MIDI Zeusaphone

Final Project Report

Team Number: SDMAY19-11
Client: Dr. Joseph Zambreno
Advisor: Dr. Joseph Zambreno

Team:
Jacob Feddersen
William Brandt
Luke Heilman
Gregory Harmon
Leo Freier
Gunnar Andrews

Team Email: sdmay19-11@iastate.edu
Team Website: https://sdmay19-11.sd.ece.iastate.edu

Table of Contents

Executive Summary

When prospective students are given a tour through Iowa State, they are shown the accomplishments and senior design projects of past undergrad students. The Electrical and Computer Engineering Department currently has two arcade cabinets that were constructed by previous electrical and computer engineers. They are rarely seen in use. In order to continue attracting students to ECpE, the department needs a new showpiece to demonstrate what prospective students could be capable of if they choose to attend Iowa State.

Our solution to this problem is to construct a Tesla Coil that plays music, also called a Zeusaphone. The Zeusaphone is able to load and play preset songs from MIDI files. It can also be played with a MIDI musical keyboard so there is a more interactive aspect for the viewer. The project includes specialized circuits and a microcontroller. It will appeal to prospective students with an interest in embedded/low level software, circuit design, and power electronics. The wide range of topics should help the project be successful by potentially appealing to many different student interests. Because it will potentially be shown on tours, an operating manual is provided to ensure the operator is using the Zeusaphone properly. An additional safety manual will also be provided so that operators are aware of the safety measures to be taken when operating the device.

The team is comprised of four Computer Engineering students and two Electrical Engineering students. The project appeals to the team as many have a musical background and all students find the tesla coil to be an intriguing subject on its own.

1 Requirements Specification

1.1 Functional Requirements

- Coil powered from 120V, 60Hz, standard wall outlet
- Create visible and audible sparks while the coil is running
- Coil can play two notes at the same time
- User-facing API, where users can play music from a MIDI keyboard or midi files

1.2 High-Level Requirements & Use-Cases

1.2.1 High-Level Requirements

- Safety cannot be compromised for any functionality
- Be an entertaining showpiece

1.2.2 Use-Cases

- Mostly as a demonstration for prospective students
- Run by ECpE employees

1.3 Non-Functional Requirements

- Cost less than $1,000
- Size constraints: 1x1 foot base, less than 2 feet tall
- Reliable for demonstrations
- Easy to move, setup, and use

2 System Design & Development

2.1 Design Plan

We propose a Zeusaphone that is controlled by a microcontroller. The microcontroller processes MIDI events from a variety of inputs and controls the tesla coil accordingly. A very general overview of the layout can be seen below in figure (Fig.1).

We chose to use a Raspberry Pi for the microcontroller. The Raspberry Pi is a small, credit card sized computer that is capable of running a full Linux operating system. We chose to use the Raspberry Pi for a number of reasons:

- The Raspberry Pi has a number of general purpose I/O pins, including two with hardware-timed pulse-width modulation, which we use for outputting the audio waveforms.
- The Raspberry Pi is capable of acting as a wireless access point, and it can host its own web server. We use this to host an interface for controlling the tesla coil system.
- We had several Raspberry Pi's immediately available for testing, and several of our team members have experience setting them up and using them.

Music can be played on the tesla coil from two different sources. First, MIDI files can be stored on the Raspberry Pi and played back. These MIDI files are loaded, managed, and played using the web interface. Second, a MIDI keyboard can be plugged into the Raspberry Pi and used to create live input. The web interface is used to select between keyboard mode or just playing a loaded song. When in keyboard mode, the pitch bending knob will (by intentional design) affect the frequency. This ranges from plus or minus two semitones from the original notes. It is essentially an added feature that a normal keyboard would have.

The Raspberry Pi is connected to the tesla coil with a fiber optic cable, to avoid interference from the operation of the tesla coil. The output from this fiber line is used to modulate the tesla coil itself. When the line is active, the tesla coil is enabled and sparks are created. When the line is not active, the tesla coil is off. By modulating this output, music can be played through the sparks on the tesla coil.

In order for a user to control the Raspberry Pi itself, the Pi hosts a simple web interface accessible through http using any web browser. This web interface will have the following features:

- Upload a new MIDI file to the Raspberry Pi, to enable it to be played on the tesla coil
- Play a MIDI file currently stored on the Raspberry Pi
- Enable/Disable the MIDI keyboard input functionality

To keep this control limited to the correct people, the Pi will transmit its own WiFi Access Point (WAP) protected by WPA2. The web page will only be available on this WAP, which will not be connected to the internet itself, thus providing a layer of security.

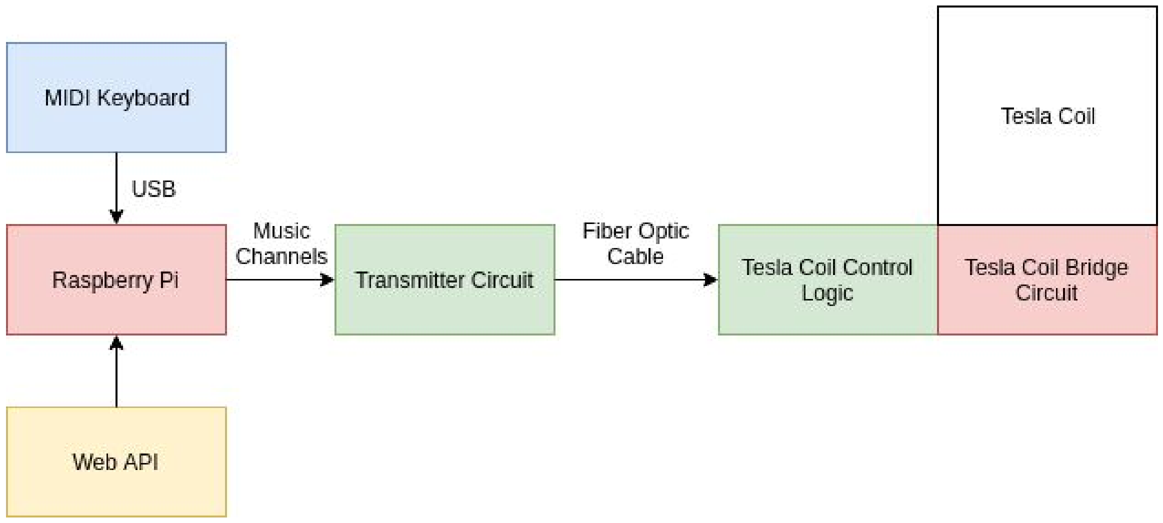


Figure 1: Overview of the Project Layout

The tesla coil itself is classified as a solid state tesla coil (SSTC), which means it uses semiconductors (not a spark-gap) to drive the primary coil. Our tesla coil uses power MOSFETs to switch the voltage across the primary coil, which then causes the secondary coil to resonate. This resonation steps the voltage up drastically in the secondary coil, and that causes the air to breakdown around the top of the coil, releases sparks and sound. For the high frequency switching to occur, the MOSFET's are driven by the control logic circuit. This circuit takes in the output from the transmitter circuit and syncs it with the output of the coil with an antenna. The synced signal is then used to drive the MOSFETs. The secondary coil is wound with 30 gauge magnet wire, and attaches to a conductive, toroidal topload. This topload adds capacitance to the secondary, allowing the coil to build up more energy before it releases it as sparks. The coil and its associated electronics has a base of 8 inches by 8 inches, and is shy of 2 feet tall.

2.2 System Constraints and Trade-Offs

2.2.1 Assumptions and Limitations

Assumptions:

On Usage

- The operator will be able to play a MIDI keyboard to produce sounds
- The operator can play pre-loaded MIDI songs to play through the web client.
- The operator can load MIDI songs through the web client to be played later.

On Safety

- The primary use of the Zeusaphone will be as a showcase item.
- The operator will be fully aware of the safety considerations and proper use of the Zeusaphone.
- During operation, all safety standards will be followed by the operator and the audience.

- When not being shown, the operator assumes responsibility as laid out by the provided safety standards for a safe startup and/or shutdown.

On Reliability

- The system can be safely stored in any room safe enough to store high voltage circuits.
- The full project will be able to be reliably moved to and from storage with minimal assembly and disassembly.
- A combination of input will not result in a dangerous situation.
- The system can be safely shut down and de-energized immediately at any time.

Limitations:

- The end product is no larger than 2 ft tall with a 1 x 1 square foot area
- It must be able to be run off of a wall outlet. (120V 60Hz)
- Can only play two different tones at once
- Operators must be associated with the EcpE Department.
- The Tesla coil will only be able to be activated using the project interfaces.

2.2.2 Trade-Offs

Design choices for the Zeusaphone were all made based on research and reasonable estimation. Three main trade-offs that ended up in the final design were using Power MOSFETs instead of IGBTs, a Raspberry Pi, and using a full H-bridge instead of a half bridge circuit to drive the primary coil.

In a solid state tesla coil, different transistors have different qualities for the circuit. In our Zeusaphone, we use power MOSFETs instead of another popular transistor for an SSTC, the IGBT. The functional difference is that power MOSFETs have higher conduction losses but lower switching losses, and IGBTs have lower conduction losses but higher switching losses. Since we are building an SSTC, which has lower peak currents than a DRSSTC, we chose to use power MOSFETs. This allows us to have simpler control circuitry, since we do not need to worry about hard switching the MOSFETs.

Using a Raspberry Pi was a point of contention in the design. A major reason we used the Pi was because we were already familiar with it. It can also easily host the web interface, so it made that implementation much smoother. We considered using an Arduino as a more lightweight solution, meaning we could get much more accurate arbitrary waveforms. The best solution would have been using an FPGA and implementing a hardware solution. However, we were not familiar with FPGAs at the beginning of the project and might not have had the time to completely implement a hardware solution. A Raspberry Pi would still have to be used for the web interface, but an Arduino or FPGA solution would have given us more flexibility for the actual waveform generation used to drive the tesla coil.

Our original plan was to use a half bridge of MOSFETs as the primary coil driver, because the circuit could be made more compact with fewer parts. However, we decided later to expand to a full H-bridge because it enables us to apply twice the voltage across the primary coil. This made a significant increase in the spark length we could achieve, even though it requires twice the components and a larger PCB footprint.

2.3 Architectural Diagram

The figure below shows the architecture of the software, which follows a linear data flow. The user interacts with system through the website, also using a MIDI keyboard if they want. The website controls two applications, one for each input method. These apps both connect to the driver software, which converts the software signals into analog signals sent to the hardware.

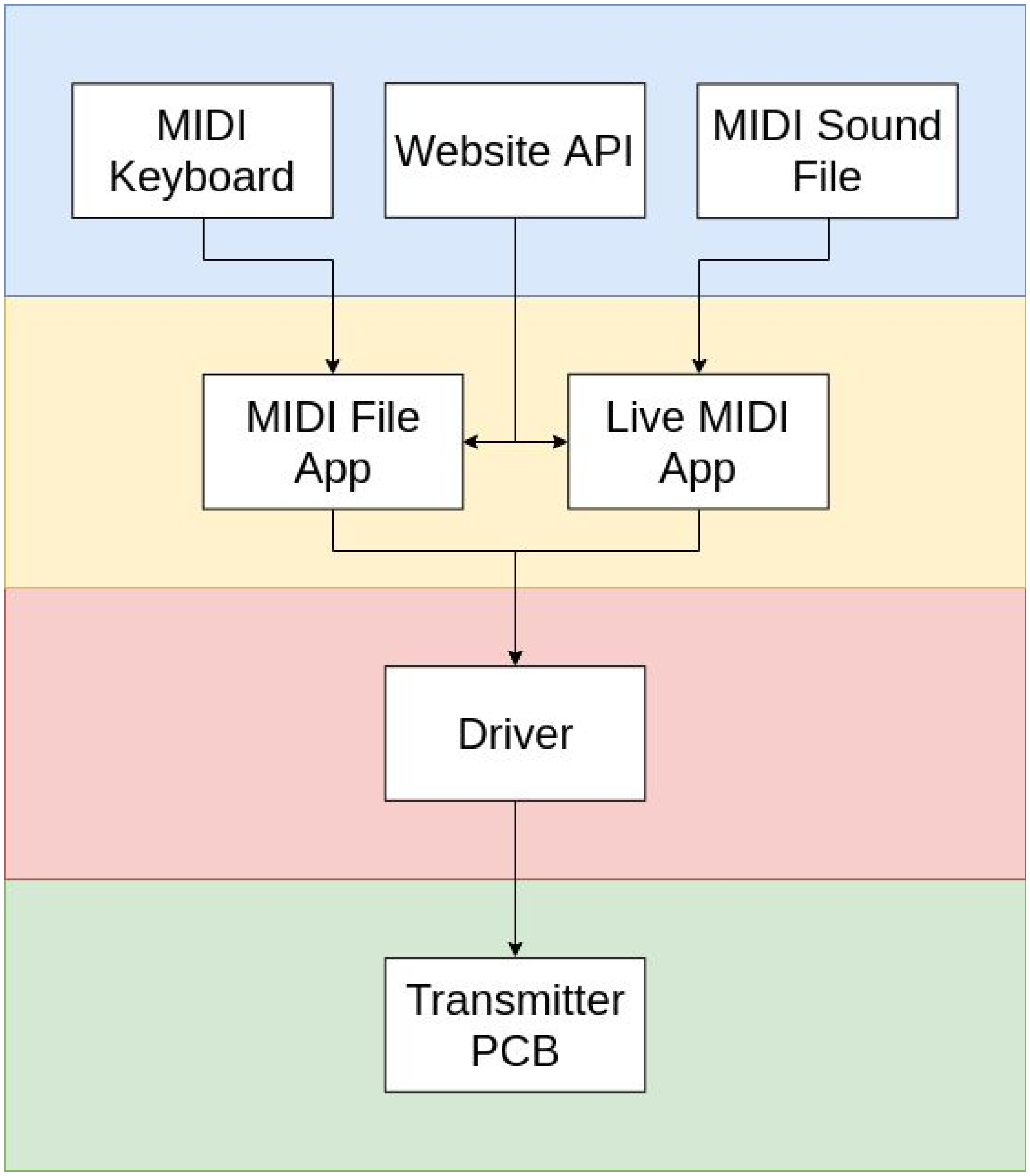


Figure 2: Software Architecture

The following figure shows the hardware architecture of the project. There are three main physical components: the MIDI keyboard, the transmitter, and the coil itself.

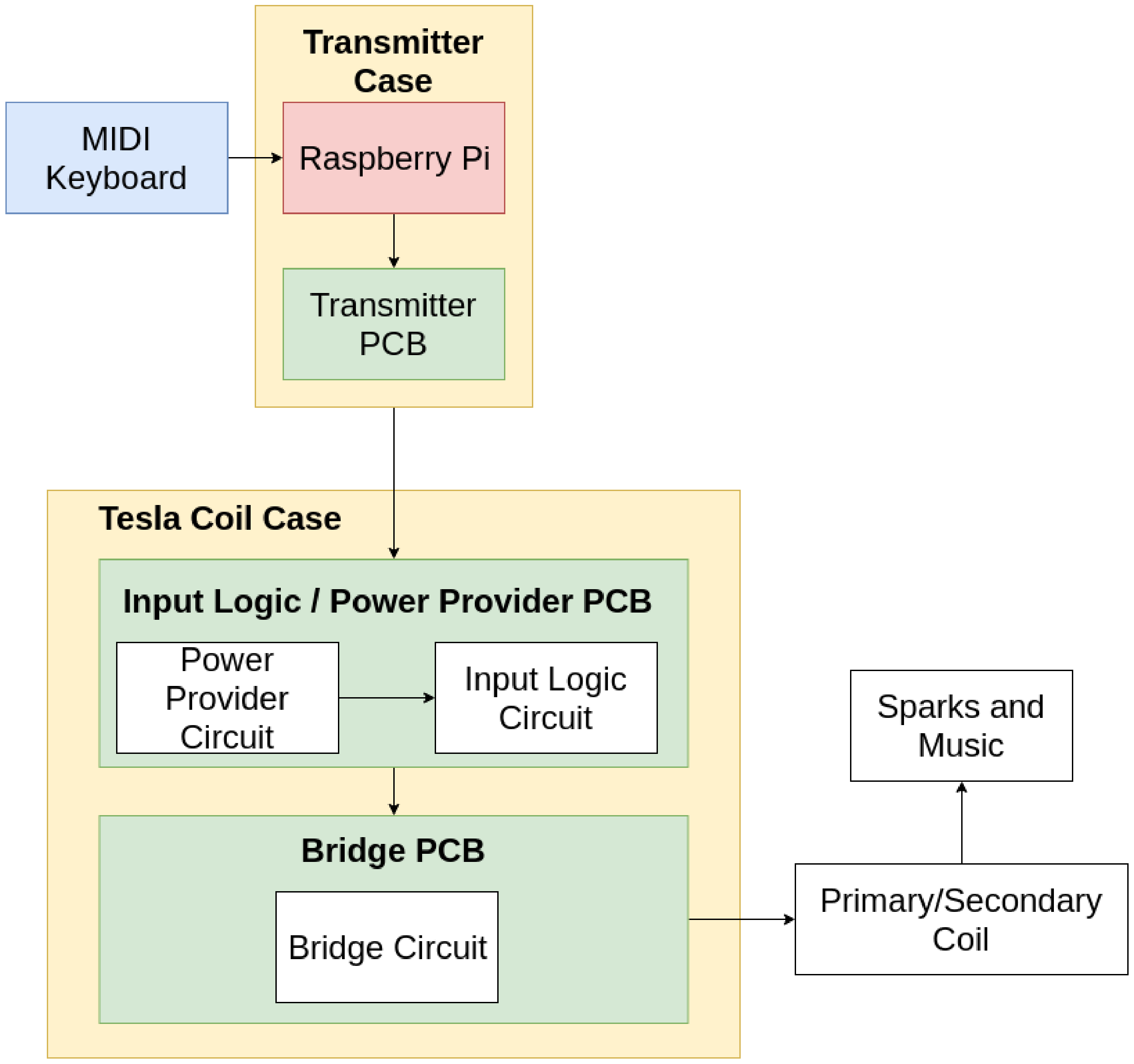


Figure 3: Hardware Architecture

2.4 Description of Modules, Constraints, and Interfaces

2.4.1 MIDI Keyboard

The keyboard is a AKM320 32 key MIDI controller. It was selected for its price, simplicity, and because it connected over USB. The only thing the keyboard needs to accomplish is to take user input via the keyboard keys and output them as a MIDI signal. Thus a simple, lower-cost keyboard was selected. Since the coil nearby produces a magnetic field, a keyboard that communicates over wire was selected over one that communicates wirelessly.

As a bonus feature, the keyboard does have a pitch-bend wheel. This functionality was added to our software, so using the pitch-bend wheel while playing the keyboard affects the musical output accordingly.

2.4.2 Transmitter

The transmitter consists of the Raspberry Pi microcontroller and the transmitter PCB. The Pi was chosen because of its abilities to run a Linux OS and host its own wifi access point (WAP). Several team members also had prior experience with setting up and using a Pi. The Pi hosts the project website API on its own wifi network. The API allows a user to control the system by enabling either the MIDI keyboard or by playing a MIDI file song on the coil. Whichever is selected, the Pi outputs the appropriate audio frequency waveform through its two hardware pulse-width modulation (PWM) pins, one pin for each note channel. These two signals are then merged in the transmitter PCB, which also prevents two pulses from overlapping with each other and causing a pop in the music. The transmitter PCB also contains status LEDs and a power off button, which shuts down the Pi when pressed. The transmitter outputs the merged note signal via a fiber optic transmitter.

2.4.3 The Tesla Coil

The tesla coil circuitry consists of two PCBs, the primary and secondary coils, and the topload.

The first PCB, the Power Provider/Input Logic PCB, accomplishes two tasks. It steps down voltage from a 120V, 60Hz wall outlet, and then rectifies it, providing a 12V power rail and a 5V power rail. These are used by the input logic, which receives signals from the transmitter and syncs them with the secondary coil with antenna feedback. This synced signal is used to control an inverted and uninverted gate driver chips. The gate driver signals are sent to the bridge PCB.

The bridge PCB is powered directly by rectified 120VAC from the wall outlet. The circuit takes the gate driver output and sends it through a pair of gate drive pulse transformers to isolate the circuits. The signal is used to drive the four H-bridge MOSFETs, which in turn drive the primary coil of the tesla coil. The primary coil is wrapped around the secondary coil, with a turn ratio of approximately 1:250. The primary coil induces current and voltage in the secondary coil, which steps up the voltage. A capacitive topload briefly builds up this energy before releasing it as sparks and sound.

3 Implementation

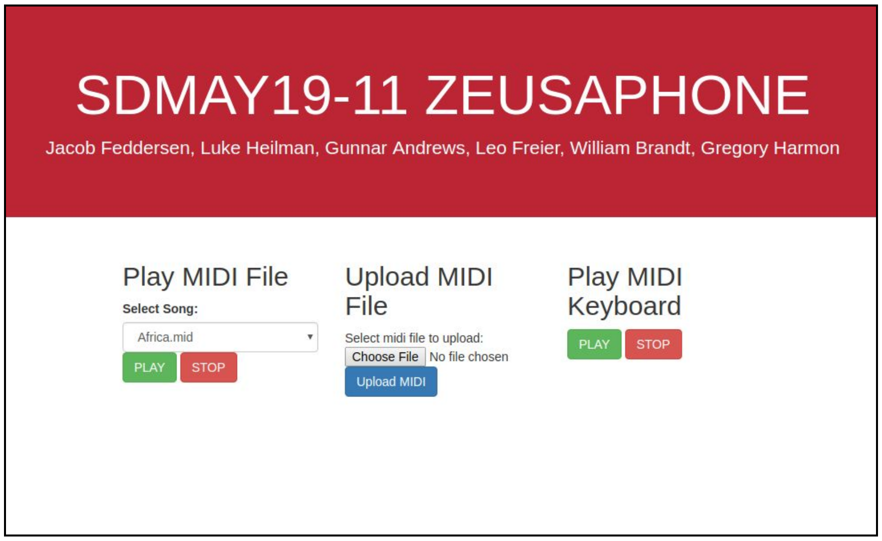
3.1 Implementation Technologies

Software and hardware implementation were all done with modularity in mind, so we were able to complete parts of the project without needing everything working. Software was separated enough that our software implementation was completed in the first semester, so the only software remaining for second semester was the Web API. The final step of hardware is PCB fabrication, so before we even started work on PCBs we ensured that the system worked with preliminary testing. Implementation of hardware encountered some obstacles, but we were able to move past them and complete full hardware implementation.

3.1.1 Software Implementation

3.1.1.1 Web API

Before the Web API could work there had to be some configuration changes made to the PI. The first thing that happened was by following instructions given on the raspberry pi website, we set up the raspberry pi to host a wifi access point and its own apache web server. We set the static IP address to be 192.168.4.1. So to connect to the Web API a user just has to log onto the access point (with password: sdmay19zeus) and the go to that IP address in their browser. They will then hit our homepage which is an HTML frontend connected to a php backend. From this page users can access all the functions of our project. They can play/stop a song that is saved on the PI, they can upload their own MIDI file to the PI so they can play it, or they can activate the keyboard that is plugged in to the PI and then start playing the keyboard. All these functions are handled through php scripts that run in the background. The CSS design of the frontend was made using bootstrap 4 to make it much more appealing to look at when it is on a smartphone!


SDMAY19-11 ZEUSAPHONE

Jacob Feddersen, Luke Heilman, Gunnar Andrews, Leo Freier, William Brandt, Gregory Harmon

Play MIDI File

Select Song:

Africa.mid

Upload MIDI File

Select midi file to upload:

Play MIDI Keyboard

Figure 4: The Web Interface

3.1.1.2 Application Software Layer

The app layer is where music is input into the system. There are two different applications, one for reading MIDI files from storage and one for reading live input from a MIDI keyboard. Both applications write the notes in the same way to a Unix socket. The apps are responsible for reading and parsing the input data, converting it into a two-channel stream of notes, and sending it to the driver layer through the socket where notes will be read and generated.

3.1.1.2.1 MIDI File Application

The MIDI file reader opens and reads MIDI files from the filesystem using the midifile C++ library written by Craig Sapp. This library reads the MIDI file and parses it into a C++ object. All of the events in the MIDI file are stored in an array, in order. The MIDI file app reconstructs the timing of events using a wait loop, and reads the notes specified in the file. If more than two notes are played at once, some of the notes may be discarded; the MIDI file app will only play two channels at a time.

3.1.1.2.2 Live MIDI Application

The MIDI keyboard receiver program listens for MIDI messages from devices attached to the Raspberry Pi. These messages are initially processed by the Advanced Linux Sound Architecture library (ALSA) in the kernel. The program then makes use of the RtMidi library by Gary Scavone to parse these messages and setup a callback function to handle them. The MIDI keyboard app ignores all MIDI events except note on, note off, and pitch bend. It only outputs two channels at a time, but it overwrites the oldest note that is still being held if a third note is played.

3.1.1.3 Driver Software Layer

The driver is the program on the Raspberry Pi responsible for interfacing with the tesla coil. It runs in the background, and acts as a server for the MIDI file app and MIDI keyboard app to connect to. It listens to the note messages input to a socket, and it generates output waveforms on the Raspberry Pi GPIO pins accordingly. Frequency is varied based on the notes being played, which is what determines the pitch of a notes, but the pulse width remains constant for all notes.

After experimenting with timing loops, threading, and interrupts, we finally decided to use the Raspberry Pi's hardware pulse width modulation pins to output the waveform. None of the other options provided the stability and timing accuracy that we needed. The hardware PWM pins have their own timing chip and counter that are seperate from the system clock of the Raspberry Pi. These counters will continue to output a wave at exactly the frequency set no matter what the workload of the processor is, and no kernel process can preempt or delay them. Issues came from using other pins because other tasks on the Pi can stall the output wave, leading to notes being played incorrectly with lots of static sound. The final waveform on the PWM pin will not be affected by these problems. Each channel is then sent as two outputs into the transmitter circuit, so two channels of music can be played. The transmitter circuit filters the waveform before it is finally sent to the coil.

3.1.2 Hardware Implementation

3.1.2.1 Transmitter PCB

The purpose of the transmitter circuit is to convert the signals output from the Raspberry Pi to a waveform over a fiber optic transmitter. The circuit also contains some logic to add a minimum inter-pulse period, so two pulses played overlapping each other will not create an extended pulse. This extended pulse causes an pop in the sound and also has the potential to damage the coil, since it is being held on for a longer period of time. This logic is done in hardware rather than software because of the time sensitivity; even small delays in this logic will drastically affect the output of the coil.

The three LEDs output from the Pi (LEDR, LEDY, and LEDG) are signal LEDs that show the state of the transmitter circuit. The red LED shows that the Pi is powered on. The green LED signals that the Tesla Coil driver software is running. The yellow LED signals that a program (either a keyboard or a Midi file) is connected to the driver software and running on the Tesla Coil.

The data sheets for the LEDs state that 30 mA is the maximum forward DC current they can handle. We did not want to push this, so we decided to select resistor values that placed the current about 20 mA. The LEDs are powered by 3.3V rails. The green and yellow LEDs draw 2.4 V, while the red LED draws 2 V. The remaining voltage needs to go across the resistor. The following equations show the how the resistor values were selected:

$$\frac{3.3-2.4V}{20mA} = 45\Omega \text{ (green and yellow LED)}$$

$$\frac{3.3-2V}{20mA} = 65\Omega \text{ (red LED)}$$

Since we already had 47 and 68 Ohm resistors, we just used those. This makes the LEDs produce the maximum brightness we are willing to let them create. Combined they draw about 60 mA, which is acceptable for the Raspberry Pi power supply.

The Raspberry Pi's two PWM pins each carry info for a note channel. These two signals are combined with an OR gate in IC1. Since these output pins function at 3.3V, but the 555 timers require a 5V signal, a pull-up transistor (IC5) is used to bring up the voltage to 5V. This also inverts the signal which is then fed into a second OR gate (IC4), which operates with the 555 timers. Both 555 timers operate in monostable mode. The first 555 timer (IC6) acts as a timer - when it receives a pulse (active-low) it goes high for 110 µs. This output is fed back into the OR gate. When the 2nd 555 timer (IC7), which is also active-low, receives a pulse, it goes active for 90.2 µs. This signal is finally fed into the fiber optic transmitter.

The amount of time a 555 timer operating in monostable mode stays active after being enabled is given by the following equation from Kane Philip [6]:

$$t = 1.1 * R * C$$

Rearranging to solve for R gives

$$R = \frac{t}{1.1 * C}$$

With a capacitance of 0.01 μF, and times of 110 μs and 90 μs, the resistor values are calculated to be 10kΩ and 8182Ω, respectively. Thus we selected 10 kΩ and 8.2 kΩ resistor values were used as the closest standard resistor values to what we calculated.

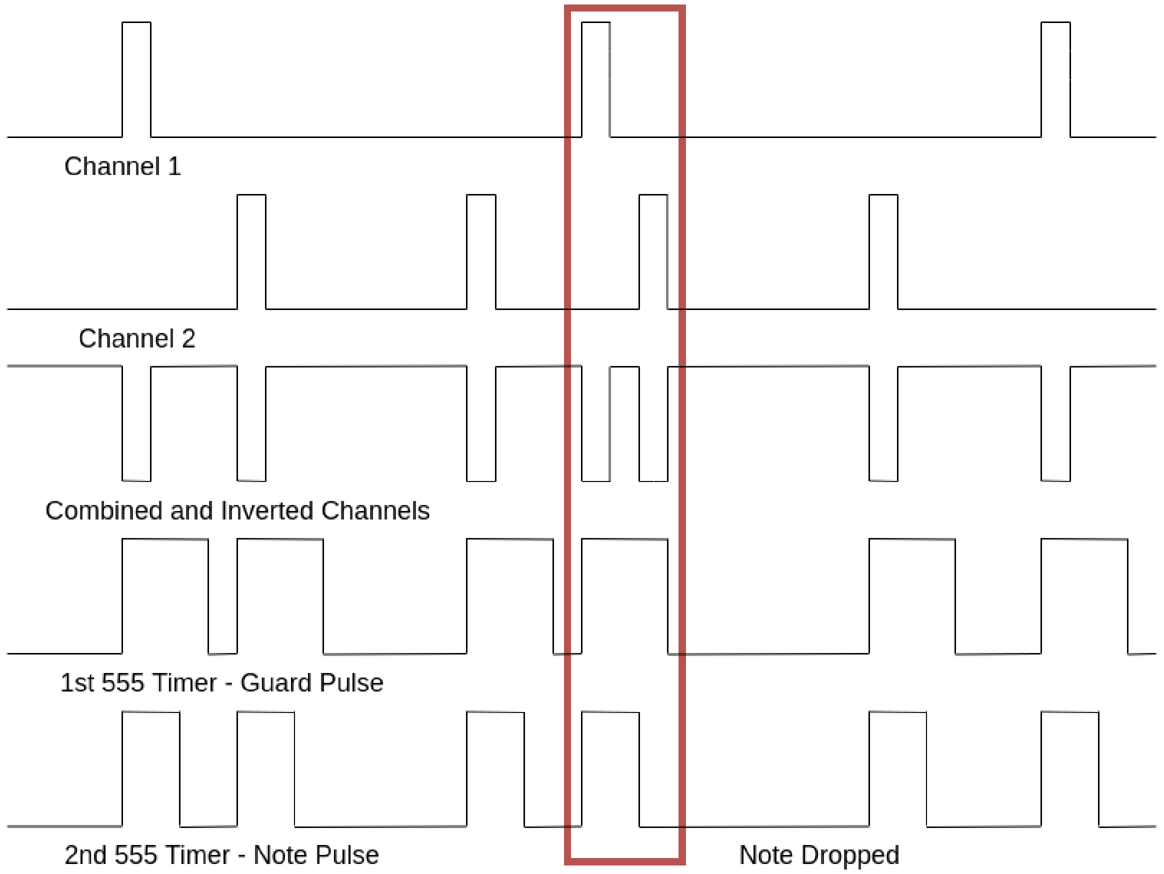

Channel 1
Channel 2
Combined and Inverted Channels
1st 555 Timer - Guard Pulse
2nd 555 Timer - Note Pulse
Note Dropped

Figure 5: Graph of Transmitter Waveforms

The transmitter circuit was also designed to operate within the power limits of the Raspberry Pi power supply, which supplies 2.5A max. The Raspberry Pi will use at most 1A. The LEDs require 60mA. The MIDI keyboard uses at most 500mA since it operates within USB 2.0 limitations. The fiber optic transmitter uses 30mA. This totals up to about 1.6A, which is well within the 2.5A limit.

The final output of the transmitter circuit is a single waveform being sent into a fiber optic transmitter into the fiber optic cable. All of the transmitter circuitry is assembled onto a single printed circuit board. The Tesla coil receives the waveform from the transmitter through the fiber optic receiver. This waveform is then used to trigger the coil on and off. A fiber optic connection is used as it is not affected by the electromagnetic field generated by the tesla coil.

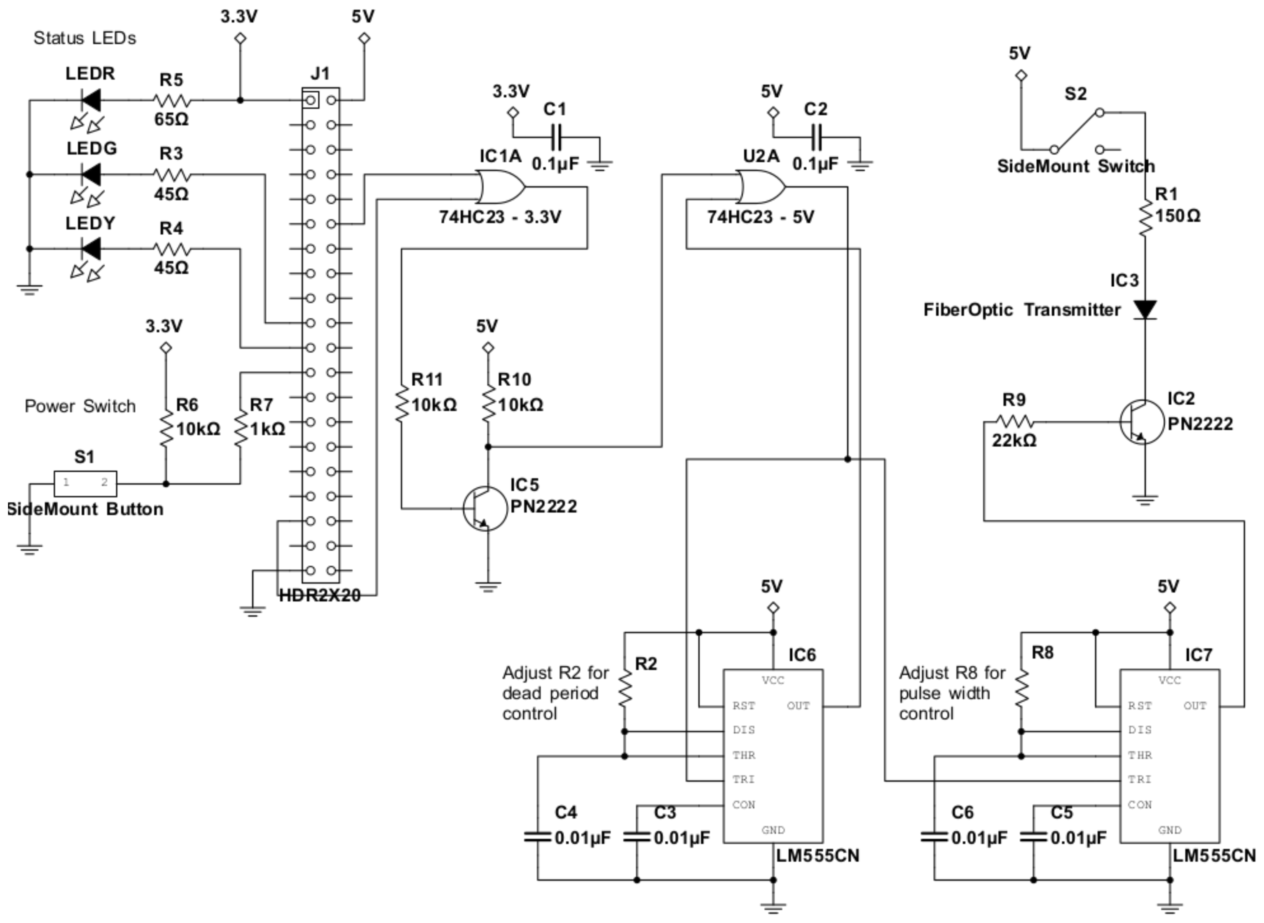


Figure 6: Transmitter Schematic

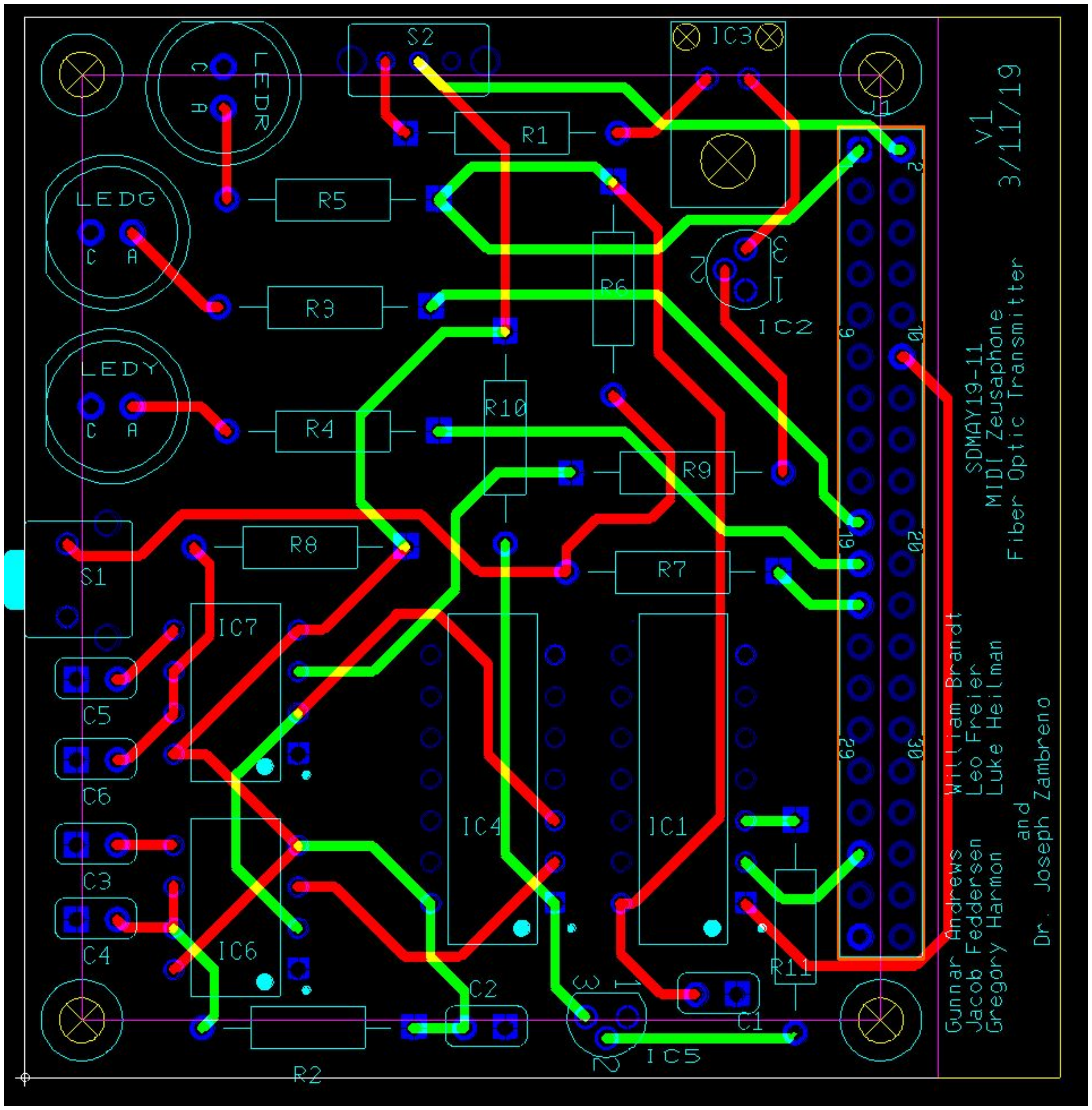


Figure 7: Transmitter Printed Circuit Board Layout

3.1.2.2 Power Provider and Input Logic PCB

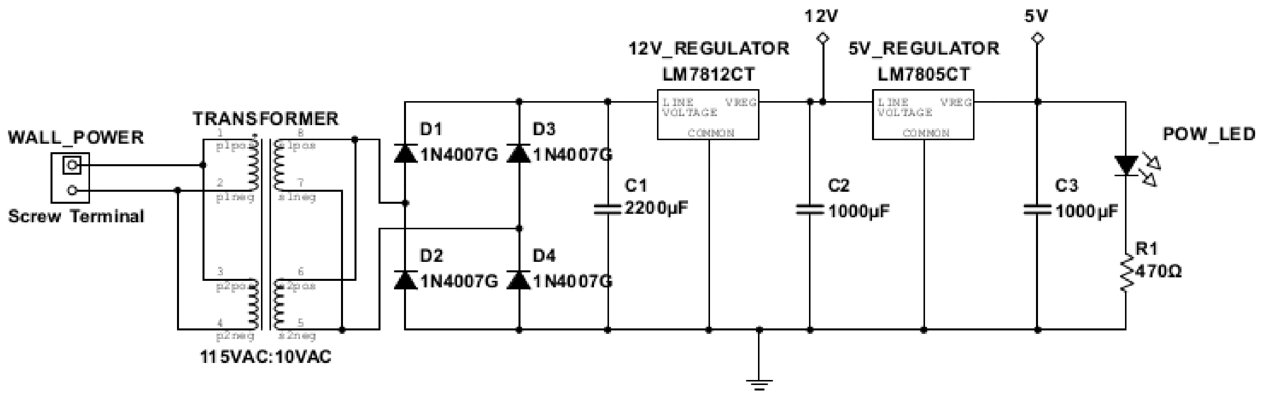


Figure 8: Tesla Coil Power Provider Schematic

The power provider circuit is simply a circuit that provides 12V and 5V rails for the rest of the tesla coil. It uses a transformer to convert 120V from the wall power to a lower voltage. We initially designed it to go from 120V to 20V. However, we discovered that transformers don't output a constant voltage - it depends on the current draw. Since the circuit does not draw much current, the output voltage of the transformer was actually closer to 40V, which is too high for the 12V regulator. The transformer could be re-wired to output 10V instead, which when used with the circuit actually output around 16.2V, which works well with the 12V regulator.

After stepping down the voltage, it is rectified through a full bridge rectifier before powering the 12V regulator, which in turn powers the 5V regulator. A red LED was attached to the 5V rail to indicate if the power provider circuit is running. Note that it does take several seconds for the resistor attached to the LED to bleed off remaining energy after the circuit is unplugged.

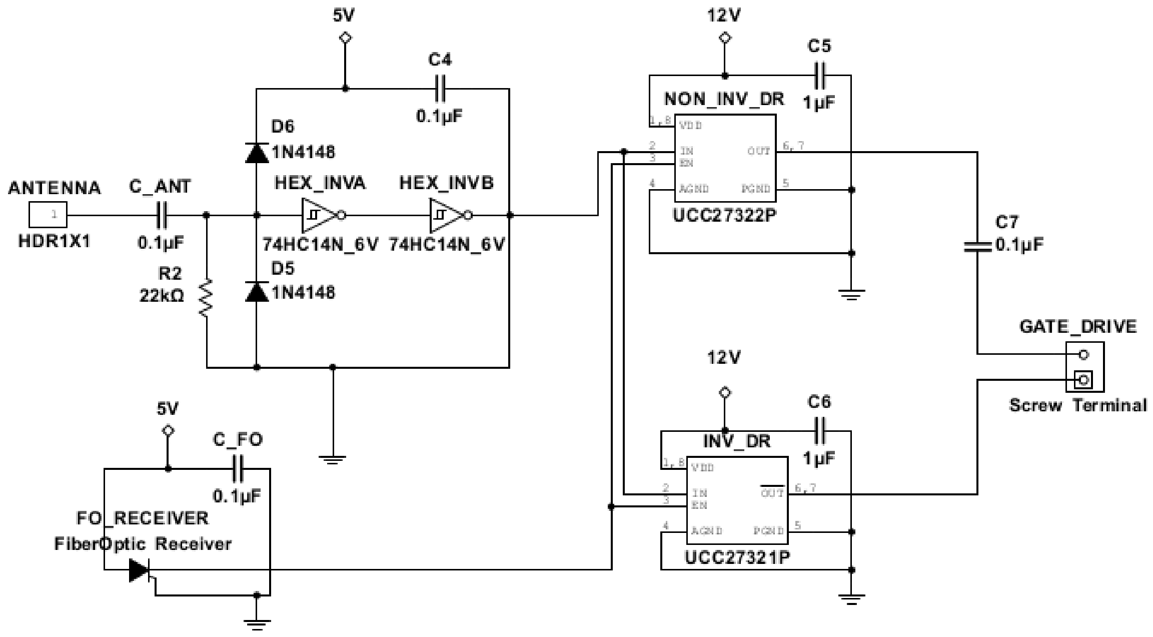


Figure 9: Tesla Coil Input Logic Schematic

The input logic circuit receives the signals generated by the transmitter via a fiber optic cable and synchronizes them with the resonance of the coil. The fiber optic receiver signal is used to drive two MOSFET drivers (INV_DR and NON_INV_DR), which are designed to connect to MOSFETS and switch them on and off according to the signals they receive. One driver is an inverting driver, while the other is a non-inverting driver - this allows one driver to switch off while the other switches on. The output of these drivers are then fed to the gate drive transformer on the bridge circuit.

The enable pins of the MOSFET drivers are connected to the fiber optic signal received from the transmitter. When enable is off, both the inverting and non-inverting drivers output a low signal. When the music pulse arrives, the enable pin is driven high, and the inverting driver also goes high. This first pulse starts the coil oscillation, which is picked up by the antenna and used to driver further oscillation.

The drivers are synchronized with the coil itself by attaching the input pins of the drivers to the output of an antenna, which picks up the electric field generated by the coil. To convert the antenna's output to a digital signal, it is run through a Not Gate (the HEX_INV) twice. To prevent the gate from getting fried by large signals from the antenna, the antenna's output is clamped between 0V and 5V by two diodes, one tied to ground, one tied to 5V.

The input of the first hex inverter is tied to ground through a pull-down resistor. We found that we needed this resistor due to the high electrical noise in the environment of the operating tesla coil. Without the pull-down resistor, the hex inverter generated an unstable signal when the tesla coil was run at high power.

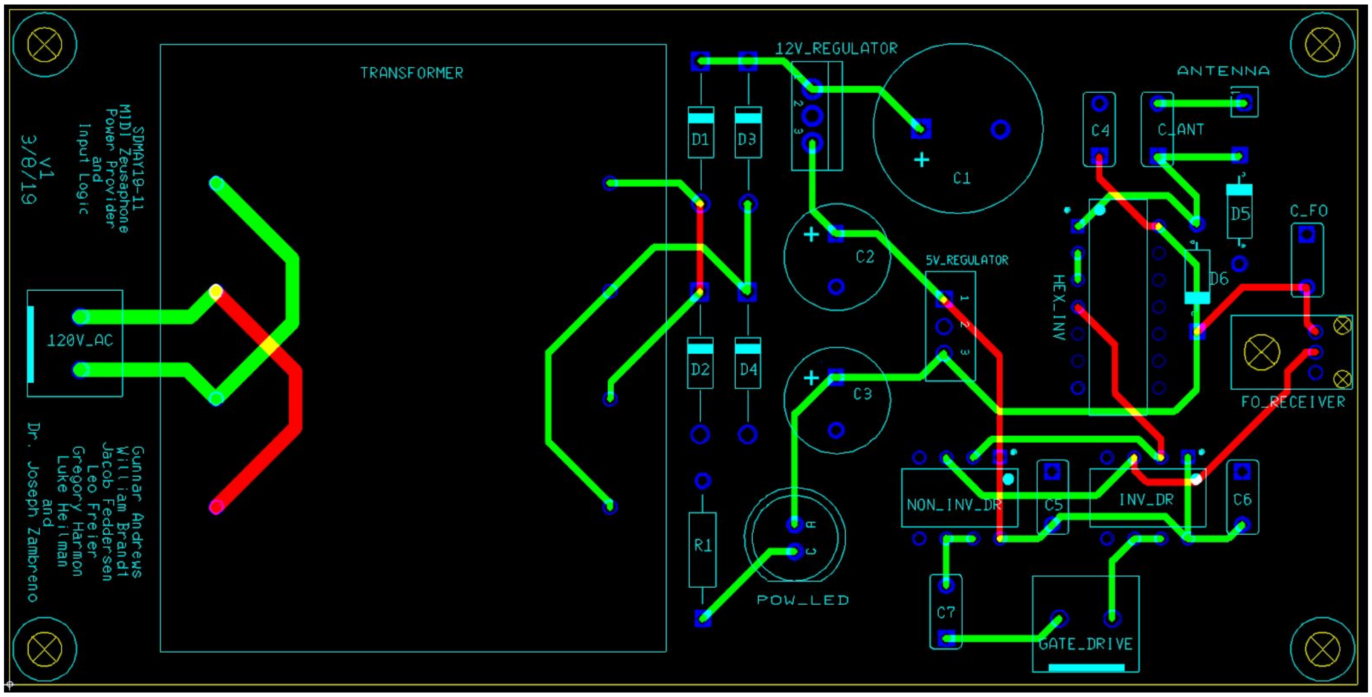

TRANSFORMER
12V_REGULATOR
ANTENNA
SDMAY19-11 MIDI Zeusaphone Power Provider and Input Logic
V1 3/8/19
120V_AC
Gunnar Andrews William Brandt Jacob Feddersen Leo Freier Gregory Harmon Luke Heilman and Dr. Joseph Zambreno
D1
D2
D3
D4
C1
C2
C3
C4
C_ANT
C_FO
D5
D6
HEX_INV
5V_REGULATOR
FO_RECEIVER
NON_INV_DR
INV_DR
C5
C6
C7
R1
POW_LED
GATE_DRIVE

Figure 10: PP/IL Printed Circuit Board Layout

3.1.2.3 Bridge PCB

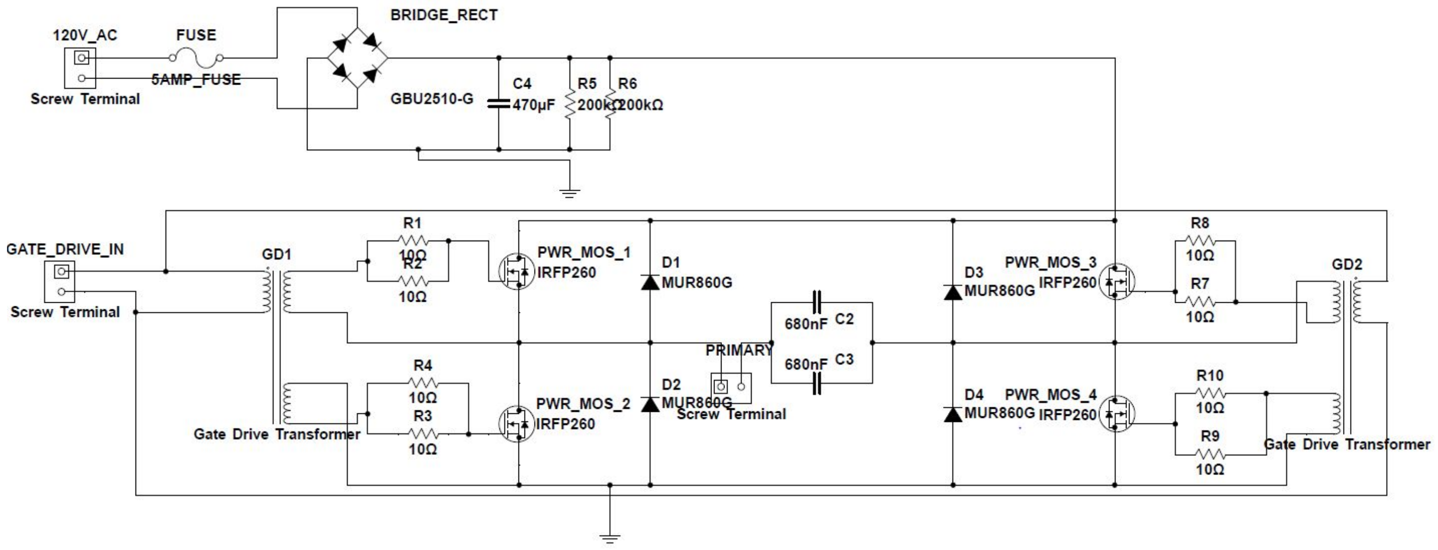


Figure 11: Tesla Coil Bridge Circuit Schematic

The bridge circuit uses the output from the input logic circuit and a 120VAC source to create an alternating voltage across a load consisting of the primary coil (PRIMARY) and two capacitors (C2 & C3). This is achieved by first using a full bridge rectifier to rectify the AC voltage. C4, R5, and R6 are added to smoothen out the voltage ripple to create a more uniform DC voltage. Then the rectified voltage is used as the source on a full bridge single-phase inverter.

The four power MOSFETs (PWR_MOS_1-4) on the inverter are driven by the gate drive transformer (GDT) which is a one to one transformer that allows for the signal from the input logic circuit to be used while also isolating one circuit from the other. To achieve the functionality of the inverter from one signal, MOSFETs 3 and 2 are connected to the GDT in reverse of MOSFETs 1 and 4. Therefore, when a HIGH signal is coming from GATE_DRIVE_IN, MOSFETs 1 and 4 will conduct, while 3 and 2 will be open and vice versa for when a LOW signal is sent. This ultimately creates the desired square wave of +170V to -170V across the primary coil and parallel capacitors.

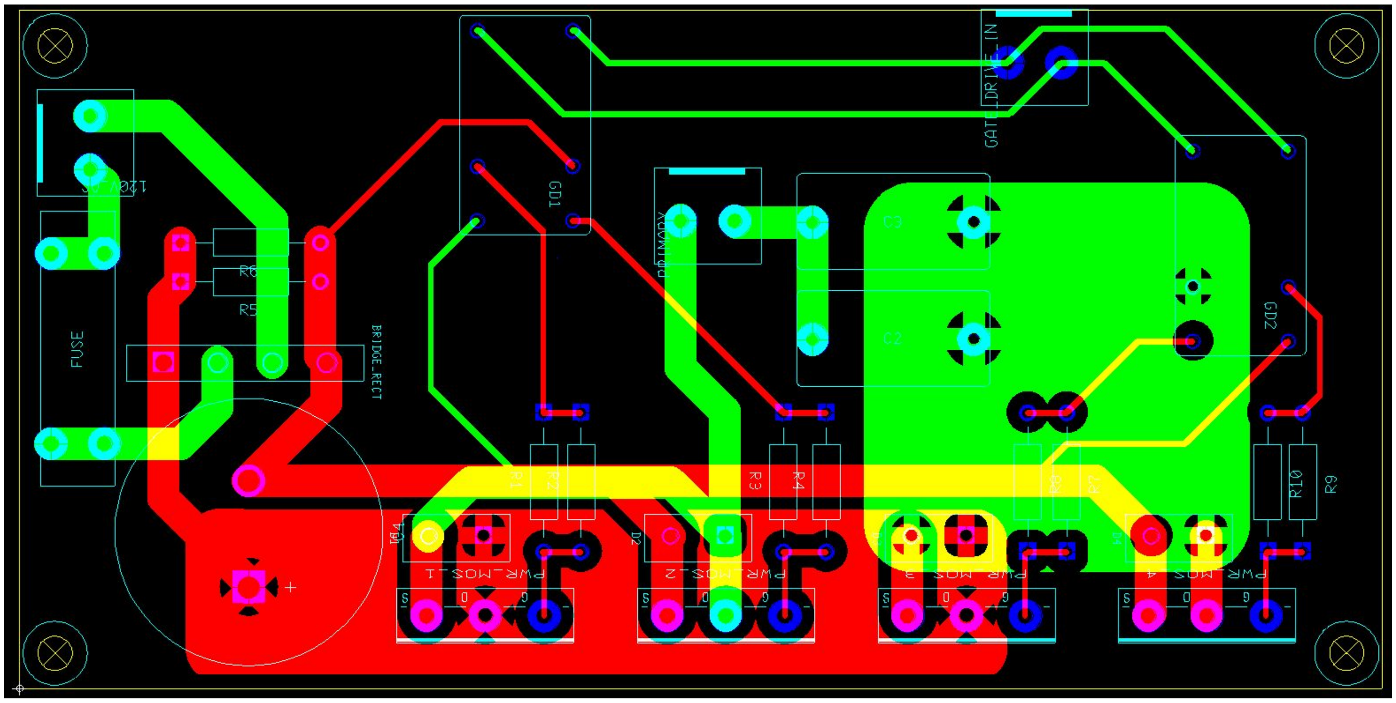


Figure 12: Bridge Printed Circuit Board Layout

3.1.2.4 Physical Construction

A case was made for both the transmitter component and the coil component of the project. The cases were made with acrylic, since the material is relatively cheap and durable. It is made with clear acrylic specifically to allow people to see the interiors of the components. The case designs were modeled in the SolidWorks CAD software and then cut with a laser cutter.

3.1.2.4.1 Transmitter Case

The transmitter case contains both the Raspberry Pi and the transmitter PCB. The transmitter PCB was created to fit on top of the Raspberry Pi and align with the Pi's screw holes. The case was made with ⅛ inch clear acrylic and cut with a laser cutter. The pieces are held together with screws and standoffs that fit through the screw holes of both the transmitter PCB and the Pi PCB. Care was taken to align holes in the case to access the buttons, switches, and ports on the PCBs.

The following figure shows the amount of the case that was designed within SolidWorks. The Raspberry Pi 3D model compliments of Richard Barber [2].

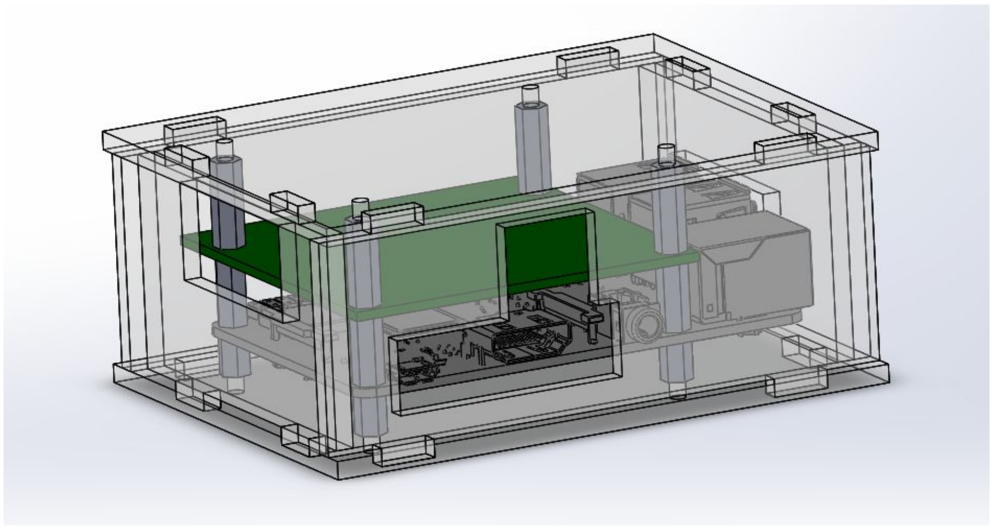


Figure 13: Transmitter Case

3.1.2.4.2 Tesla Coil Case

The case for the Tesla Coil contains the Input Logic/Power Provider PCB and the Coil Bridge PCB. The two PCBS are placed next to each other in the case, again held in place with standoffs and screws. The two PCBs are connected with wire that attaches to screw mounts on each PCB. Since the case has the secondary coil and topload attached to it as well, it was made with ¼ inch clear acrylic. The case itself was designed with an open side to allow an easy view of the eternal electronics. Two sides are made from acrylic, while the last side consists of the aluminum heat sink for the power MOSFETs in the bridge circuit. Screw holes were bored and tapped into the aluminum heat sink to allow the acrylic top and bottom to screw into the heat sink. The heat sink also had holes that were bored and tapped to allow the MOSFETs to be screwed into the heat sink. The MOSFETs each have sil pad separating them from the heat sink, allowing thermal contact while preventing electrical contact. Lastly, a hole was drilled and tapped to screw in a wire that connects the heatsink to earth ground from the power outlet. This allows the bottom the secondary to be grounded by electrically connecting it to the heatsink.

The secondary coil is attached with a PVC flange. The flange is attached to the case with screws and bolts. The PVC pipe rests inside the flange - it is a snug fit. There are holes in the top of the case for the antenna and the primary coil wire.

The following figure shows the amount of the tesla coil case that was designed within SolidWorks. The 3D models of the AC Power Receptacle and the PVC flange compliments of cdowney [3] and McMaster-Carr [8], respectively. Note the PVC flange model is slightly different than the actual flange used - the footprint of the holes on the flat side is different.

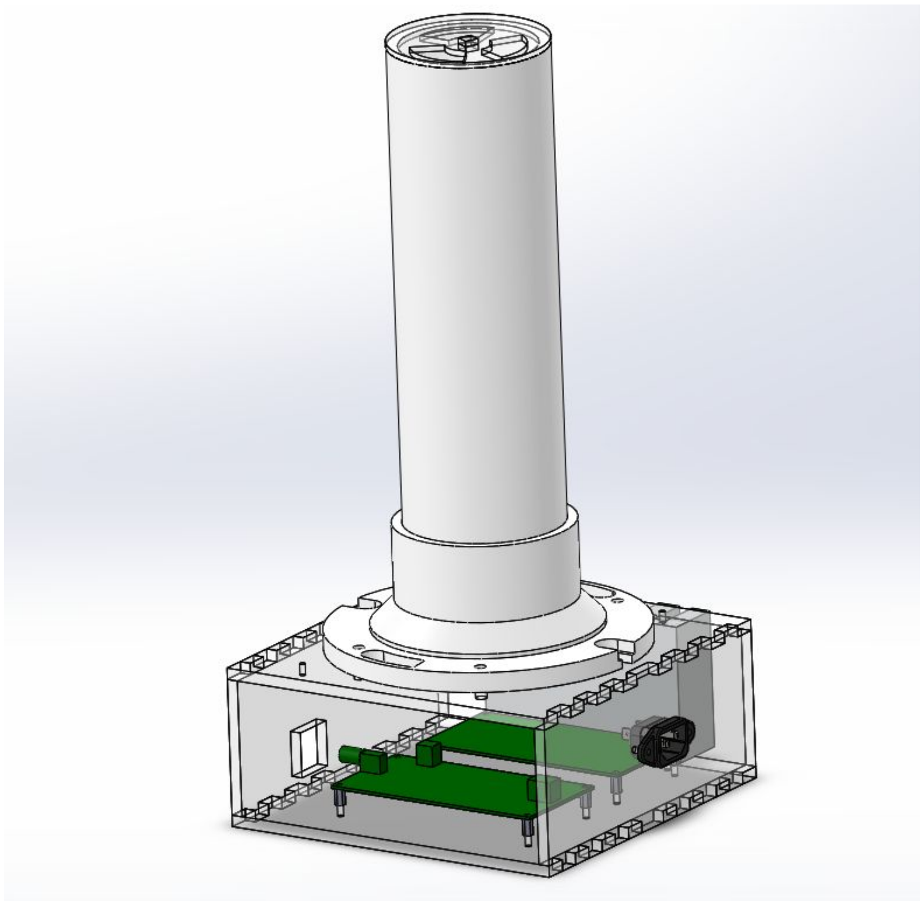


Figure 14: The Coil Case with Secondary PVC

3.1.2.4.3 Secondary Coil Winding

The secondary coil consists of 3” PVC pipe cut to 14” in length, with 30 AWG gauge magnet wire wound around the outside.

In order to wind and varnish the [number of turns here] turns on the secondary coil, a small rig was constructed out of 80/20 aluminum extrusion. The rig consists of frame slightly longer than the coil itself with arms on either end that stick up about 5 inches. ¼ inch screw holes were drilled into these arms. The coil could then be placed into the rig and spun about the center of its cylinder. In order to connect the framework to the coil, acrylic end caps were made for each end. Each end cap had two layers - one that fits just inside the cylinder, and one that fits right on top of the cylinder. The end caps had square holes cut in their center to allow a carriage bolt to fit into them. After gluing the two parts of an end cap together, a carriage bolt was secured in the hole with a washer and a nut, with the carriage bolt sticking out away from the cylinder. Special care was taken to ensure that the bolts were completely perpendicular to the end caps. These end caps were then glued onto the PVC cylinder. The bolts sticking out of the end caps can then be placed into the holes in the rig, allowing the whole coil to spin freely. A drill was attached to on of the bolts, and used to spin the coil both for winding the wire and spraying/drying the varnish.

One end cap was made with spokes instead of being solid. This allowed the bolts to be unscrewed and then slipped out of the cylinder through the holes. The other end cap was left solid to allow the topload to be screwed onto it. Both end caps were left on the cylinder in the final coil.

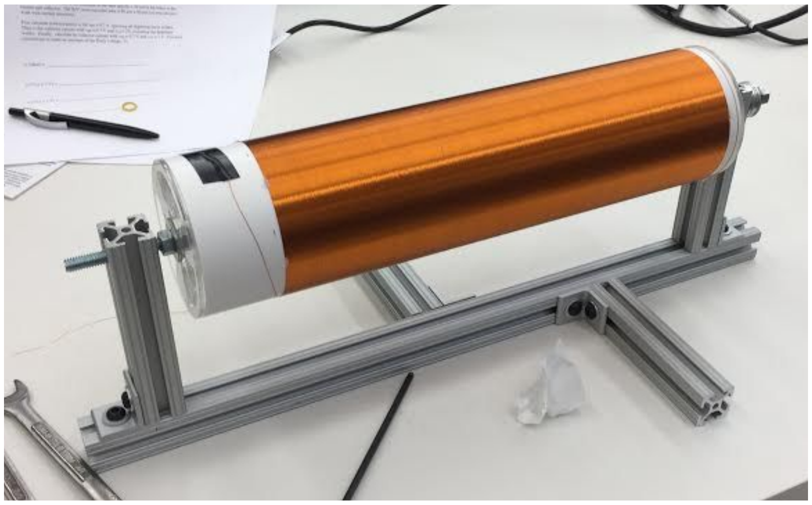


Figure 15: Winding Rig After Completion

The secondary coil has a measured resistance of 94 Ohms. With the topload, it resonates at around 290 kHz.

3.1.2.4.4 Topload Construction and Attachment

The toroidal shape of the topload was made with standard 3" heating duct, wrapped around to make a loop. The two ends are held together with aluminum tape. To attach the topload, two small disks that fit inside the heating duct loop were cut from cardboard and wrapped in aluminum foil. These were then taped to the heating duct loop. Holes were cut in the middle of the disks that are big enough to allow the bolt sticking from the top of the secondary pipe to fit through. The topload was then connected with washers and nuts to the bolt. The top of the secondary coil was taped and soldered to the topload, allowing electrical connection.

3.2 Applicable Standards and Best Practices

Several standards and best practices were used by the project. They are as follows:

- The MIDI message format was followed when receiving MIDI messages from a keyboard
- IPC-2221 for PCB trace clearances at high voltages
- Git was used for both source control and backups

4 Testing, Validation, and Evaluation

To test if the App Layer works correctly and accurately records and transmit the MIDI data, a Driver Emulator software was written. This software has the same interface as the actual driver, receiving data on turning notes on/off for specific channels via a socket connection. However, instead of interfacing with a GPIO pin and outputting voltage, the emulator creates a wav sound file from the data. This file can then be listened to in order to determine if the information was processed correctly. This software has been used to test both the keyboard input and MIDI file input programs (in the App Layer).

As we began developing our own hardware, we also purchased and constructed a OneTesla TS musical tesla coil kit. This proved an invaluable benchmark and testing tool, as we were able to modularly test some of our components against the OneTesla before we had the rest of our own tesla coil circuits built. It also gave us some experience and confidence working with tesla coils, since none of us had ever built or used one before.

The output of the driver was tested by viewing the wave on an oscilloscope. We verified that the frequency and length of the pulses were what we expected, and we also compared it to the OneTesla's transmitter output before using it with that tesla coil circuit. Once confirmed that the waveforms were similar in pulse width and frequency, it was sent to the coil with suboptimal results. The sound output contained lots of noise, and the coil would intermittently output a large spark accompanied by a larger noise. At this point, we developed the transmitter circuit to clean up the waveform. Then, we got the OneTesla running with the Raspberry Pi's wave generation through the breadboard transmitter. With the filtering circuit in place, the issues were resolved. Without the OneTesla kit to test our transmitter before we had our own coil built, we would not have discovered this issue until far later in the semester, likely with not enough time to resolve it.

A waveform generator and a power supply was used to provide controlled inputs into the circuits. This enabled testing of ideal and extreme cases of voltages and waveforms so that we could test the limits of the circuit without creating an uncontrolled and unsafe testing environment that could potentially damage test equipment, the Zeusaphone, or others nearby. This method of testing was used extensively when debugging the entire hardware system with the OneTesla coil.

4.1 Functional Testing

Since our design was very modular, we were able to test components individually before integrating them with the rest of the system. We were also able to test components together with modules from the OneTesla kit, when our own modules were not constructed yet. At each stage, we verified the expected behavior before moving on to the next component or integrating the system. Thorough testing of the system helped ensure that our entire project was a success once all of the pieces were complete and integrated.

4.1.1 Application Software Layer Testing

The application software layers, taking input from MIDI files and the MIDI keyboard, were first tested with the driver emulator software. Thus we were able to read MIDI input and verify that the correct songs were synthesized before we had even a single circuit built to test on.

4.1.2 Driver Software Layer Testing

The driver software, which took the notes and output pulses on the GPIO pins, were then tested with an oscilloscope. We sent a single output frequency through the driver layer, and compared the frequency of the output to the input frequency. We also could plug the output of the pins into a speaker and listen to the audio that was played, verifying that the song was playing correctly. It was in this stage of testing that we discovered that the software-driven PWM pins of the Raspberry Pi did not provide enough stability for the project, and we switched to the hardware-driven PWM pins.

4.1.3 Transmitter Circuit Testing

The transmitter circuit was built in iterations on a breadboard, so that we could easily make modifications as we needed. We tested this circuit by examining the output to the fiber optic transmitter on the oscilloscope, and verifying that the frequency and pulse width matched what we were expecting. We were also able to use our transmitter prototype to drive the OneTesla coil setup. It was in this stage of testing that we found the issue with the overlapping pulses described in section 3.1.2.1 and updated our circuit to fix it.

Once we were confident in our transmitter circuit design, we laid it out on a PCB to be manufactured. When we had received and constructed the PCB, we again tested the circuit by looking at the output frequency and pulse width. We initially had an issue; however, we realized that we had not soldered all of the pins into the board properly, and when we resoldered the board it performed as expected.

4.1.4 Tesla Coil Circuit Testing

All circuits were tested in breadboard form with low voltages to make sure the overall design would work. We constructed a ‘mini’ coil to test our circuit designs at low voltages where a mistake would not lead to the destruction of the circuit or the coil. The ‘mini’ coil proved to be extremely useful as it added a layer of preliminary testing to debug some problems and finally confirm our designs. A major milestone was when we could play music on the mini coil using hardware entirely designed ourselves with no components from the OneTesla kit. This setup involved the Raspberry Pi running our software, the breadboard version of the transmitter, the breadboard low-voltage bridge circuit, and the mini coil. We used the oscilloscope to verify the waveforms on several pins throughout the circuit, and we were also able to view the induced voltage from the coil in the air to verify that the coil was oscillating properly and generating the correct frequencies.

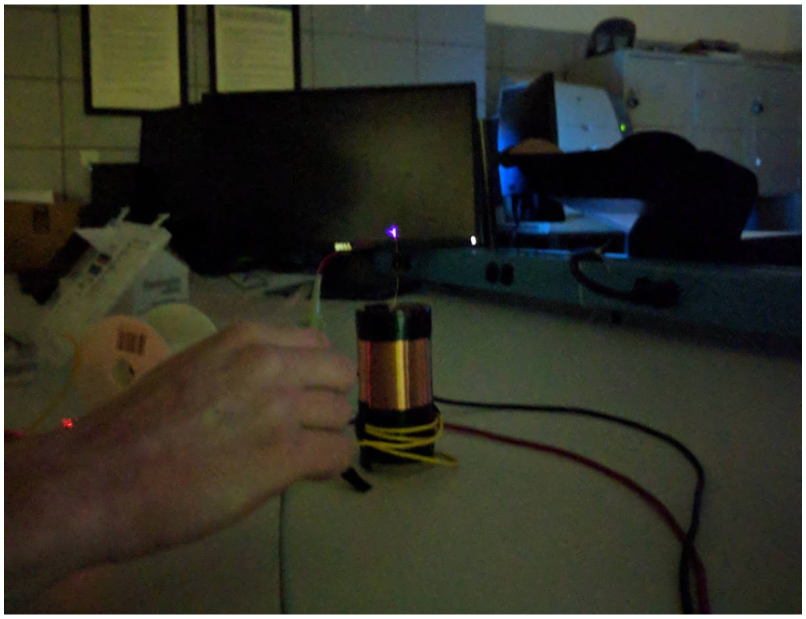


Figure 16: The Mini Coil

After confirming functionality on the breadboard, high voltage versions of the circuits were prototyped on perfboard, which was connected to the primary coil from the OneTesla kit. We did not use a perfboard iteration for the transmitter circuit, since it did not use high power components. Before designing and ordering PCBs, we made sure the circuits displayed the same behaviour when running on high voltage. The major issue we discovered at this stage was that the higher voltage coil suffered from noise induced by itself back in its control circuits. We traced the noise using the oscilloscope to the antenna input, and when we touched the oscilloscope to this pin the noise disappeared. We deduced that the pin was floating, and we solved the issue by adding a pull-down resistor.

Finally, we tested the driver circuits with our own coil. The main thing we needed to test was the control signal pulse-width needed to make the coil to output a good volume with visible sparks. This was done by sending pulses across a fiber optic transmitter using a function generator. With this setup, we could easily change the frequency and pulse widths of the signals sent. During this testing, we pushed the coil up to 100 μs pulse width, which was impressive but a little too loud. To be on the safe side, we used a resistor in our transmitter that gave about an 90 μs pulse width.

During this testing, we also discovered some glowing about the primary coil, as seen in the picture below. We reasoned that the issue was due to the primary coil being about an inch up the secondary coil, where the voltage had already been stepped up high enough that it was trying to arc back to the primary coil. Moving the primary coil windings down below the secondary coil solved this issue.

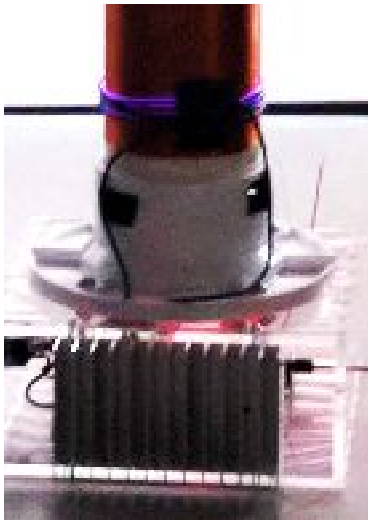


Figure 17: Glowing around the primary coil

4.1.5 Integration Testing

Once all of our components were complete and their functionality verified individually, we tested the full system. We used the web API to upload songs and play them, and also activate the live keyboard input. Since we had tested each component so thoroughly individually, and we had such a clear standard of how the components would work together, our first full system test went flawlessly and we discovered no major issues to address.

4.2 Non-Functional Testing

In the process of running the coil for different experiments, stress testing has been indirectly performed. The coil has been ran for at least an hour at a time without problems, and the setup process has been performed many times. Nothing in the setup is particularly fragile or difficult.

5 Project and Risk Management

5.1 Task Decomposition & Roles/Responsibilities

Gunnar Andrews

- Raspberry Pi configuration (Access point, DNS, Static IP, Apache Web Server)
- Driver software
- Application software (play midi files, play keyboard/pitch bending)
- Documentation
- Status Report uploading and editing
- Webpage upkeep and editing
- Web API front end (HTML webpage with back end hooks)
- Bootstrap 4 web design (for web page)
- PHP back end to interface with driver/application layers of software
- Testing hardware and software

William Brandt

- Tesla coil pre research
- Bridge circuit research
- Circuit schematic
- Safety document

Jacob Feddersen

- Team communications and scheduling
- Midi File App
- Raspberry Pi Driver Software
- Mini-coil construction and testing
- Circuit design and schematics
- Prototype circuit construction, soldering, and testing
- Circuit prototype, PCB design, and testing
- Tesla coil hardware construction
- Tesla coil integration testing and troubleshooting

Leo Freier

- Raspberry Pi Driver Software
- Some Web API help
- Mini-coil testing
- Prototype circuit testing
- PCB design and testing
- Tesla coil construction and testing
- Documentation and User Manual

Gregory Harmon

- Tesla coil research
- OneTesla soldering
- Circuit analysis of bridge and transmitter
- Theory of operation
- Bridge circuit part sourcing

Luke Heilman

- MIDI Keyboard App
- OneTesla soldering and testing
- Mini-coil testing
- Circuit design and testing
- CAD modeling in SolidWorks
- Acrylic laser cutting
- Coil hardware construction
- Soldering our PCBs
- Tesla coil testing and troubleshooting

5.2 Project Schedule (Gantt chart, proposed vs. actual)

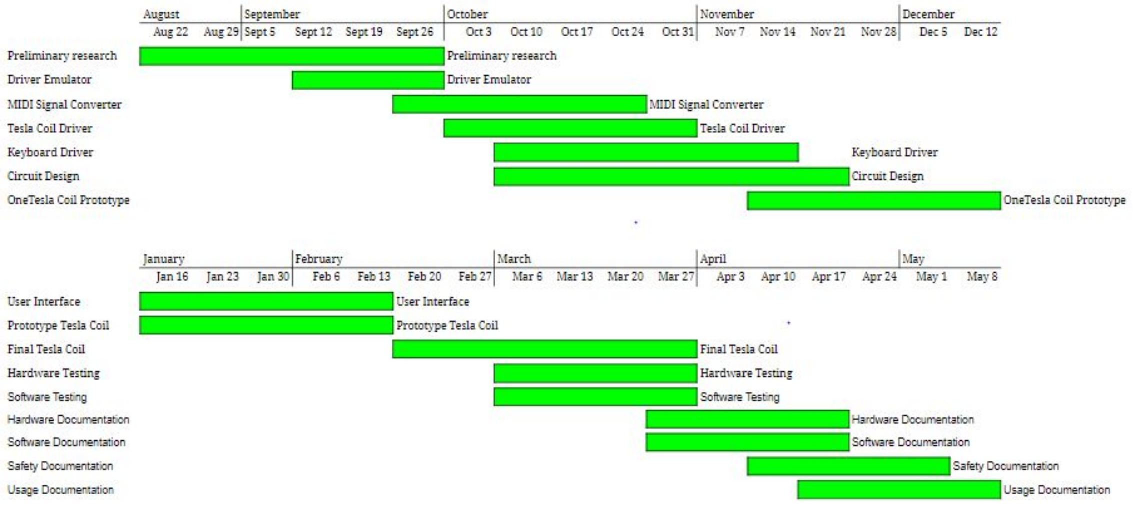


Figure 18: Proposed Gantt Chart

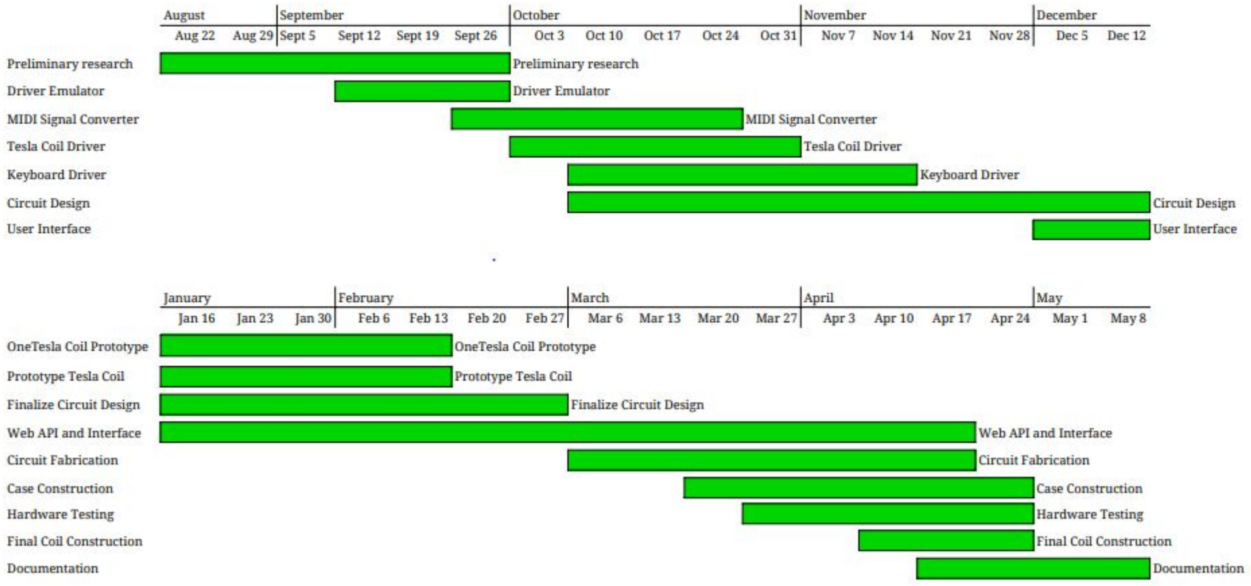


Figure 19: Actual Gantt Chart

5.3 Risks and Mitigation: Potential and Actual

Safety risks were a large concern during the entire project. Working with high voltage electronics always requires care, and a tesla coil is no exception. Circuits were always tested at a lower power first, to ensure that the design worked as intended before ramping up the voltage. Whenever operating a coil, care was taken to have people and electronics at least 10 feet away to prevent them from being harmed. For safety in the future, a safety document has been made. Thankfully, no one and nothing was harmed due to these precautions.

Another large risk, due to the nature of a tesla coil, was that components running at a high voltage would burn out or break. Again, by running the circuits at a lower voltage first, many problems were found and solved before running the circuits at higher voltages. The tesla coil, if overloaded, has the ability to destroy itself. Care was taken to make sure signals sent to the coil were what was intended by observing them in an oscilloscope first.

The team make-up itself was also a risk - the project was predominantly a hardware one, but the team has only 2 EEs and 4 CprEs, all of which had little hardware experience. This risk was mitigated by having 3 of the CprEs spend considerable time learning new hardware concepts and stepping into the hardware world. This was especially true with the PCB fabrication, as no one in the group had prior experience with designing PCBs.

5.4 Lessons Learned

The team initially struggled with the hardware side of the project, because it was a large aspect of the system, and most of the team had little experience working with power electronics. We learned that hardware development takes a lot of time, which we expected. Since we realized that a lot more work would be required on hardware, we made a hard transition into hardware work which is why we ended up completing the project. The team used this as an opportunity to expand their knowledge base, and in the end, created a finished product.

Another lesson to take from the project is that working as a team is much more successful when physically working together. In the first semester, we accomplished a lot but not nearly as much as the second semester. We attribute a large part of this to the fact that the team would meet in person to work many times a week, even if people were not working on the same part. Work sessions like this improved team communication and effectiveness tremendously.

6 Conclusion

6.1 Closing Remarks

The project was a success! A zeusaphone was created that fits the design specifications given by the client. It plays music from a MIDI keyboard, and outputs clean tones. The system is reliable, and relatively easy to setup and use. To aid with this, a user document and a safety document were also created. The project will make a great showpiece for the ECpE department, which shows what EEs and CprEs are capable of creating.

6.2 Future Work

While the project was successful, there are many things that could be improved. The transmitter has a lot of room for improvement. While the 555 timers do prevent waveforms from overlapping, they also require a fixed pulse-width. Varying the pulse-width varies the volume and size of sparks output from the secondary. The team also discovered that different audio frequencies sound better at different volumes. Thus designing a transmitter that can create variable pulse-widths would provide more functionality to the system. This could be accomplished with either an arduino or an FPGA, which would be used in addition to the Raspberry Pi.

The topload construction of the coil leaves a lot to be desired. Admittedly, the coil construction was rushed due to time constraints on the project. A more sturdy topload could be created by cutting a specific shape out from aluminum or another conductive metal.

Another simple upgrade which we did not have time to explore using a current transformer on the base of the secondary coil instead of an antenna for feedback. While we have not had any trouble with the antenna once the noise issue was solved, switching to current feedback would make the coil a little bit more self-contained, and it would not require the antenna to be set up protruding from the base of the tesla coil.

Another improvement would be to upgrade the coil from a SSTC to a DRSSTC (dual-resonant solid-state tesla coil). In this variant, the primary coil circuit is tuned to the same frequency as the secondary coil and the two resonate with each other. This design presents quite a few additional challenges which would need to be solved; much higher power output can be generated, and the components must be able to handle the higher currents. More complicated control circuitry is also required to prevent the bridge from being destroyed by hard-switching.

7 References

Design References:

[1] abyz.me.uk/rpi/pigpio/index.html. *pigpio library.* [online]. Available at: abyz.me.uk/rpi/pigpio/.

[2] Barber, Richard, "Raspberry Pi 3 Model B+," *grabcad*, 3-Mar-2018. [online] Available at: https://grabcad.com/library/raspberry-pi-3-model-b-2

[3] cdowney, "IEC 320-C14 Receptacle Panel Mount," *grabcad.* 21-JUL-2016. [online] Available at: https://grabcad.com/library/tag/703w-00-04

[4] Kaizer Power Electronics. (2012). *Kaizer DRSSTC II*. [online] Available at: http://kaizerpowerelectronics.dk/tesla-coils/kaizer-drsstc-ii/ [Accessed 1 Dec. 2018].

[5] Kaizer Power Electronics. (2016). *Musical SSTC/DRSSTC interrupter*. [online] Available at: http://kaizerpowerelectronics.dk/tesla-coils/musical-sstcdrsstc-interrupter/ [Accessed 1 Dec. 2018].

[6] Kane, Phillip, "555 Timer Tutorial," *jameco.com,* 2019 [online] Available at: https://www.jameco.com/jameco/workshop/techtip/555-timer-tutorial.html

[7] Learn.adafruit.com. (2018). *Setting up a Raspberry Pi as a WiFi access point*. [online] Available at: https://learn.adafruit.com/setting-up-a-raspberry-pi-as-a-wifi-access-point/overview [Accessed 1 Dec. 2018].

[8] McMaster-Carr, "Flanged Socket-Connect Reducing Adapter, for PVC Pipe for Drain, Waste and Vent," 2389K91 Datasheet. [online] 2015 Available at: https://www.mcmaster.com/2389k91

[9] Music.mcgill.ca. (2017). *The RtMidi Tutorial*. [online] Available at: https://www.music.mcgill.ca/~gary/rtmidi/ [Accessed 1 Dec. 2018].

[10] oneTesla.com. (n.d.). *oneTeslaTS Schematic*. [online] Available at: http://onetesla.com//media/wysiwyg/downloads/tsschem.png [Accessed 2 Dec. 2018].

[11] Sapp, C. (2018). midifile. [online] Available at: https://github.com/craigsapp/midifile [Accessed 2. Dec. 2018].

[12] Thestk, "thestk/rtmidi," *GitHub*, 14-Sep-2018. [online]. Available: https://github.com/thestk/rtmidi.

[13] Steve Ward High Voltage. (2009). *New DRSSTC Driver.* [online] Available at: http://www.stevehv.4hv.org/new_driver.html

8 Team Information

Gunnar Andrews is a computer engineer from Cottage Grove, MN. His plans after graduation will be working as a Security/Software Engineer for Optum in Eden Prairie, MN. He will be working to develop and revise software that is both safe and SECURE that will better the healthcare industry. Permanent email address: g97andrews@yahoo.com

William Brandt is an electrical engineer from Clarinda, IA. His plans after graduation will be working as a project engineer at ITC Holdings in Cedar Rapids, IA. This will involve running power flow simulations to ensure projects meet standards. Permanent email address: wbrandt97@gmail.com

Jacob Feddersen is a computer engineer from Ames, IA. After graduation, he is staying in Ames to work at Workiva developing cloud software for the financial and regulatory industry. Permanent email address: jtfedd@gmail.com

Leo Freier is a computer engineer from Scandia, MN. He will be working as a Software Engineer at Open Systems International Inc. in Medina, MN, working on power grid automation.

Gregory Harmon is an electrical engineer from Crystal Lake, IL. He will be working as an Electrical Field Construction Coordinator at Blattner Energy in Avon, MN. Permanent email address: gwrighth@gmail.com

Luke Heilman is a computer engineer from Ames, IA. After graduation, he is staying in Ames to work at Renewable Energy Group in their IT department. Permanent email address: lsheilman2015@gmail.com