{
  //send A#4 3rd
  //tone(speakerPin,466);
  //delay(150);
  msg[0] = 'c';
  msg[1] = '0';

}
else if( (nut_to_fret_press4 > 28.33) && (nut_to_fret_press4 < 30) )
{
  //send B4 4th
  //tone(speakerPin,494);
  //delay(150);
  msg[0] = 'c';
  msg[1] = '1';

}
else if( (nut_to_fret_press4 > 26.67) && (nut_to_fret_press4 < 28.33) )
{
  //send C5 5th
  //tone(speakerPin,523);
  //delay(150);
  msg[0] = 'd';
  msg[1] = '0';

}
```

```
else if( (nut_to_fret_press4 > 25) && (nut_to_fret_press4 < 26.67) )
{
  //send C#5 6th
  //tone(speakerPin,554);
  //delay(150);
  msg[0] = 'd';
  msg[1] = '1';

}
else if( (nut_to_fret_press4 > 23.33) && (nut_to_fret_press4 < 25) )
{
  //send D5 7th
  //tone(speakerPin,587);
  //delay(150);
  msg[0] = 'e';
  msg[1] = '0';

}

else if( (nut_to_fret_press4 > 21.67) && (nut_to_fret_press4 < 23.33) )
{
  //send D#5 8th
  //tone(speakerPin,622);
  //delay(150);
  msg[0] = 'f';
  msg[1] = '0';

}

else if( (nut_to_fret_press4 > 20) && (nut_to_fret_press4 < 21.67) )
{
  //send E5 9th
  //tone(speakerPin,659);
  //delay(150);
  msg[0] = 'f';
  msg[1] = '1';

}
else if( (nut_to_fret_press4 > 18.33) && (nut_to_fret_press4 < 20) )
{
  //send F5 10th
  //tone(speakerPin,698);
  //delay(150);
  msg[0] = 'g';
  msg[1] = '0';

}
```

```
 else if( (nut_to_fret_press4 > 16.67) && (nut_to_fret_press4 < 18.33) )
 {
  //send F#5 11th
  //tone(speakerPin,740);
  //delay(150);
  msg[0] = 'g';
  msg[1] = '1';
 }

 noTone(speakerPin);

 //Send the data into the receiver
 vw_send((uint8_t *)msg, 2);
 vw_wait_tx();
 delay(150);
}


//aditional function
long microsecondsToInches(long microseconds)
{
 return microseconds / 74 / 2;
}

long microsecondsToCentimeters(long microseconds)
{
 return microseconds / 29 / 2;
}


float voltageToCentimeters(int voltage_in)
{
 return voltage_in * .0049;
}
```

Right Hand implementation

```
/*
 *
 * DDS Sine Generator mit ATMEGS 168
 * Timer2 generates the  31250 KHz Clock Interrupt
 *
 * KHM 2009 /  Martin Nawrath
 * Kunsthochschule fuer Medien Koeln
 * Academy of Media Arts Cologne
 * Modified by Satyo Iswara UIUC 2012
```

```
 */

#include "avr/pgmspace.h"
#include <VirtualWire.h>

// table of 256 sine values / one sine period / stored in flash memory
PROGMEM  prog_uchar sine256[]  = {

127,130,133,136,139,143,146,149,152,155,158,161,164,167,170,173,176,178,181,184,187,190,192,195,
198,200,203,205,208,210,212,215,217,219,221,223,225,227,229,231,233,234,236,238,239,240,

242,243,244,245,247,248,249,249,250,251,252,252,253,253,253,254,254,254,254,254,254,254,253,253,
253,252,252,251,250,249,249,248,247,245,244,243,242,240,239,238,236,234,233,231,229,227,225,223,

221,219,217,215,212,210,208,205,203,200,198,195,192,190,187,184,181,178,176,173,170,167,164,161,
158,155,152,149,146,143,139,136,133,130,127,124,121,118,115,111,108,105,102,99,96,93,90,87,84,81,
78,

76,73,70,67,64,62,59,56,54,51,49,46,44,42,39,37,35,33,31,29,27,25,23,21,20,18,16,15,14,12,11,10,9,7,6
,5,5,4,3,2,2,1,1,1,0,0,0,0,0,0,0,1,1,1,2,2,3,4,5,5,6,7,9,10,11,12,14,15,16,18,20,21,23,25,27,29,31,

33,35,37,39,42,44,46,49,51,54,56,59,62,64,67,70,73,76,78,81,84,87,90,93,96,99,102,105,108,111,115,1
18,121,124

};

#define cbi(sfr, bit) (_SFR_BYTE(sfr) &= ~_BV(bit))
#define sbi(sfr, bit) (_SFR_BYTE(sfr) |= _BV(bit))

int ledPin = 13;              // LED pin 7
int testPin = 7;
int t2Pin = 6;
byte bb;


double dfreq;
// const double refclk=31372.549;  // =16MHz / 510
const double refclk=31376.6;      // measured

// variables used inside interrupt service declared as voilatile
volatile byte icnt;             // var inside interrupt
volatile byte icnt1;            // var inside interrupt
volatile byte c4ms;             // counter incremented all 4ms
volatile unsigned long phaccu;   // pahse accumulator
volatile unsigned long tword_m;  // dds tuning word m
//add-on
volatile float reducer = 1;
volatile byte count;
```

```
volatile byte count2 = 0;
volatile byte adder = 0;

//flex sensor variable
int string_G = 0;
int string_C = 1;
int string_E = 2;
int string_A = 3;
double freq_G = 385;
double freq_C = 515;
double freq_E = 648;
double freq_A = 865;

//receiver pin
const int receive_pin = 3;
const int led_pin = 13;

//for dfreq
volatile byte ready = 1;
boolean pressed = false;

//dont change this setup
void setup()
{
 delay(1000);

 pinMode(ledPin, OUTPUT);      // sets the digital pin as output
 Serial.begin(9600);       // connect to the serial port
 Serial.println("DDS Test");

 pinMode(6, OUTPUT);     // sets the digital pin as output
 pinMode(7, OUTPUT);     // sets the digital pin as output
 pinMode(11, OUTPUT);     // pin11= PWM  output / frequency output

 Setup_timer2();

 // disable interrupts to avoid timing distortion
 //cbi (TIMSK0,TOIE0);            // disable Timer0 !!! delay() is now not available
 sbi (TIMSK2,TOIE2);            // enable Timer2 Interrupt

 //dfreq=1000;                // initial output frequency = 1000.o Hz
 tword_m=pow(2,32)*dfreq/refclk;  // calulate DDS new tuning word

 //for rx
 vw_set_rx_pin(receive_pin);
 vw_set_ptt_inverted(true);
 vw_setup(2000);
 vw_rx_start();
```

```
}

void loop()
{

  //rx
  uint8_t buf[VW_MAX_MESSAGE_LEN];
  uint8_t buflen = VW_MAX_MESSAGE_LEN;

  if(vw_get_message(buf, &buflen)){
    int i;
    digitalWrite(led_pin, HIGH);
    Serial.print("Got: ");
    for(i = 0; i < buflen; i++)
    {
      Serial.print(char(buf[i]));
      Serial.print(' ');
    }
    freq_G = determine_freq(char(buf[1]), char(buf[2]));
    freq_C = determine_freq(char(buf[4]), char(buf[5]));
    freq_E = determine_freq(char(buf[7]), char(buf[8]));
    freq_A = determine_freq(char(buf[10]), char(buf[11]));
    Serial.println();
    digitalWrite(led_pin, LOW);
  }
  //rx_end

  //determine frequency first

  //somehow we should store the buf[i]

//  freq_G = determine_freq(buf[1], buf[2]);
//  freq_C = determine_freq(buf[4], buf[5]);
//  freq_E = determine_freq(buf[7], buf[8]);
//  freq_A = determine_freq(buf[10], buf[11]);
  Serial.print("f_G :");
  Serial.println(freq_G);
  Serial.print("f_C :");
  Serial.println(freq_C);
  Serial.print("f_E :");
  Serial.println(freq_E);
  Serial.print("f_A :");
  Serial.println(freq_A);


  //flex_sensor
```

```
if (string_pick(string_G))
{
  ready = 0;
  //sbi (TIMSK2,TOIE2);
  dfreq = freq_G;
  Serial.println("string_g");
  pressed = true;
}
else if (string_pick(string_C))
{
  ready = 0;
  //sbi (TIMSK2,TOIE2);
  dfreq = freq_C;
  Serial.println("string_c");
  pressed = true;
}
else if (string_pick(string_E))
{
  ready = 0;
  //sbi (TIMSK2,TOIE2);
  dfreq = freq_E;
  Serial.println("string_e");
  pressed = true;
}
else if (string_pick(string_A))
{
  ready = 0;
  //sbi (TIMSK2,TOIE2);
  dfreq = freq_A;
  Serial.println("string_a");
  pressed = true;
}
//flex_end

Serial.print("f :");
Serial.println(dfreq);


//if (c4ms > 250) {              // timer / wait fou a full second
//  c4ms=0;

  //decide frequency             // determine the frequency played
  //dfreq=analogRead(0);           // read Poti on analog pin 0 to adjust output frequency from 0..1023 Hz
  //dfreq = 600;

  cbi (TIMSK2,TOIE2);            // disble Timer2 Interrupt
  tword_m=pow(2,32)*dfreq/refclk;  // calulate DDS new tuning word
  sbi (TIMSK2,TOIE2);            // enable Timer2 Interrupt
```

```
 if (pressed == true)
 {
  reducer = 1;
  pressed = false;
 }
 else
 {
  pressed = true;
 }

   //Serial.print(dfreq);
   //Serial.print("  ");
   //Serial.println(tword_m);
   //Serial.println(reducer);

    /*
    sbi(PORTD,6); // Test / set PORTD,7 high to observe timing with a scope
    cbi(PORTD,6); // Test /reset PORTD,7 high to observe timing with a scope
    */

  //}


}
//*************************************************************
// timer2 setup
// set prscaler to 1, PWM mode to phase correct PWM,  16000000/510 = 31372.55 Hz clock
void Setup_timer2() {

// Timer2 Clock Prescaler to : 1
  sbi (TCCR2B, CS20);
  cbi (TCCR2B, CS21);
  cbi (TCCR2B, CS22);

  // Timer2 PWM Mode set to Phase Correct PWM
  cbi (TCCR2A, COM2A0);  // clear Compare Match
  sbi (TCCR2A, COM2A1);

  sbi (TCCR2A, WGM20);  // Mode 1  / Phase Correct PWM
  cbi (TCCR2A, WGM21);
  cbi (TCCR2B, WGM22);
}

boolean string_pick(int x)
{
  int y = analogRead(x);
  //Serial.print("int val");
```

```
  //Serial.println(y);
  if ( y >= 430)
  {
    //Serial.println("not picked");
    return false;
  }
  else if ( (y <= 430) && (y >= 409) ) // 2.1 v
  {
    //Serial.println("picked");
    return true;
  }
  else if ( y < 409 )
  {
    return false;
  }
  else //general condition
  {
    return false;
  }
}

//determine the frequency played
double determine_freq(char note, char sharp)
{
  double freq;

  if (note == 'g')//G3
  {
    if (sharp =='0')
    {
      freq = 385.0;
    }
    else if(sharp == '1')
    {
      freq = 408.0;
    }
  }
  else if (note == 'a')
  {
    if (sharp =='0')
    {
      freq = 432.0;
    }
    else if(sharp == '1')
    {
      freq = 458.0;
    }
  }
```

```
else if (note == 'b')
{
  freq = 486.0;
}
else if (note == 'C')
{
  if (sharp =='0')
  {
    freq = 515.0;
  }
  else if(sharp == '1')
  {
    freq = 545.0;
  }
}
else if (note == 'D')
{
  if (sharp =='0')
  {
    freq = 578.0;
  }
  else if(sharp == '1')
  {
    freq = 612.0;
  }
}
else if (note == 'E')
{
  freq = 648.0;
}
else if (note == 'F')
{
  if (sharp =='0')
  {
    freq = 687.0;
  }
  else if(sharp == '1')
  {
    freq = 728.0;
  }
}
else if (note == 'G')//G3
{
  if (sharp =='0')
  {
    freq = 771.0;
  }
  else if(sharp == '1')
```

```
  {
    freq = 817.0;
  }
}
else if (note == 'A')
{
  if (sharp =='0')
  {
    freq = 865.0;
  }
  else if(sharp == '1')
  {
    freq = 917.0;
  }
}
else if (note == 'B')
{
  freq = 971.0;

}
else if (note == 'H')
{
  if (sharp =='0')
  {
    freq = 1029.0;
  }
  else if(sharp == '1')
  {
    freq = 1090.0;
  }
}
else if (note == 'I')
{
  if (sharp =='0')
  {
    freq = 1155.0;
  }
  else if(sharp == '1')
  {
    freq = 1224.0;
  }
}
else if (note == 'J')
{
  freq = 1297.0;
}
else if (note == 'K')
{
```

```
  if (sharp =='0')
  {
    freq = 1374.0;
  }
  else if(sharp == '1')
  {
    freq = 1456.0;
  }
 }
 else if (note == 'L')//G3
 {
  if (sharp =='0')
  {
    freq = 1542.0;
  }
  else if(sharp == '1')
  {
    freq = 1634.0;
  }
 }
 else
 {
  //prevent error f != 0;
  freq = 600;
 }

 return freq;
}

//****************************************************************
// Timer2 Interrupt Service at 31372,550 KHz = 32uSec
// this is the timebase REFCLOCK for the DDS generator
// FOUT = (M (REFCLK)) / (2 exp 32)
// runtime : 8 microseconds ( inclusive push and pop)
ISR(TIMER2_OVF_vect) {

 sbi(PORTD,7);        // Test / set PORTD,7 high to observe timing with a oscope

 phaccu=phaccu+tword_m; // soft DDS, phase accu with 32 bits
 icnt=phaccu >> 24;    // use upper 8 bits for phase accu as frequency information
                       // read value fron ROM sine table and send to PWM DAC

 //add-on
 count = count++;
 count2 = count2++;

 if(count == 100)
 {
```

```
   if ((reducer == 127))
   {
     reducer = 1;
   }
   reducer = reducer + .2;
   count = 0;
   count2 ++;
   adder = 127 - (127/reducer);
  }

  if (reducer > 68 )
  {
   //ready to receive a new note
   ready = 1;
   //cbi (TIMSK2,TOIE2);
  }
  else
  {
   OCR2A=(pgm_read_byte_near(sine256 + icnt)/reducer)+adder;
  }

  //OCR2A=(pgm_read_byte_near(sine256_2 + icnt) >> reducer)+127;//"/2;" to reduce Vout to half
//determine the output analog voltage directly
                                  //idea divided with speed that depends on the frequency

  if(icnt1++ == 125) {  // increment variable c4ms all 4 milliseconds
   c4ms++;
   icnt1=0;
  }

 cbi(PORTD,7);          // reset PORTD,7
}
```

Appendix D Note Encoding

Note	real frequency(Hz)	programmed frequency(Hz)	Comment
G_3	196	385.532	open freq string_g
$G^{\#}_3/A^{b}_3$	207.65	408.4476	
A_3	220	432.74	
$A^{\#}_3/B^{b}_3$	233.08	458.4684	
B_3	246.94	485.731	
C_4	261.63	514.6262	open freq string_c
$C^{\#}_4/D^{b}_4$	277.18	545.2131	
D_4	293.66	577.6292	
$D^{\#}_4/E^{b}_4$	311.13	611.9927	
E_4	329.63	648.3822	open freq string_e
F_4	349.23	686.9354	
$F^{\#}_4/G^{b}_4$	369.99	727.7703	
G_4	392	771.064	
$G^{\#}_4/A^{b}_4$	415.3	816.8951	
A_4	440	865.48	open freq string_a
$A^{\#}_4/B^{b}_4$	466.16	916.9367	
B_4	493.88	971.462	
C_5	523.25	1029.233	
$C^{\#}_5/D^{b}_5$	554.37	1090.446	
D_5	587.33	1155.278	
$D^{\#}_5/E^{b}_5$	622.25	1223.966	
E_5	659.26	1296.764	
F_5	698.46	1373.871	
$F^{\#}_5/G^{b}_5$	739.99	1455.56	
G_5	783.99	1542.108	
$G^{\#}_5/A^{b}_5$	830.61	1633.81	