ThereminFreaks - Theremin Rhythm Game

By

Esteban Looser-Rojas

Daniel Olivas Hernandez

Michael Recklein

Final Report for ECE 445, Senior Design, Spring 2019

TA: Amr Martini

1 May 2019

Project No. 34

Abstract

The aim of this project was to develop a rhythm game based on the concept of the theremin, an electronic instrument whose pitch and volume change depending on the distance between the user's hands and the instrument's antennae. A theremin was built and its pitch and volume signals were turned into voltage levels passed through ADCs. A PC video game was also developed that uses these sampled signals, displaying a thin rope-like 3D object as the desired pitch (X axis) and volume (Y axis), with the Z axis representing time. In order to simulate the sound coming from a theremin, the two ADC samples were also used in a small program developed to synthesize music. For the most part the software and the digital circuitry met all requirements, although much research and development was necessary to design and iterate upon the oscillators in the theremin.

Contents

1. Introduction

There are many different kinds of rhythm games on the market, stretching back all the way to the 90s and early 2000s with games that operate based on dance panels one must step on and controllers shaped like actual instruments. They typically involve the player pressing a button or a touch screen and/or simulating a conventional rock instrument like a guitar or a drum set. They also typically only allow discrete notes, and when there is an analog or continuous control, it's usually not very precise, only measuring whether or not one is actuating the control in a certain direction. In other words, it might as well function like a button, like the turntable in beatmania IIDX by Konami, where the player simply has to push the turntable clockwise or counter-clockwise in order for the game to register it as a "scratch" [1]. The closest thing to what we planned on making is the singing mode in Rock Band, based off a previous game called Karaoke Revolution, which only takes the user's voice pitch into account [2]. Uniqueness of this rhythm game therefore rests on its being based off an unconventional instrument where there is no contact between the player and the antennae, and the pitch and volume being continuous rather than being discrete like a piano. Rhythm games with realistic controllers also have the advantage of teaching people how the instruments they simulate are played, and interactive and game-based instruction on how to play a theremin is lacking.

Our objective is then to create a theremin-based rhythm game for the PC platform. The hardware component is a PC peripheral theremin where the capacitance is controlled by moving one's hands closer or farther from the antenna-like plates on the theremin controller. This capacitance affects an oscillator's frequency by a range of several kilohertz. Mixing it with a reference oscillator, one can take the difference of the two frequencies to arrive at the frequency change induced by the antenna to effectively measure the distance between the hand and the antenna. Capturing this distance as a variable on the PC involves sending an ADC sample of it through USB using a microcontroller. On the software side, a simple driver for this controller sends the samples to a video game written in C++ using OpenGL. The game displays a continuous stream of notes coming at the player, with the vertical axis representing the volume and the horizontal axis representing pitch. A sound synthesis engine works alongside the game to simulate how a theremin would sound at a given pitch and volume. In the end, it will be the success of the design of the oscillators that determines the functionality of the game, as all software depends on the noise immunity and stability of this theremin component.

2 Design

The most high-level way to split the design is to conceive of it as a hardware section - the theremin controller - split into an analog and a digital circuit, and a software section composed of different threads of execution.

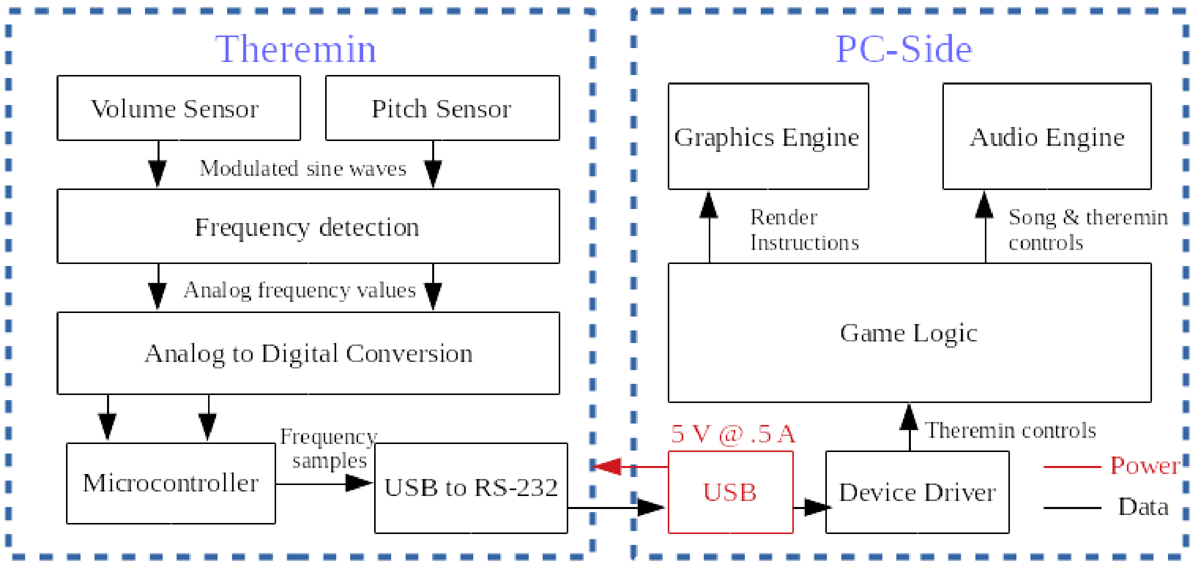


Figure 1: Block Diagram

The analog circuit consists of the pitch and volume sensors, which effectively capture the distance between the player's hand and the antennae, and a frequency detection circuit that translates this into DC voltage values. The digital circuit converts this DC voltage into a 16-bit digital integer with ADCs and sends the data to the game using a USB connection set up as a virtual COM port. A small program receives these two 16-bit integers and sends them to the video game to tell it where the user's hands are and what the pitch and volume the audio engine should be using.

2.1 Volume and Pitch Sensors

The volume and pitch sensors consist of a reference oscillator and an antenna oscillator as well as a mixer and integrator. The reference oscillator and the antenna oscillator are almost identical, except that the antenna oscillator is connected to the antenna such that it is sensitive to the antenna capacitance. Both oscillators are LC oscillators tuned to the same frequency, but due to the antenna capacitance the antenna oscillator oscillates at a slightly lower frequency. The outputs of the two oscillators are multiplied together using a Gilbert cell, which results in a new signal with two frequency components. One component is the sum frequency, and the other is the difference or beat frequency. The Sum frequency is not used for sensing and it is filtered off by the integrator. The difference frequency is a low frequency since the two oscillators oscillate at a similar frequency. This is the signal that is used for sensing the position of the players hands. When the player moves his hands closer to the antennas the capacitance of the antennas increases and the oscillator that is connected to that antenna will oscillate at a lower frequency. This change is very small compared to the oscillator frequency so it is not practical to detect it by direct frequency measurement. The difference in frequency between the two oscillators varies by the same amount as the antenna oscillator frequency, and since the difference frequency is much lower than the frequency of the antenna oscillator it is easy to detect the small change by measuring this frequency. To measure the difference frequency we use an integrator, in order to avoid having to sample the signal at a high rate. The frequency response of an integrator circuit is proportional to the inverse of the frequency, so as the player moves his hands closer to the antenna and the difference frequency increases the output amplitude of the integrator will decrease. The output of the integrator is connected to a detector circuit, which produces a DC voltage (DC when the players hands are not moving, otherwise this voltage does change) proportional to the amplitude of the integrator output, which is in turn inversely proportional to the difference frequency. This DC voltage is what we use to determine what pitch or volume is being played.

2.1.1 Antennae

The antennae designs were taken out of an online PDF of a magazine article written by Robert Moog where he describes how to construct a DIY theremin [3]. He recommends using ⅜" diameter copper pipe for both antennae, with a straight copper tube for the pitch antenna and a curved pipe for the volume antenna. His provided rationale for the particular shapes is that, "The pitch antenna is straight because this configuration is more sensitive to changing hand position when the hand is farther away and less sensitive when the hand is close. The change in hand capacitance is extremely small when the hand is far away, and the change in pitch as a function of distance must be as uniform as possible. The volume antenna is looped because this configuration is less sensitive when the hand is far away and more sensitive when the hand is close. This gives you greater control over the low end of the dynamic range and lets you articulate notes by quickly dipping your left hand into the loop". Our antennae ended up with dimensions of two feet for the pitch antenna and 6" between the ends of the volume antenna and 10" from the ends to the edge of the loop. Although we were worried about the antennae being too small to change the oscillators by a significant enough amount, what ended up happening was they were too big and messed with the stability of the oscillators. A shorter length is thus recommended.

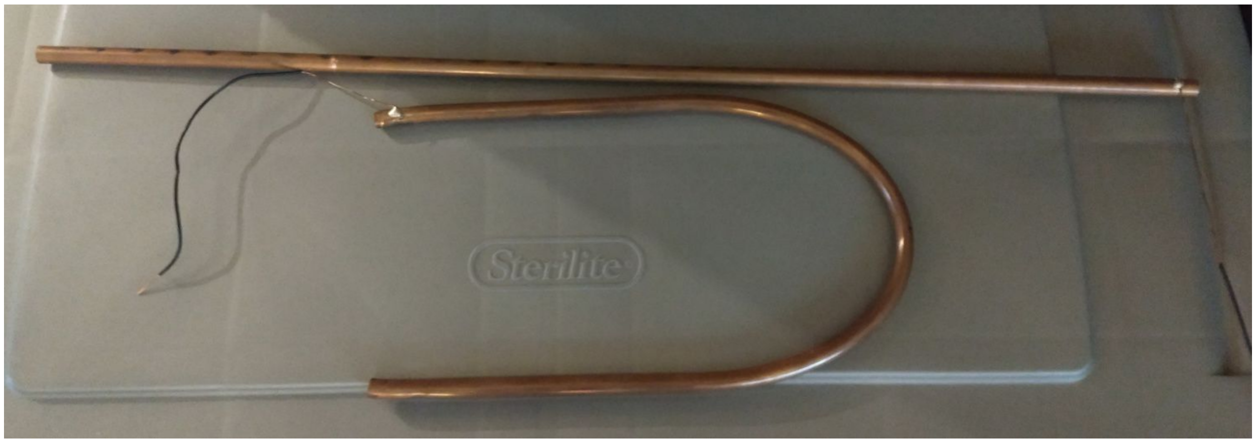


Figure 2: Theremin Antennae

2.1.2 Oscillators

The oscillators were the most sensitive part of the entire project. Coincidentally, any sort of noise or instability that would result from how the oscillators were designed would find its way into the video game and the sound engine, introducing a jittery reticule and unpleasant sounds, respectively. In the end, however, what was used before the deadline was a Colpitts oscillator, which uses a single 27 µH through-hole inductor in parallel with two 240 pF capacitors in series to create a tank circuit that oscillates at a frequency of 2.80 MHz, according to Formula 1:

$$f = \frac{1}{2\pi\sqrt{LC}} \tag{1}$$

where *f* is the oscillator's frequency, *L* is the inductance of the inductor, and *C* is half the capacitance of each capacitor (assuming the two capacitors are identical). In order to increase the reliability of the circuit, the oscillators were fed into Schmitt trigger inverters in order to convert their signals into 5V square waves. To reduce the frequency of the oscillators we also fed the inverter outputs into flip-flops with the inverting output fed back into the input and the oscillator output attached to the clock, effectively halving the frequency of the oscillators.

In our original design we used Hartley oscillators, which is typical for a theremin design. Due to the characteristics of the transformers we used our oscillator design produced a very weak output signal. In order to solve this problem we changed our design to use Colpitts oscillators. The two types of oscillators are very similar, the main difference is that the Hartley oscillators use a tap on the transformer to obtain the necessary feedback, while in the colpitts oscillator feedback is obtained from a capacitor divider. The capacitors in the divider are the same capacitors used for tuning the LC circuit, thus to calculate the frequency the effective capacitance is the capacitance of the two capacitors in series. In our case both capacitors were the same so the capacitance was just half the capacitance of each capacitor. There is a third capacitor that is connected to the emitter of the transistor, in the Hartley oscillator this capacitor is required to block the DC voltage that would otherwise affect the biasing, however in the colpitts oscillator this capacitor is not needed (if removed it should be replaced with a short circuit, not left open). Since the Colpitts oscillator takes feedback from the capacitor divider and not a tap on the inductor, it can be constructed with a simple inductor as opposed to a transformer. This makes it easier to use since we don't have to find a suitable transformer and characterize it.

The oscillators were the source of most of our problems, even after we fixed the oscillators to give a suitable amplitude our design didn't work because the oscillators would stop oscillating or become unstable when the antennas were connected. We carried out many experiments to find the cause of this problem, and found that the problem was due to having a low input voltage to power the oscillators. Our solution to this was a modified Hartley oscillator, shown in Figure 3, that used a current source to bias the transistor as opposed to a simple resistor. Our implementation of a current source was a typical current mirror implemented with 2 BJT transistors. We never got a chance to build a complete theremin using this design, however we did build several prototypes and ran LTSPICE simulations to confirm that the design works. The simulation results as well as the experiment data show that using this biasing technique we can achieve much higher output amplitude as well as higher stability. The same biasing technique can be used on Colpitts oscillators too.

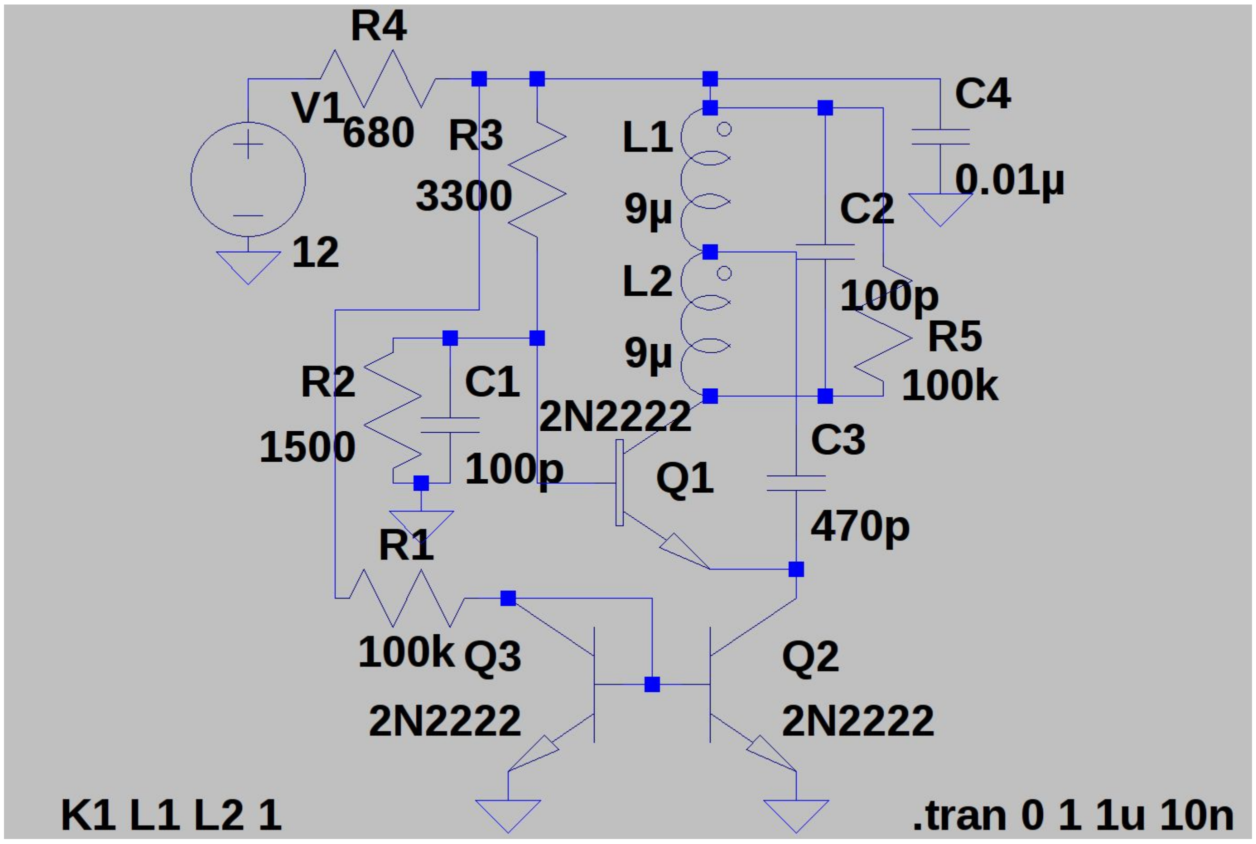


Figure 3: Modified Hartley Oscillator

2.1.3 Mixers

The mixers are used to find the difference in frequency between the antenna oscillator and the reference oscillator. Finding the difference frequency between the antenna and reference oscillators makes it much easier to detect small frequency changes, since the difference frequency is much smaller than the oscillator frequency but changes by the same amount. Our original design used Gilbert cells, but we then changed to using an XOR gate mixer to make the design more reliable. The Gilbert cells have two inputs and one output, all of which are differential signals. The output is simply the product of the two inputs. This was used as the mixer in our original design. Later versions of the design use an XOR gate as the mixer. This works in a slightly different way since the inputs are now square waves (and so is the output), but we still get a frequency component that is the difference between the two oscillator frequencies. The sum frequency as well as all of the harmonics are filtered off by the integrator, which has a low response to high frequencies. The output that we get from the XOR gate is essentially a PWM signal. Its frequency is the sum of the two oscillator frequencies, and the duty cycle oscillates at a rate given by the difference of the two oscillator frequencies.

2.2 Frequency Detection

In order to measure the difference frequency between the antenna oscillator and the reference oscillator we first integrate the signal from the mixer. The integrator removes the unwanted high frequency components and gives us a known frequency response for the difference frequency.

Assuming that the output of the mixer has a constant amplitude, we can determine what frequency is if we know the frequency response of the integrator and can determine the output amplitude of the integrator. A detector is used to measure the amplitude of the output of the integrator, and we know the the frequency response of the integrator is inversely proportional to the frequency. The constant proportionality factors depend on the value of the integrating capacitor as well as the gain of the integrator circuits used. With this information we can determine the change in frequency of the antenna oscillator by just knowing the output amplitude of the integrator.

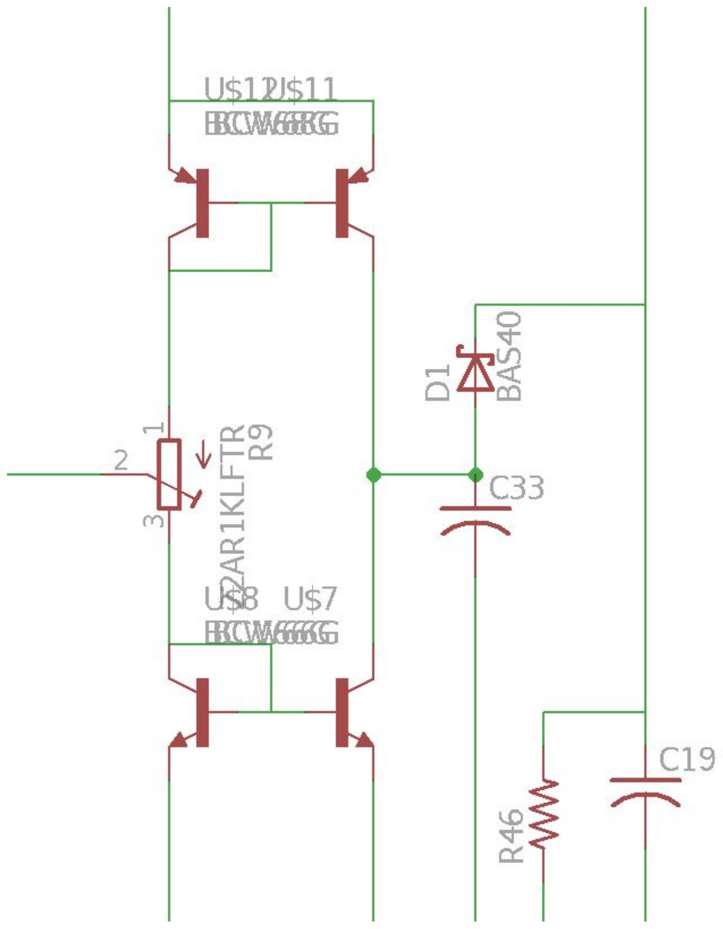


Figure 4: Integrator circuit (left) and detector circuit (right)

2.2.1 Integrator

The integrator serves as a low pass filter and also to give us a known frequency response that we can use for frequency detection. In our design the integrator does not need to have a high bandwidth since the frequencies that we are interested in are in the audio range. It is important however that the integrator is able to function properly when high frequency components are present at the input, since we rely on the integrator to filter out the sum frequency of the two oscillators. Typical integrator circuits use an op-amp, however most op-amps are not designed to operate from a single 5 volt supply and will not work correctly when the output voltage is close to ground or the supply voltage. In order to avoid these problems we used our own integrator design that relies on current mirrors to function. First we have two current mirrors connected such that we get an output current that is independent of the output voltage, and is proportional to the input voltage. For simplicity we assume that these are ideal current mirrors and the output current is exactly the same as the input current. This is a very good approximation if the transistors have a high gain. The current mirrors have a low input impedance, so if we connect a resistor in series with the current mirror input the input current will be approximately the input voltage divided by the resistance value. To get both positive and negative current output we need two current mirrors, one implemented with PNP transistors and one with NPN transistors. With this circuit we can effectively convert a voltage source into a current source, and if we then connect the output current to a capacitor the voltage across the capacitor with be the time integral of the current output (times a 1/C proportionality constant). This circuit proved to be very effective, however we found that it was extremely sensitive to temperature differences between the transistors that make up the current mirrors. Solving this problem was just a matter of adding a very small negative feedback to the circuit. Since our voltage to current converters were inherently inverting, this was done by connecting a high value resistor between the input and output of the integrator. The negative feedback ensures that the DC output voltage is not significantly affected by temperature, and is always near half of the supply voltage. We are aware that the negative feedback makes the output current dependent on output voltage but this non ideality was acceptable.

Note: the exact current output of the current mirrors is given by Iin*B/(B+2), where Iin is the input current and B is the gain of the transistors used. For high gain transistors we can assume the output current is the same as the input current.

2.2.2 Detector

The detector is used to measure the output amplitude of the integrators. Our detector design consisted of a diode and capacitor. The diode would conduct when the voltage from the integrator is higher than the voltage on the capacitor, thus keeping the capacitor charged at the peak voltage of the integrator output. a resistor in parallel with the capacitor ensures that the capacitor voltage drops is the ouput amplitude of the integrator is reduced. This resistor and capacitor form an RC filter, the resistor and capacitor values are chosen so that the AC signal from the integrator is blocked, but the capacitor voltage is still able to change quickly enough to make the theremin responsive to fast movements of the players hands.

2.3 Analog-to-Digital Conversion

The chosen analog-to-digital converters were off-the-shelf integrated circuits manufactured by Maxim Integrated. They have 16-bit accuracy and have a sampling rate of 165k samples/s, along with being controlled using four control signals and an 8-bit data bus. Because the number of pins on our microcontroller were limited, we couldn’t use the one with the 16-pin data bus. Other candidates for ADCs communicated serially or using I^2C, although getting those to work would require more work on our part programming the microcontroller. Additionally, the high sampling rate seemed like a good choice in order to minimize input lag, taking up only 4.7 µs for a conversion to take place [4].

2.4 Microcontroller

The requirements for the microcontroller were very simple in our design. The only task that the microcontroller has to do is read data from the analog to digital converters and send it the the USB to RS232 converter, using the RS232 protocol. We decided to use the PIC16C65 microcontroller because we were already familiar with the PIC16 architecture and we had already written the code that implements an RS232 interface. Although the PIC16C65 is a very old part it was more than capable of completing the tasks that we needed, and it is packaged in a CERDIP40 package that is easy to solder and can be used on a breadboard. It also had enough IO pins that we could use ADCs with a parallel interface to further simplify our job.

2.5 USB-to-RS-232 Interface

To simplify our project we used the CP2102 USB to RS232 converter. This chip is very similar to the FTDI chips that are commonly used for USB connectivity in low cost devices. The CP2102 communicates with our microcontroller through an RS232 interface, and connects to a PC via USB 2.0. On the PC the driver provided by Silicon Labs creates a virtual COM port that the software can use. Any data that the microcontroller sends will be received by the software on the PC through the virtual COM port, and likewise data written to the COM port will be received by the microcontroller. This means we don't have to worry about the USB protocol, and we also don't have to worry about writing to any internal registers on the CP2102. On both the hardware and the software side of our project the CP2102 lets us pretend that we are using an RS232 port to connect our project.

2.6 Device Driver

The device driver is a simple C program that performs several system calls on the file representing the serial port. On a Linux machine, when the theremin is plugged in it will either show up as /dev/ttyACMx or /dev/ttyUSBx, with *x* simply being an integer that the system decides to use. The job of the device driver is to open up the serial port file for reading and writing, write a byte to it to signal to the MCU to initialize sampling and conversion, and read four bytes back. The four bytes it receives are the low and high bytes of the two 16-bit ADC samples, with the first two being the pitch sample and the last two being the volume sample. The two samples are then scaled to a floating-point value so the game can use them effectively.

2.7 Game Logic

The software for the ThereminFreaks game was designed in order to abstract as much of the rendering process away so that, should someone else need to build on the software and add more features, they could do so easily without needing to know the inner workings of rendering, thus speeding up their development process. The function used to calculate the score was designed in such a way to account for noise from the theremin controller and to allow people with unstable hands to still be able to get a perfect score, while still having a fair drop-off if the player is playing an incorrect note or playing at the wrong volume, altogether. If the player is within .5 units, they can obtain a perfect score, as shown in Figure 5 displaying a MATLAB visualization of the scoring function. In terms of pitch this amounts to a quarter-tone, or 2±1/24 times the correct pitch. In terms of volume this amounts to halfway between forte and fortissimo or halfway between mezzo piano and mezzo forte.

```
f = Min[ 1/(Sqrt[x ^ 2 + y ^ 2] + .5) ^ 3 , 1]
Plot3D[f , {x , -1 , 1} , {y , -1 , 1} , PlotRange → All]
```

$$\mathrm{Min}\left[1, \frac{1}{\left(0.5+\sqrt{x^2+y^2}\right)^3}\right]$$

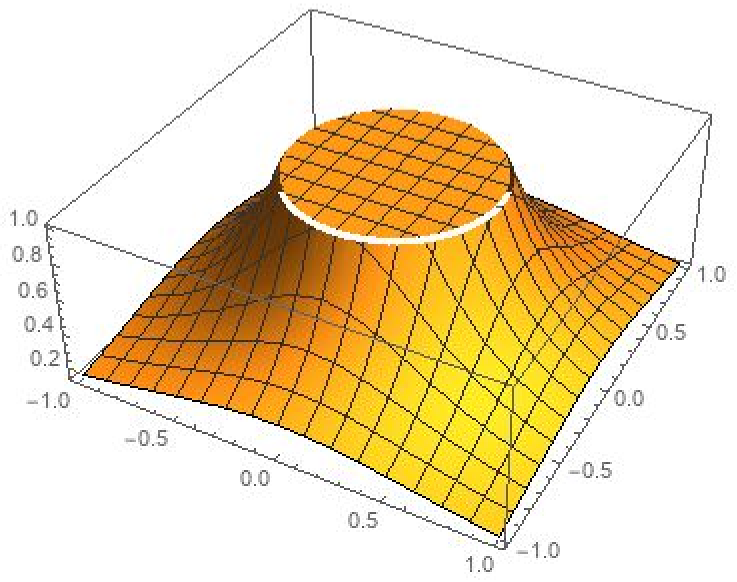


Figure 5: Scoring Function Visualization

2.8 Graphics Engine

To save on computation while rendering, any given song mesh is generated as a single mesh class instance, instead of separating the song mesh into multiple mesh instances at each break in the music. This was done in order to capitalize on the concept of batch rendering, where multiple meshes are combined into a single mesh structure so that only one write to GPU memory is necessary. To avoid the speed of the game from depending on the framerate at which it is running, the program calculates the amount of time it takes to render a frame and multiplies the distance that the mesh needs to move in one second by that time step. This decoupling of the frame rate from the speed of the game is important, as it ensures that the motion of the mesh and the timing of the game is uniform regardless of frame rate jittering.

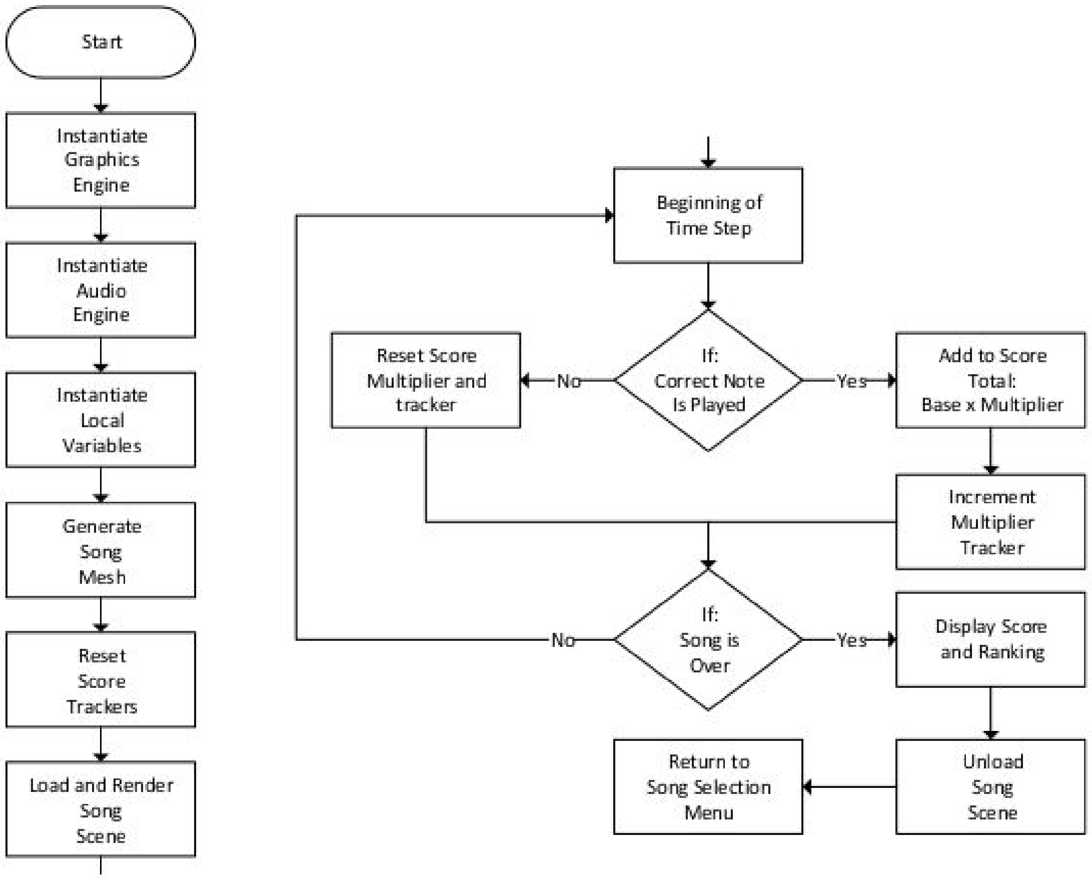

Start
Instantiate Graphics Engine
Instantiate Audio Engine
Instantiate Local Variables
Generate Song Mesh
Reset Score Trackers
Load and Render Song Scene
Beginning of Time Step
If: Correct Note Is Played
No
Yes
Reset Score Multiplier and tracker
Add to Score Total: Base x Multiplier
Increment Multiplier Tracker
If: Song is Over
No
Yes
Display Score and Ranking
Unload Song Scene
Return to Song Selection Menu

Figure 6: Game Flowchart

2.9 Audio Engine

In order to simulate the sound of a theremin, we used our knowledge from two ECE classes: digital signal processing and electronic music synthesis. A simple way to synthesize a musical instrument's sound is through additive synthesis, where one takes multiple sine wave generators and adds them together to generate a desired sound. In order to implement this, we searched for a relatively clean sample of a theremin being played: an MP3 file of a song called "Jungle Lullaby", obtained from a website called ThereminVox [5]. Using an audio editor called Audacity, we selected what appeared to be a single note being played and took the FFT of it.

What was noticed from this analysis and analyses of other portions of the song is that the theremin follows a general pattern of a fundamental tone with around 7 overtones that roll off in amplitude by around 1.5 until reaching a partial whose amplitude is so low that we figured it wouldn't be necessary to include in the synthesis.

In order to implement this using solely a laptop running Linux and no specialized DSP hardware, we used a C++ library called The Synthesis Toolkit in C++ (STK) [6]. This library was very straightforward to use; it allowed us to create a sinusoid object for the fundamental tone and seven more overtones in a C++ vector that were simply integer multiples of this sinusoid object. If any of the overtones went over the audio frequency of 20 kHz we simply didn't instantiate them. Playing this program, recording it with Audacity, and taking the FFT of it, we managed to get a result, shown on the right in Figure 7, that appeared similar to the sampled theremin note, shown on the left.

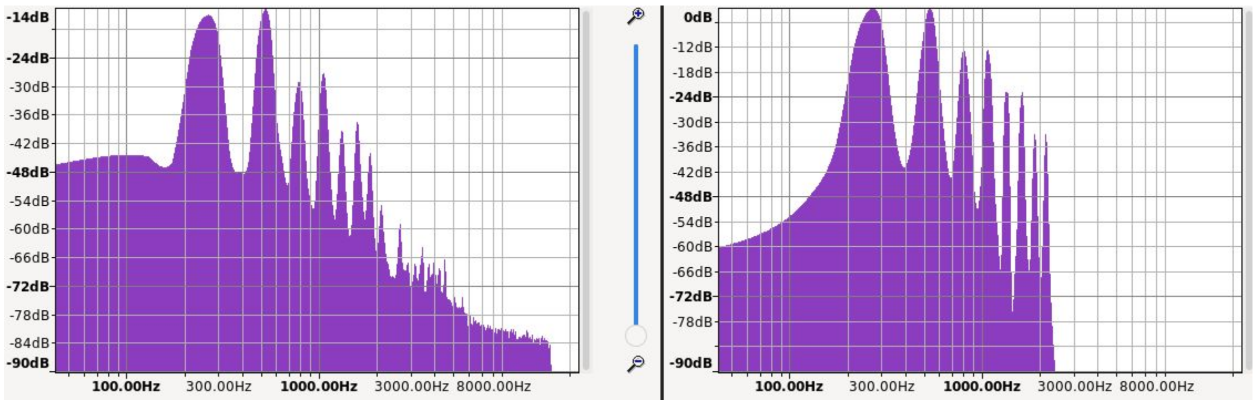


Figure 7: Comparison between sampled theremin spectrum and synthesized audio spectrum

All that was left was to use the samples from the theremin controller to attenuate the output using a floating-point number from 0.0 (muted) to 1.0 (full volume) and to scale the frequency from 110 Hz (A2 note) to 1710 Hz (A6 note).

3. Design Verification

3.1 Volume and Pitch Sensors

3.1.1 Oscillators

In order to see if the antennae would be able to change the frequency by a significant amount, it was necessary to connect an antenna to an oscillator and see by how much the oscillator's frequency changed. Specifically, an antenna was connected to the collector of the oscillator, which would change the resonance frequency of the tank circuit. With no hand near the antenna, the oscillator stayed at around 1.492 MHz, as seen in Figure 8. Upon moving a hand right next to the antenna, the oscillator changed frequency to around 1.486 MHz, shown in Figure 9, showing a difference of around 6 kHz. This shows that the capacitance in the tank circuit increased enough to decrease the oscillator's frequency. We only need around 5-10 kHz difference between the base frequency and the minimum frequency since the frequency scale can be adjusted in software.

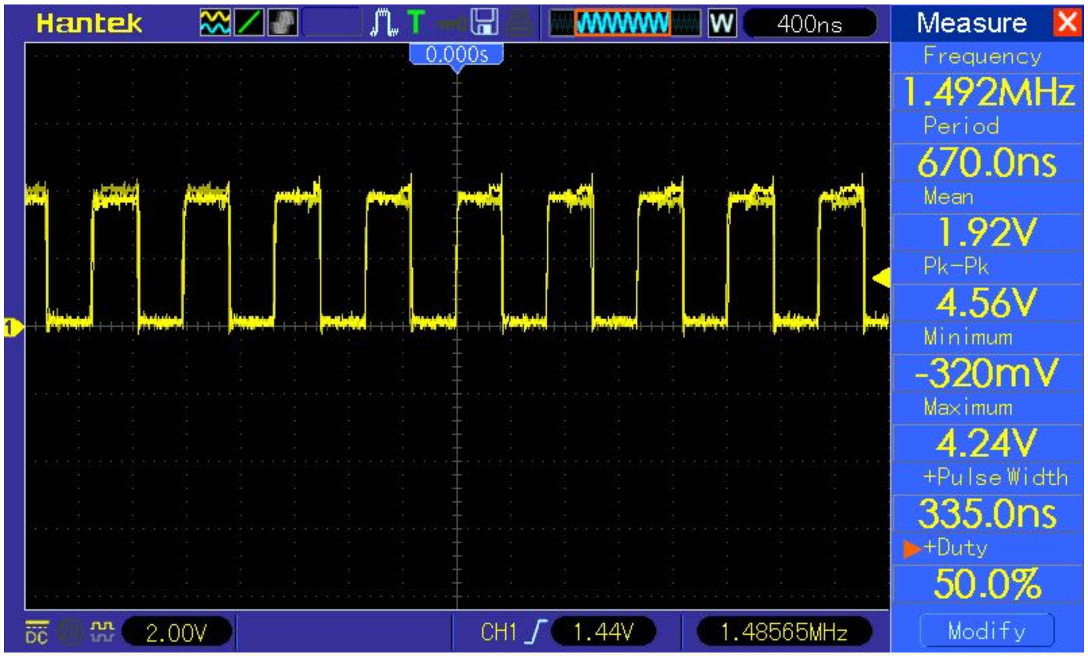


Figure 8: Oscillator with no hand near it

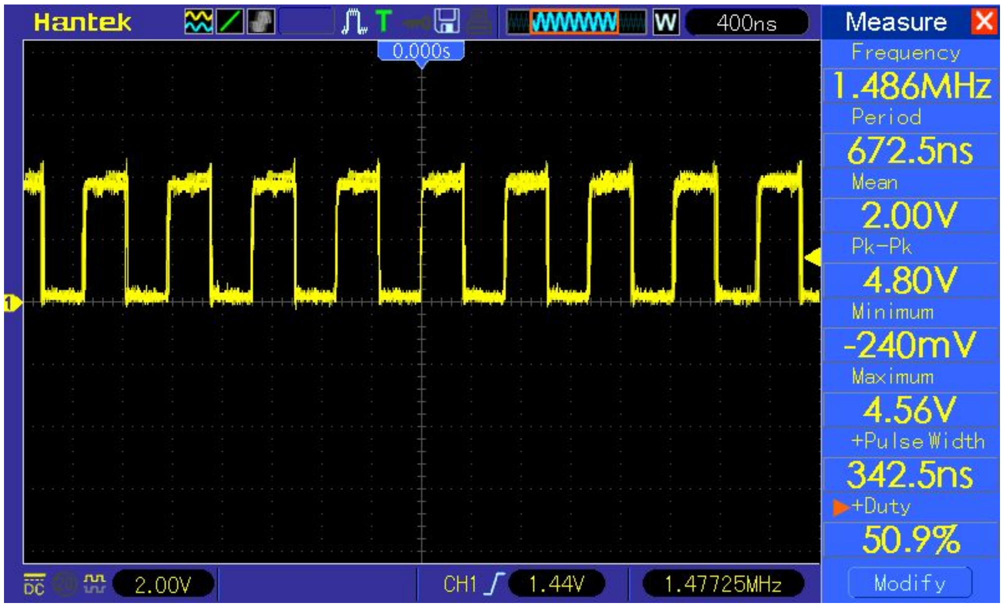


Figure 9: Oscillator with hand near it

3.1.2 Mixers

Connecting two function generators outputting square waves, one at 3.2 MHz and the other at 2.8 MHz, to an XOR gate would theoretically produce a modulated square wave of frequency 400 kHz as the output. As one can see in Figure 10, that is exactly what happened when we tested our XOR mixer concept. The time scale is 400 ns and the first fully-visible period of the low-frequency oscillation takes up around 6.25 time steps. Multiplying these out we get a period of 2.5 µs, which corresponds to 400 kHz.

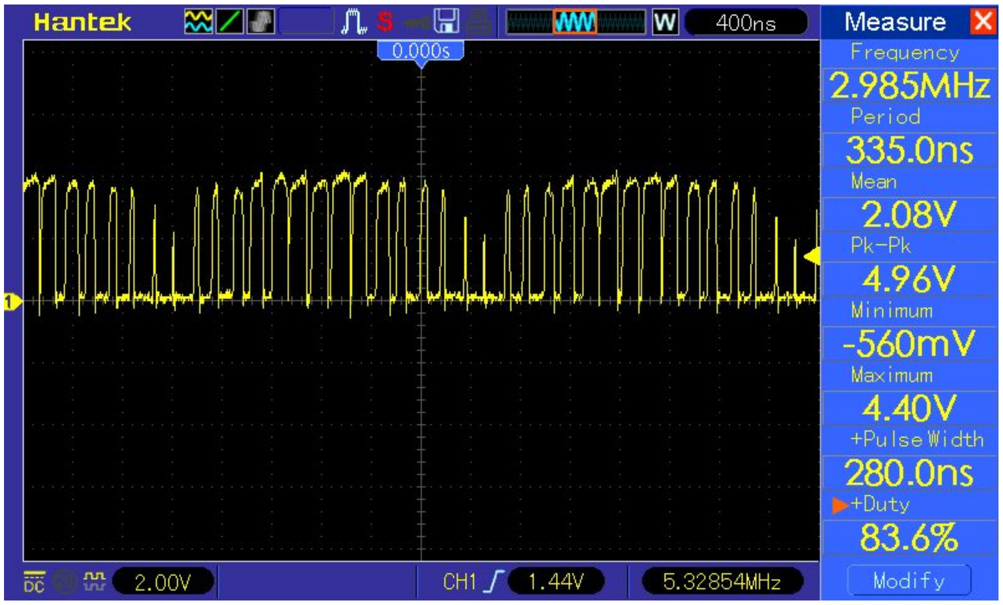


Figure 10: Result of XORing 2.8 MHz and 3.2 MHz square waves

3.2 Frequency Detection

To test the frequency detection circuit, we connected a function generator outputting a sine wave and monitored the output with an oscilloscope. The function generator was swept throughout the audio range and beyond, specifically from 20 Hz to 30 kHz. This was done in order to tell if the integrator and detector circuit would work with both extremes of what the mixer might output. At 20 Hz and 50 Hz the integrator and detector output a signal that averages to around 4 V. Pictured in Figure 11 is the output at 20 Hz. The integrator and detector then show smaller signals when the frequency is increased, from 3.36V at 200 Hz to 2.48V at 2 kHz and finally to 2.32V at 30 kHz, showing the inverse dependency on frequency that the integrator has. Pictured in Figure 12 is the output at 30 kHz.

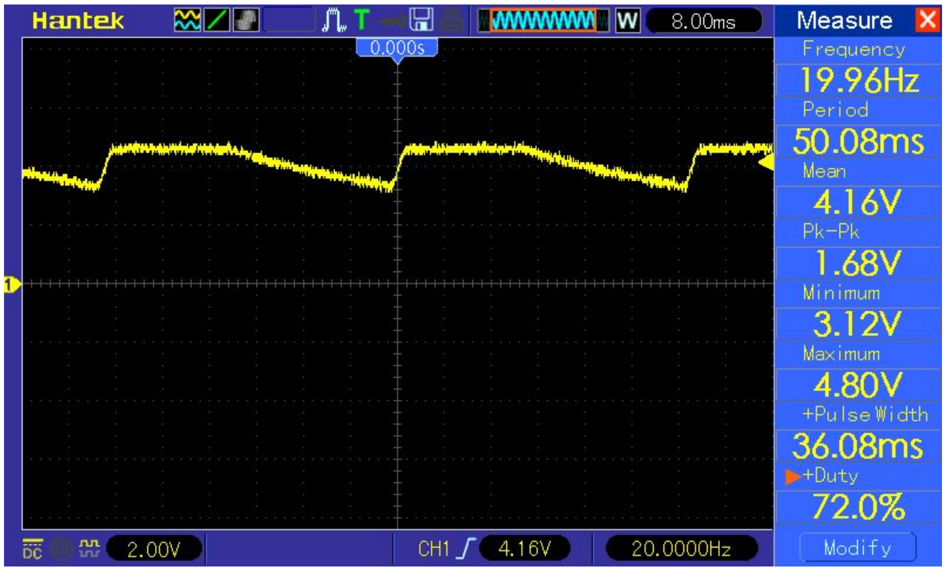


Figure 11: Output of frequency detection circuit at 20 Hz

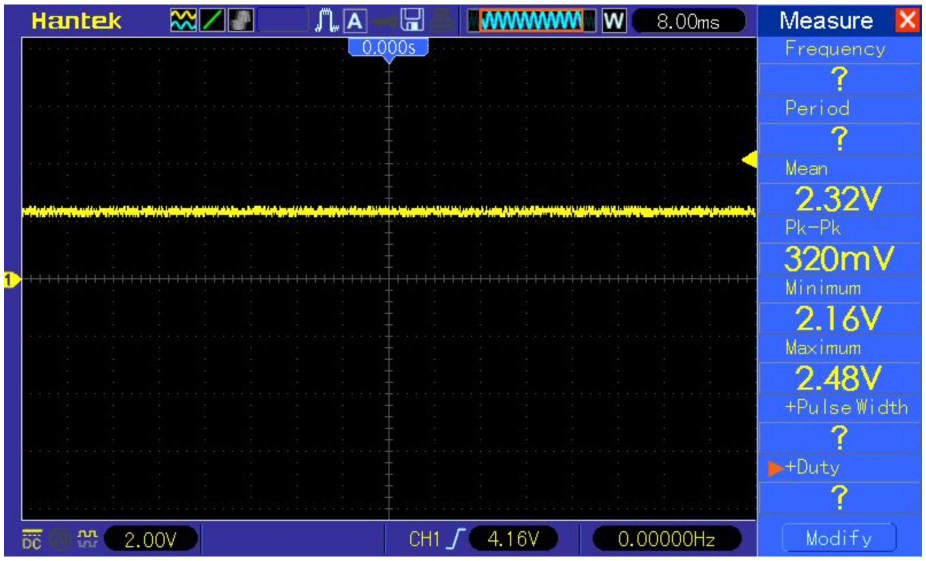


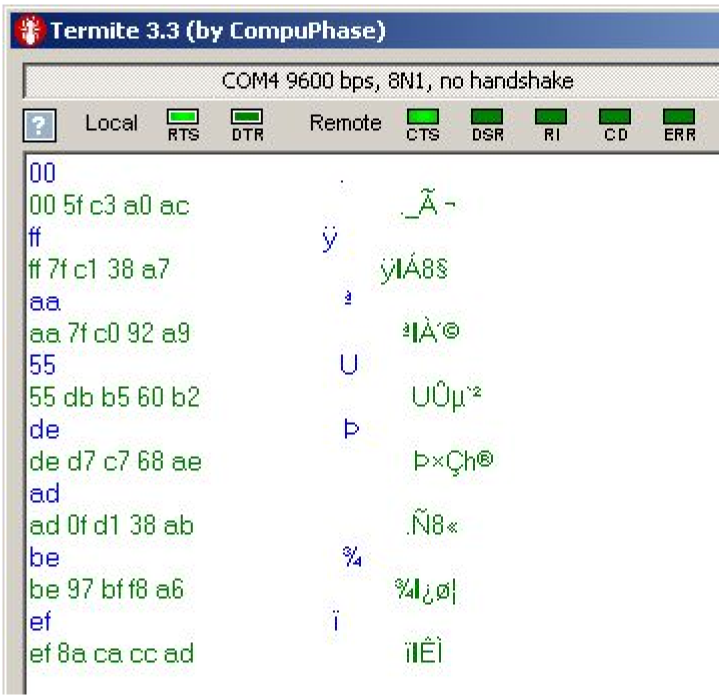
Figure 12: Output of frequency detection circuit at 30 kHz

3.3 Analog-to-Digital Conversion

Testing the ADCs separately involved soldering a spare chip onto a breakout board to connect it to a breadboard. From there, we used wires to connect the pins on the ADC to an Arduino, taking up twelve pins on the Arduino: eight for the data bus and four additional ones for the control signals. Some code was written on the Arduino that followed the setup and acquisition process for the ADC, detailed in its datasheet [4]. The basic summary of the program is that it would send a 16-bit sample to a PC connected to the Arduino by manipulating the ADC and storing the result from the 8-bit data bus rather than the analog pins on the Arduino. A trimmer was wired to the ADC's analog input and turned around in order to see what the values would be. Turning it all the way to one side would produce a value very close to 0V, as we'd expect. Turning it to the other side produces a value of 4.064V, which is the maximum level that the ADC can get to. Thus our way of controlling the ADC was verified.

3.4 Microcontroller and USB-to-RS-232 Interface

In order to determine if the PIC16 and RS-232 interface were working properly, we had to create a program for the PIC16 that would receive a byte from the PC and then send the same byte over USB. To do so on the PC we connected the board to the PC via USB and opened up a serial terminal program on the PC. The board showed up as a COM port, which meant the CP2102 chip was working correctly. Sending multiple bytes of data to the board, we received the same byte back along with four other bytes, which were the two 16-bit samples from the ADCs, showing our RS-232 protocol was working.



```
COM4 9600 bps, 8N1, no handshake

00                       .
00 5f c3 a0 ac              ._Ã ¬
ff                      ÿ
ff 7f c1 38 a7            ÿ Á8§
aa                       ª
aa 7f c0 92 a9              ª À'©
55                       U
55 db b5 60 b2              UÛµ`²
de                       Þ
de d7 c7 68 ae              Þ×Çh®
ad
ad 0f d1 38 ab              .Ñ8«
be                       ¾
be 97 bf f8 a6             ¾ ¿ø¦
ef                      ï
ef 8a ca cc ad              ï ÊÌ
```

Figure 13: Screenshot of terminal following PIC16 test

3.5 Device Driver

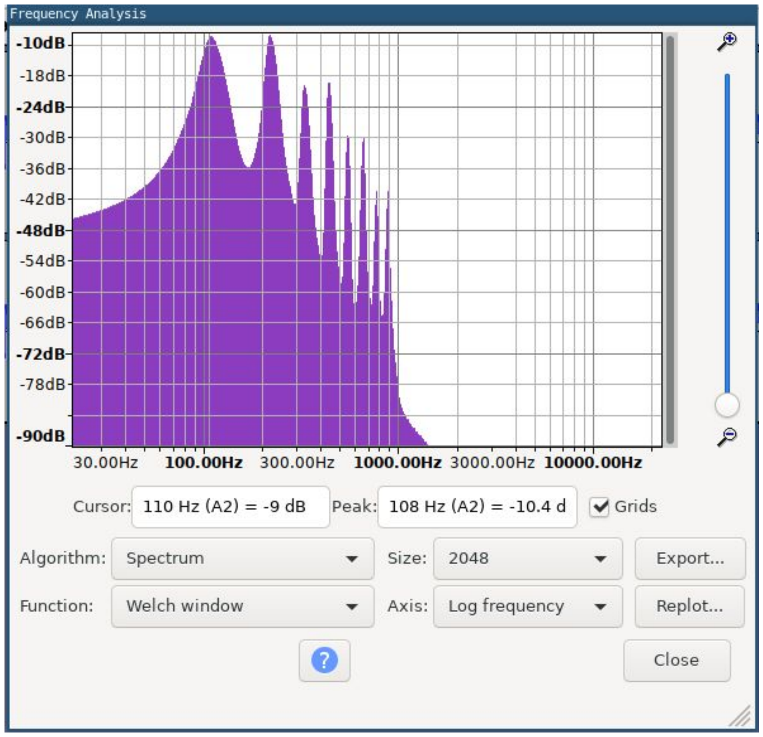
From personal experience, a rhythm game with high input lag will need to be compensated in some manner through either lagging the audio and video by the same amount or shifting the timing window so that being late in terms of timing isn't penalised. Of course, if the input lag is low then compensation will be averted, so we wanted to keep the input lag below 50 ms or else it would become very noticeable. To test the input lag, we used the C++ standard library to take a snapshot in time before requesting a sample from the microcontroller. The program then reads the four bytes from the microcontroller and then takes the time after all samples are received. Then the program calculates the difference between the two times and casts it to microseconds. This reveals that the whole input loop has a delay of around 9.6 ms, which makes for a very quick response for a video game, seeing as it's below a frame if the game is running at 60 fps.

3.6 Graphics Engine and Game Logic

Video games in general tend to run at 60 fps in order to make the animations appear smooth. To make sure the game runs at 60 fps, the time difference between one frame and the next was sent to the terminal while the game was running. This frame rate counter hovered at around 60 while playing the game on an eight-year-old ThinkPad, showing it doesn't cause too much strain on hardware. Despite how simple the game looks, the audio engine is quite computationally-intensive, needing to fill an audio buffer with synthesized audio constantly.

3.7 Audio Engine

Testing the audio engine individually without waiting for the game to be made involved searching online for a mouse input library. The idea was to use the mouse in order to change the pitch and volume of the synthesized sound. We found a simple library called Gainput which communicates with an X window for this purpose, setting the X axis to be the pitch and the Y axis to be the volume like in the game [7]. Using Audacity, we captured the PC audio output and ran the audio engine with this mouse modification. Scanning from one edge of the screen to the other, the two extremes of the synthesized theremin audio spectrum were recorded. The lowest fundamental frequency was 110 Hz, shown in Figure 14, corresponding to a note of A2, and the highest fundamental frequency was 1710 Hz, shown in Figure 15, corresponding to a note of A6. This shows an octave range of five octaves, which satisfies our requirements for pitch range.

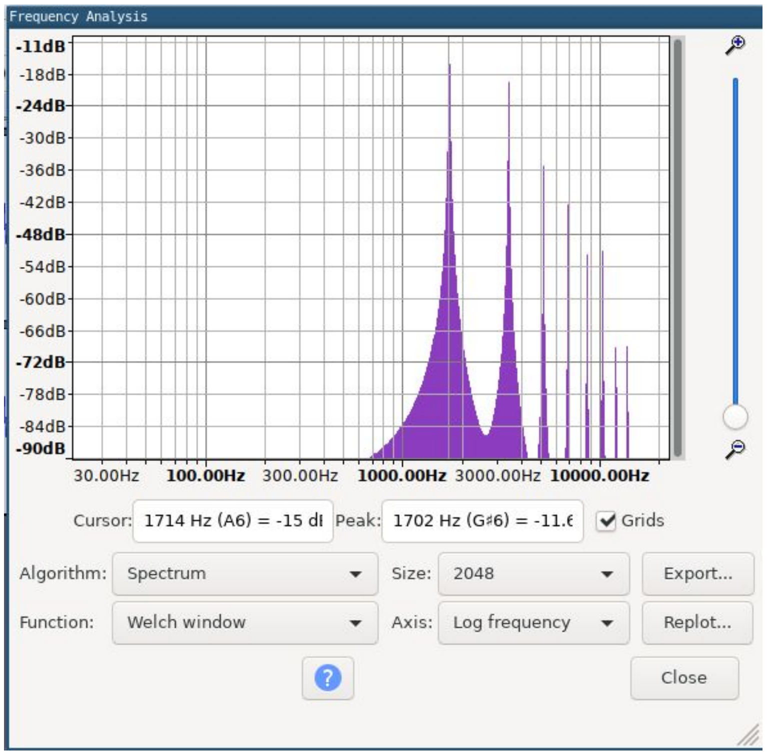


Cursor: 110 Hz (A2) = -9 dB Peak: 108 Hz (A2) = -10.4 d ✓ Grids

Algorithm: Spectrum Size: 2048

Function: Welch window Axis: Log frequency

Figure 14: Lowest Synthesized Note



Cursor: 1714 Hz (A6) = -15 dI Peak: 1702 Hz (G♯6) = -11.€ ✓ Grids

Algorithm: Spectrum Size: 2048

Function: Welch window Axis: Log frequency

Figure 15: Highest Synthesized Note

4. Costs

4.1 Parts

Table 1: Parts Costs

Part	**Manufacturer**	**Retail Cost ($)**	**Bulk Purchase Cost ($)**	**Actual Cost ($)**
PIC16C65A	Microchip	9.00	N/A	9.00
MAX1166 ADCs	Maxim Integrated	23.90	N/A	23.90
USB-B Connector	CUI Inc.	0.65	N/A	0.65
CP2102	Silicon Labs	1.33	N/A	1.33
2N2222 transistors	N/A	1.00	N/A	1.00
1N4148 diodes	Vishay	0.40	N/A	0.40
Reset button	E-Switch	0.23	N/A	0.23
WBC4-1TLB transformers	Coilcraft	14.68	N/A	14.68
74LS14 inverters	Texas Instruments	0.58	N/A	0.58
74LS109 flip-flops	Texas Instruments	2.68	N/A	2.68
74LS86 XOR gates	Texas Instruments	0.64	N/A	0.64
22AR1KLFTR trimmers	TT Electronics	7.26	N/A	7.26
RLC parts	N/A	~10.00	N/A	~10.00
Total		**72.35**		

4.2 Labor

The average salary for ECE graduates with a BS is around $40 an hour, and we had around eight weeks or so to work on the project. Averaging out how long we worked as ten hours a week with three people in the team, that amounts to about $24,000 in labor costs.

5. Conclusion

5.1 Accomplishments

Although the analog circuitry did not work too well in the end, we managed to think of different ways to tackle the problems we were encountering regarding the oscillators, detailed in the next subsection. And the two mixers (XOR and Gilbert cell) along with the integrator and detector met their unit tests for audio frequency signals. The digital portion of the theremin proved to be reliable, as we had no bugs regarding the PIC16 or the USB to RS-232 chip once we iterated upon the assembly program we created for the PIC16. The sound synthesis engine works decently well for the octave range it is set to and even works with a background music file. Although the video game is very barebones at the moment, it can perform the necessary functions of displaying a song map and taking the player's score.

5.2 Uncertainties

It is unknown just how well we'd be able to have the oscillators working if we kept iterating upon our design for them. The design that was used for the demonstration was very noisy, which made the game difficult to play and created unpleasant sounds out of the sound engine. If the problem still persists, it would perhaps be necessary to implement some kind of smoothing over the theremin samples to keep them from oscillating back and forth from noise and instability.

5.3 Ethical considerations

The portions of the ACM code of ethics that most apply to our project are the ones regarding personal safety and abiding by copyright law [8]. All of the software we have written has only used free software libraries of the kind where using the libraries simply requires a notice in the source code as to who authored that library. Since all the voltages we are working with are very low, touching the theremin antennae is completely harmless as well, and a USB port isn't going to be dangerous either.

5.4 Future work

More research and development would be necessary in order to create fully-functional reliable oscillators, as it was only in the last few weeks of the class that we realized how we would be able to improve an oscillator's design if its amplitude seemed lacking. Another way to improve upon the design would be to use subtractive synthesis in place of additive synthesis for the audio engine, as filtering out something like a saw wave would produce a richer tone due to the many harmonics that a saw wave contains. It would also help to create a menu and song select for the video game, as well as a score display and some kind of background to keep the game from looking too simple. And lastly, a proper wooden enclosure and brackets or sockets for the antennae would be necessary in order to give the theremin controller a more professional look.

References

[1] "What is beatmania IIDX," RemyWiki.com, Jan. 22 2019. [Online]. Available: https://remywiki.com/What_is_beatmania_IIDX#Gameplay. [Accessed Feb 1, 2019].

[2] "Karaoke Revolution," Wikipedia, May 17 2018. [Online]. Available: https://en.wikipedia.org/wiki/Karaoke_Revolution. [Accessed Feb 7, 2019].

[3] R. Moog, "Build the EM Theremin," Electronic Musician, Feb., pp. 86-100, 1996. [Online]. Available: https://www.cs.nmsu.edu/~rth/EMTheremin.pdf. [Accessed March 2, 2019].

[4] Maxim Integrated, "Low-Power, 16-Bit Analog-to-Digital Converter with Parallel Interface," MAX1166 datasheet, July 2002 [Revised Aug. 2008].

[5] H. Mossman, "Jungle Lullaby," ThereminVox.com, April 29, 2004. [Online]. Available: https://www.thereminvox.com/filemanager/list/8/index.html. [Accessed Feb. 4, 2019].

[6] P. R. Cook and G. P. Scavone, "The Synthesis ToolKit in C++ (STK)," 2017. [Online]. Available: https://ccrma.stanford.edu/software/stk/. [Accessed Feb. 4, 2019].

[7] J. Kuhlmann, "Gainput: C++ game input library," 2017. [Online]. Available: http://gainput.johanneskuhlmann.de/. [Accessed Feb. 11, 2019].

[8] Association for Computing Machinery, "ACM Code of Ethics and Professional Conduct," Association for Computing Machinery. [Online]. Available: https://www.acm.org/code-of-ethics. [Accessed: Feb. 1 2019].

Appendix A Requirement and Verification Table

Table 2: System Requirements and Verifications

Requirement	Verification	Verification status (Y or N)
1. Pitch and Volume Sensors a. XOR mixer should create a stable square wave with input frequencies up to 1 to 4 MHz at logical high (2-5 V) b. Antennae should be able to change the oscillator's frequency by 5-10 kHz from its base frequency to allow enough dynamic range for theremin control. The base frequency is the frequency during which the user's hand is far away from the antenna, which should be in the MHz range.	1. a. Connect function generator to each mixer input, one to simulate reference oscillator and the other to simulate the variable oscillator b. Probe output of mixer with oscilloscope c. Turn on signal generators to MHz range, go through range of values from matched frequencies to differences in tens of kHz d. Verify that oscilloscope shows pulse-width-modulated square wave at appropriate voltage level 2. a. Connect variable oscillator output to Schmitt trigger inputs and Schmitt trigger output to D flip-flops b. Probe output of D flip-flop with oscilloscope c. Turn on 5 V power supply to circuit d. Verify that oscilloscope shows suitable difference in frequency between a hand	Y

	near the antenna and hand far away	
2. Frequency Detection a. Detector must have high enough bandwidth to respond to sensor output, up to 30 kHz	3. a. Probe output of detector with respect to ground using voltmeter b. Connect function generator to input of detector with respect to ground c. Verify on voltmeter that inverse dependency on frequency is achieved throughout frequency range	Y
3. Analog-to-Digital Converter a. ADCs can send at least 480 samples per second (16-bit samples + start&stop bits per byte at 9600 baud = 480)	4. a. Connect 8-pin data bus and control signals to Arduino for testing, set up voltage divider with trimmer for ADC analog input b. Load program that manipulates ADC within specified timeframe to produce samples c. Verify that trimmer sweep corresponds to values from ADC	Y
4. Microcontroller a. Able to send samples from ADC to RS-232 interface at 9600 baud	5. a. Load PIC program that will receive byte from PC and send back using serial connection b. Open up serial terminal at 9600 baud, connect USB cable between PC and theremin controller	Y

	c. Send a byte to PIC and verify that the same byte is transmitted back to the PC	
5. USB-to-RS-232 Converter a. Able to interface with PC through USB b. Able to interface with microcontroller through RS-232 UART	1. a. Connect USB cable from converter to PC b. Verify that a new COM port device or /dev/ttyACM0 appears on PC 2. a. Program microcontroller with simple program that sends a string to the computer b. Connect USB cable from converter to PC c. Use RS-232 terminal program to read text input from virtual COM port d. Verify that string appears in terminal	Y
6. Graphics Engine and Game Logic a. The game should run at least at a constant 60 fps to ensure a smooth gaming experience	1. a. There will be a frames per second counter on the screen. We just have to make sure it doesn't stay below 60.	Y
7. Audio Engine a. Should be able to play a song synchronized with the video game so that there is no noticeable lag between what is on the screen and what is being heard b. Should be able to generate theremin-like sound for a range of at least 4 octaves	1. a. Have a decently-sized sample of people play the game (probably 4 or more) b. If there is a consistent shift in timing towards late or early timing, the game is probably not synced correctly 2. a. A very subjective requirement, it can nevertheless be tested by	Y

	going through the full 4 octave range and comparing it to recordings of real theremins	
8. Device Driver a. Should provide a steady stream of 32-bit samples from the theremin at a high enough bandwidth to minimize delay below 50 ms in order to prevent input lag.	1. a. Set up microcontroller to send varying output through USB to PC b. Print out stream of data coming from device driver c. Check to see if time difference between microcontroller generating output and device driver output exceeds 50 ms	Y