Remotely Connected Electric Field Generator for Particle Separation in a Fluid
Team May1612

Dee, Timothy
timdee@iastate.edu

Long, Justin
jlong@iastate.edu

McDonnel, Brandon
bmcdonnel@iastate.edu

February 14, 2016

Abstract

1

Introduction Project Definition New research has shown that certain particles may be separated from fluids through dielectrophoresis. This process involves applying an electric field to a fluid. The field can be manipulated in order to attract or repel certain particles. This electric field is applied through metal plates. Our job is to construct a circuit which will drive these plates. This circuit must be controllable from a web interface and have a small form factor. It will be able to generate up to a 60 V peak-peak sine wave, and the frequency can be changed from 10 kHz to 1 MHz. Deliverables There are four items which must be constructed for this project. For the analog circuit components we will be able to test the functionality of the circuit by using an oscilloscope to verify that the requirements have been met and that there is minimal noise and distortion.

The construction of this device is the first phase of the project. After this has been completed we will need to use the device to experiment with particle separation in fluids. These experiments constitute the remainder of the project. For these experiments our advisor at Minetronix, John Pritchard, will be the main source of direction and testable material. Constraints Our client at Minetronix has set out a number of constraints that we must follow for the final delivered project. These constraints are based on size, voltage, and portability.

The size requirements of the project are directly related to the portability of the final design. The design has been specified to be able to be moved around from one workstation to another easily and quickly. The closest size description that we have received was about the size of a backpack with the implicit description of possibly being smaller. With the electronics we are using, baring a excessively large power supply, we should be able to easily meet these requirements.

The only other constraint that we might meet would arise from the power supply. We require a 60V power supply to feed into the amplifier portion of the circuit. This means that the device will most likely be attached to a power brick that requires being plugged in. This leads to the need to be within easy reach of a wall plugin, which should not be a problem in most, if not all, testing environments. System Level Design System Requirements The system requirements call for several different systems to run or power our devices. The first requirement would be a connection to a computer to be able to interact and use the web interface on the raspberry pi that runs the device. Without a computer to interact with you will have no practical means to change the device's function. The next requirement is a connection to the raspberry pi itself. The third system requirement is a standard wall outlet to plug the device into. The finished product will require a amplifier circuit from 3.3V AC to 60V AC which is most easily handled by regulating the usual 120V sine wave. Functional Decomposition There are four large blocks in our system. They include the Web Interface, Raspberry Pi, Minigen Signal Generator, and Amplifier Circuit. The project will be described in terms of these pieces.

The web interface will be created using an Apache web server. We will be able to use the Raspberry Pi to host the web server. The web server will need to display an interface which will allow the user to set the voltage and frequency. A simple interactive interface can be created with cgi scripts hosted by the webserver.In addition to hosting the web server the Raspberry pi can be used to regulate the voltage and frequency output by the system. The SPI interface of the raspberry pi will be used to communicate with the Minigen which controls the frequency and digital potentiometers which regulate the voltage.

The Minigen produces a frequency which is a function of the values contained in its frequency registers. These values can be modified though SPI communications with the Minigen. We will use the Raspberry Pi for these communications.

The Raspberry pi also controls the resistance of several digital potentiometers. These potentiometers regulate the gain of an amplifier circuit. The power from the amplifier will need to come from a voltage source which will supply at least 60 Vpp. System Analysis A user will interface with this system though the web interface. The interface will allow the user to choose the values for Voltage and Frequency. Once the user enters these values update scripts will run on the Raspberry pi. These scripts will cause the appropriate values to be set in the Minigen and Digital potentiometers. This will cause the output of the circuit to change to the requested values. Block Diagrams

standard non-inverting amplifier circuit Detail Description 2.1 - Web Server The primary function of the webserver will be to communicate with the Raspberry Pi. This will be the primary method of control utilized by the user. The web pages displayed by the server will have the ability to control the voltage and frequency output from the circuit. A simple web interface might look something like the following.

Figure 2: Web Interface

This could be accomplished by running an apache web server on the Raspberry Pi. The web server would need to display this page. When the user clicks update, the server could execute a cgi script which would perform the update functions. 2.2 - Raspberry Pi

The Raspberry Pi will act as the bridge between the user and the circuit. The Pi will host a webserver which the user can interact with. Based on what the user indicates in this interaction, the Pi will update the state of the GPIO pins. The GPIO pins connect to a circuit causing the output to change based on their state.

The Raspberry pi is capable of producing square waves by turning the GPIO pins on and off rapidly. We can use this functionality to produce a wave of the frequency indicated by the user. The GPIO pins can also be used to set the voltage by communicating with the circuit how much the output waveform should be amplified. The downside to this approach is the analog circuit component will need to be more complex. The analog circuit needs to output a sine wave. With this approach we would need to integrate the square wave produced by the GPIO pin.

There exist alternatives to using the GPIO pins to generate a signal with a given frequency. We could instead use the GPIO pins on the raspberry pi to communicate with a small signal generator, such as sparkfun.com s Minigen. This would make programming the Raspberry pi more complex, but could lead to higher quality waveforms. Producing a sine wave using the Minigen signal generator is likely to to produce fewer distortions compared to integrating a square wave produced by the RPIs GPIO pin twice. 2.3 - Minigen

The minigen outputs a waveform from -3.3V to 3.3V, this will be the starting point before going into the amplifier circuit. The minigen communicates over SPI, which the raspberry pi has dedicated modules for. The minigen is controlled by setting five registers, two for frequency, two for phase shift and one as a control. We have no need for phase shifting, but will be communicating with the frequency and control registers. By having two frequency registers, we are able to send data to reg0, then tell the control reg to use reg0, this allows for a nicer gradient, because the frequency wont change until the entire register is written. The control register also allows for changing between sine, square and triangle, although this doesnt interest us at the moment, it may be nice to experiment with later on. Finally, the bottom half of the control register allows us to switch between writing to the top half, bottom or whole frequency register, giving us the ability to accurately dial in small changes to the register, or large changes, or just rewrite the entire frequency. Due to the small chip size, it will be able to fit into a case with the raspberry pi, allowing for a small footprint, a requirement we need to meet.

2.4 - Amplifier Circuit

Takes input from the Minigen signal generator. Based on this input the circuit will manage the voltage and frequency of the output.

The Minigen will generate the signal applied to the input of the analog circuit. We only need a method of producing the correct voltage. One way we could accomplish this is to communicate to the circuit what the voltage should be using the GPIO pins on the PI to control a digital potentiometer. Like the minigen, we would use SPI to communicate with this component. Such a circuit might look like the following with one of R1, R2 being a digital potentiometer.

Figure 3: Voltage chooser circuit

The project requires that we generate signals which range from 1 to 60 vpp. The output of the digital potentiometer has 128 steps. This Translates into our ability to set 128 different gains on our amplifier. We will need multiple stages of amplifier to go between 1 and 60 Vpp. The most prominent reason for this is due to the gain bandwidth of the op-amps. We will not be able to have a large gain while still producing a frequency of 1Mhz.

One problem we foresee with the digital potentiometer is that it cannot handle a large amount of power. This may force us to come up with different amplifier configurations, or use the digital potentiometer in a different way. Another way we could possibly use this device is as an attenuator at the input to the amplifier.

Another problem which might arise with the digital potentiometer is the capacitance of the wiper. We dont have any context for understanding how much this will affect the output signal. According to some preliminary calculations, we have determined that the capacitance will not present a large problem.

1 - Problem Statement New research has shown that certain particles may be separated from fluids through dielectrophoresis. This process involves applying an electric field to a fluid. The field can be manipulated in order to attract or repel certain particles. This electric field is applied through metal plates. Our job is to construct a circuit which will drive these plates. This circuit must be controllable from a web interface and have a small form factor. It will be able to generate up to a 60 V peak-peak sine wave, and the frequency can be changed from 10 kHz to 1 MHz. For the analog circuit components we will be able to test the functionality of the circuit by using an oscilloscope to verify that the requirements have been met and that there is minimal noise and distortion.

The construction of this device is the first phase of the project. After this has been completed we will need to use the device to experiment with particle separation in fluids. These experiments constitute the remainder of the project. For these experiments our advisor at Minetronix, John Pritchard, will be the main source of direction and testable material. 2 - Project Design Figure 1: Block diagram, overview of components 2.1 - Web Server The primary function of the webserver will be to mediate communication between the user and the Raspberry Pi. The user will be able to connect from any networked device such as a laptop, tablet, or computer. This will be the primary method of control utilized by the user. The web pages displayed by the server will have the ability to control the voltage and frequency output from the circuit. A simple web interface might look something like the following.

Figure 2: Web Interface

In the above image, the user can control the voltage, frequency, and waveform type. In addition, the table provided allows the user to set voltage and frequency values which will be held for the specified amount of time. The motivation behind this is that DEP takes a substantial amount of time. The ability to hold values of Voltage and Frequency for minutes, even hours, may prove useful.

This web interface could be accomplished by running an apache web server on the Raspberry Pi. The web server would need to display this page. When the user clicks update, the server could execute a cgi script which would perform the update functions. 2.2 - Raspberry Pi The Raspberry Pi will act as the bridge between the user and the circuit. The Pi will host a webserver which the user can interact with. Based on what the user indicates in this interaction, the Pi will update the state of the GPIO pins. The GPIO pins connect to a circuit causing the output to change based on their state. Right now all of our components communicate through SPI.

One device we are communicating with is the SparkFun Minigen. There are two frequency registers, two phase registers, and a control register on the device. Depending on the registers we write though SPI, this device will produce sine, triangle, and square waveforms between 1Khz and 4Mhz. This is more than enough to meet our specification. We will then be able to apply the output of this device to our amplifer circuit.

The other device which we are connecting to though SPI is a digital potentiometer. This device allows us to set a resistance up to 10k Ohms. When combined with an amplifier circuit, this will allow us to set the gain of the amplifier. The digital potentiometer has 128 steps between 0 and 10k Ohms. This means we will be able to achieve a granularity of around .5Vpp. 2.3 - Analog Circuit Takes input from the Minigen signal generator. Based on this input the circuit will manage the voltage and frequency of the output.

The Minigen will generate the signal applied to the input of the analog circuit. We only need a method of producing the correct voltage. One way we could accomplish this is to communicate to the circuit what the voltage should be using the GPIO pins on the PI to control a digital potentiometer. Like the minigen, we would use SPI to communicate with this component. Such a circuit might look like the following with one of R1, R2 being a digital potentiometer.

Figure 3: Voltage chooser circuit

The project requires that we generate signals which range from 1 to 60 vpp. The output of the digital potentiometer has 128 steps. This Translates into our ability to set 128 different gains on our amplifier. We will need multiple stages of amplifier to go between 1 and 60 Vpp. The most prominent reason for this is due to the gain bandwidth of the op-amps. We will not be able to have a large gain while still producing a frequency of 1Mhz.

One problem we foresee with the digital potentiometer is that it cannot handle a large amount of power. This may force us to come up with different amplifier configurations, or use the digital potentiometer in a different way. Another way we could possibly use this device is as an attenuator at the input to the amplifier.

Another problem which might arise with the digital potentiometer is the capacitance of the wiper. We dont have any context for understanding how much this will affect the output signal. According to some preliminary calculations, we have determined that the capacitance will not present a large problem. 3 - Work Breakdown 3.1 - Timeline

Completion Date Object Description 10/1 Project Plan Create a project plan which specifies the pieces of the project. 10/15 Design Complete a detailed design of each component of our project. Assign people to work on the various pieces of the project. 11/1 Design Web Interface Design and build web interface. Outline code for Communications with Minigen and Digital Potentiometers. 11/15 Hardware Communications and Design Get communications working between Raspberry Pi, Minigen, and Digital Potentiometers. 12/1 Completion of Prototype Take final steps testing prototype. Device should be able to do everything in specification. 12/15 Presentation Present a working prototype of our project. 1/15 Begin Science Component Begin working on Research component of the project. 2/1 - 5/15 Continue Experimenting Use the project to perform some sort of research TBD

3.2 - Risk/Feasibility The risk for our current solution to the problem will come with the interface between the Raspberry Pi and the analog circuit. We have already planned to use a Minigen signal generator from sparkfun.com. With this device it would cause us to have more complex software interface between the Raspberry Pi and the signal generator but it would give us a less complex circuit. The amplifier portion of the circuit, connected to the output of the minigen, is what may prove to be our greatest barrier.

3.3 - Cost Considerations The overall cost of our project will not be very much money. We do not know the exact op-amps other electronic components we will be using, but the costs will be minimal for those parts. The main cost of the project will come in the cost for the Rasperry Pi 2 and/or the Minigen from sparkfun.com if we end up using that. The Raspberry Pi 2 package is $99.95 and the Minigen is $29.95. Thinking conservatively and assuming we need both the Raspberry Pi 2 and the Minigen, the cost of the major hardware will be $129.90. We then need to add in the cost of a resistor kit, a capacitor kit, and a handful of op-amps so that we safely budget for any parts we might need and extras for anything we break or that is defective. On sparkfun.com they have a resistor kit for $7.95 and this would have any resistors we would need and then some. As far as capacitor kits they only have very large kits or small kits with random values of capacitance. It may be cheaper to buy individual capacitors from sparkfun.com for $0.25 each. Figuring 20 capacitors giving plenty needed and some extras would add $5.00. Each of the op amps are $0.95 on sparkfun.com and we will budget for 10 of them to again make sure we have plenty of resources available to use if needed. This would add $9.50 to the cost of materials and would be the last component we need. Adding all the parts together gives a rough estimate of $152.35 which is way below the maximum allotted funds of $1,000 we are allowed to use according to the initial problem description.

Total Cost Raspberry Pi 2 Kit - $99.95 Minigen - $29.95 Resistors - $7.95 Capacitors - $5.00 Op Amps - $9.50 Estimated Total = $152.35

2 Appendix

2.1 Graphical Comprehension Aides

2.2 Code Listing - HTML

Listing 1: Main HTML Page

```
<html>
<head>
<style>
#header {
    background-color:black;
    color:white;
    text-align:center;
    padding:5px;
}
#nav {
    line-height:30px;
    background-color:#eeeeee;
    height:300px;
    width:100px;
    float:left;
    padding:5px;
```

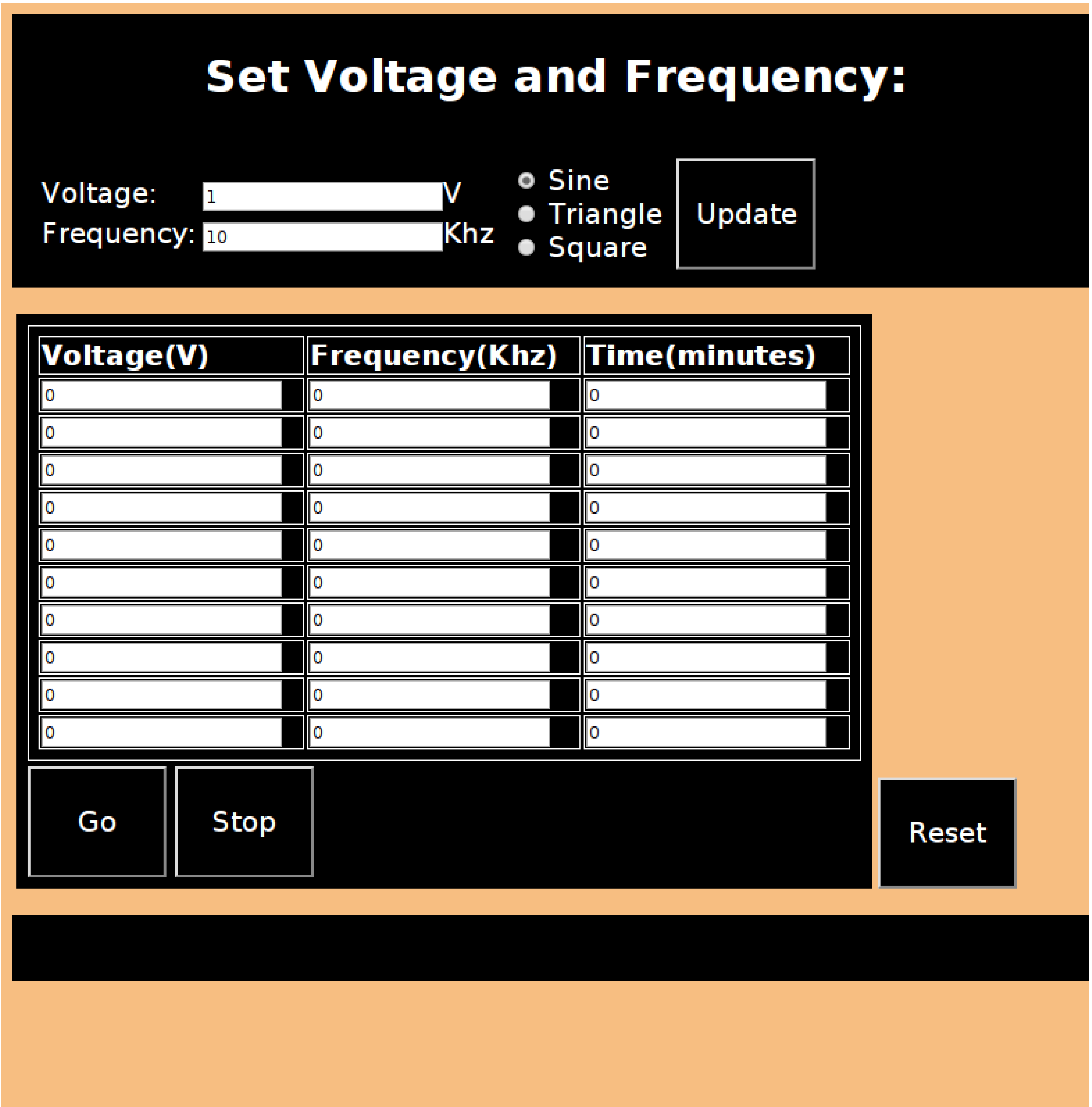

Figure 1: Web Interface

```
}
#section {
        width:350px;
        float:left;
        padding:10px;
}
.button{
        width:100px;
```

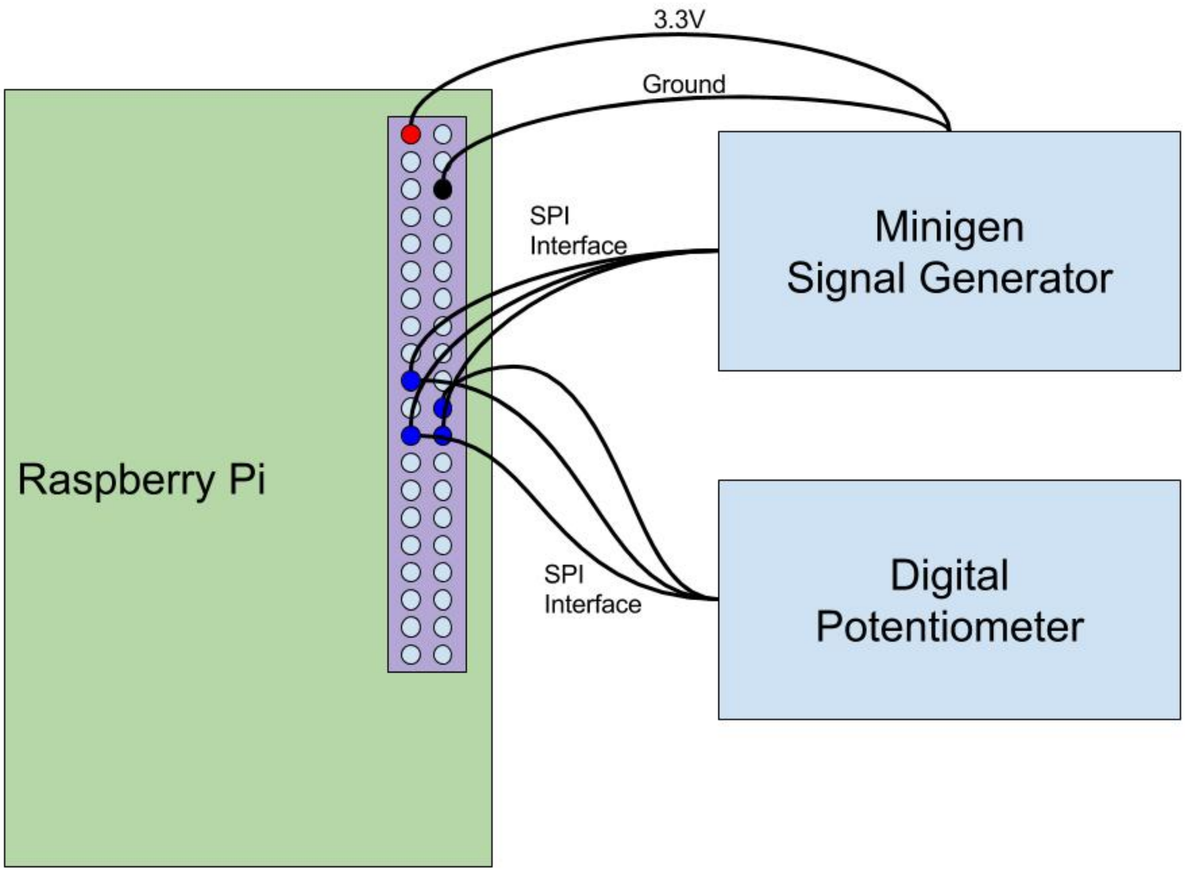

Figure 2: Raspberry Pi Connection Scheme

```
    height:80px;
    background-color:black;
    color:white;
    clear:both;
    text-align:center;
    font-size:20;
}
.text {
    color:white;
    font-size:20;
}
.text_div {
    background-color:black;
    color:white;
    clear:both;
    text-align:left;
    padding:5px;
    font-size:20;
}
```

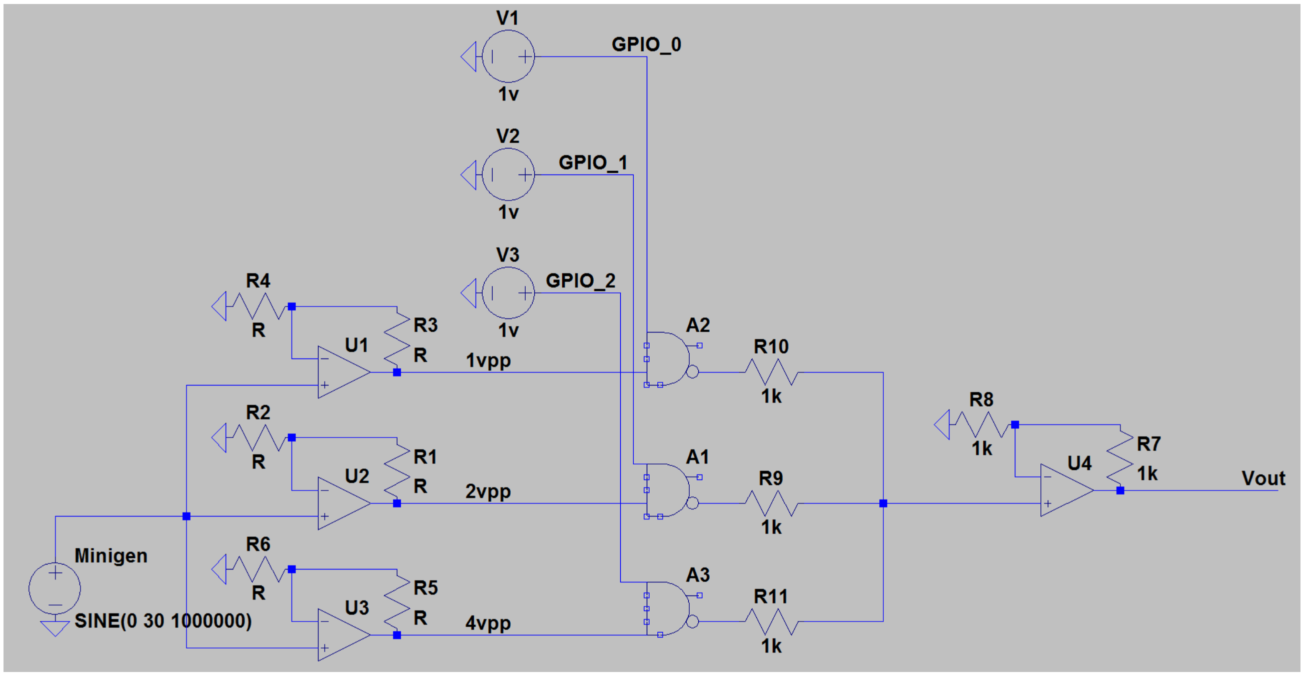

Figure 3: Voltage Control Circuit

```
.borderless_table {
    background-color:black;
    color:white;
    clear:both;
    text-align:left;
    padding:5px;
    font-size:20;
}
.table {
    background-color:black;
    color:white;
    clear:both;
    text-align:left;
    padding:5px;
    font-size:20;
    border: 1px solid white;
}
.th {
    border: 1px solid white;
}
.tr {
    border: 1px solid white;
}
.td {
    border: 1px solid white;
}
#reset{
    text-align:right;
```

```
    vertical-align: bottom;
}
#footer {
    background-color:black;
    color:white;
    clear:both;
    text-align:center;
    padding:5px;
}
</style>
</head>

<body bgcolor=”#F7BE81”>

<div id=”header”>
  <h1>Set Voltage and Frequency:</h1>
</div>

<!-- begin content -->
<div>
  <form action=”/cgi-bin/update.py” method ”post”>
  <div>
    <div class=”text_div”>
      <!-- use a table to align the table with the update button -->
      <table class=”borderless_table”>
        <tr>
      <!-- use a table to align everything -->
          <td>
          <table class=”borderless_table”>
          <tr>
            <td>Voltage:</td>
            <td><input type=”text” value=”1” name=”voltage”></input>V</td>
          </tr>
          <tr>
            <td>Frequency:</td>
            <td><input type=”text” value=”10” name=”frequency”></input>Khz</td>
          </tr>
          </table>
          </td>

          <td>
           <!-- radio buttons to select waveform -->
            <div>
              <input type=”radio” name=”waveform” value=”sine” checked> Sine
              <br>
              <input type=”radio” name=”waveform” value=”triangle”> Triangle
              <br>
              <input type=”radio” name=”waveform” value=”square”> Square
            </div>
          </td>

          <td>
            <input class=”button” type=”submit” value=”Update” />
          </td>
```

```

          </tr>
        </table>
      </div>
    </div>
    </form>

    <table>
    <tr>
    <td>
    <!-- make a table that will cause updates at the times specified -->
    <form>
    <table class=borderless_table>
      <tr>

      <td>
      <table class=”table” style=”width:600px”>
        <tr class=”tr”>
          <th class=”th”>Voltage(V)</th>
          <th class=”th”>Frequency(Khz)</th>
          <th class=”th”>Time(minutes)</th>
        </tr>
        <tr class=”tr”>
          <td class=”td”><input type=”text” value=”0” name=”v0”></td>
          <td class=”td”><input type=”text” value=”0” name=”f0”></td>
          <td class=”td”><input type=”text” value=”0” name=”t0”></td>
        </tr>
        <tr class=”tr”>
          <td class=”td”><input type=”text” value=”0” name=”v1”></td>
          <td class=”td”><input type=”text” value=”0” name=”f1”></td>
          <td class=”td”><input type=”text” value=”0” name=”t1”></td>
        </tr>
        <tr class=”tr”>
          <td class=”td”><input type=”text” value=”0” name=”v2”></td>
          <td class=”td”><input type=”text” value=”0” name=”f2”></td>
          <td class=”td”><input type=”text” value=”0” name=”t2”></td>
        </tr>
        <tr class=”tr”>
          <td class=”td”><input type=”text” value=”0” name=”v3”></td>
          <td class=”td”><input type=”text” value=”0” name=”f3”></td>
          <td class=”td”><input type=”text” value=”0” name=”t3”></td>
        </tr>
        <tr class=”tr”>
          <td class=”td”><input type=”text” value=”0” name=”v4”></td>
          <td class=”td”><input type=”text” value=”0” name=”f4”></td>
          <td class=”td”><input type=”text” value=”0” name=”t4”></td>
        </tr>
        <tr class=”tr”>
          <td class=”td”><input type=”text” value=”0” name=”v5”></td>
          <td class=”td”><input type=”text” value=”0” name=”f5”></td>
          <td class=”td”><input type=”text” value=”0” name=”t5”></td>
        </tr>
        <tr class=”tr”>
          <td class=”td”><input type=”text” value=”0” name=”v6”></td>
```

```
          <td class="td"><input type="text" value="0" name="f6"></td>
          <td class="td"><input type="text" value="0" name="t6"></td>
        </tr>
        <tr class="tr">
          <td class="td"><input type="text" value="0" name="v7"></td>
          <td class="td"><input type="text" value="0" name="f7"></td>
          <td class="td"><input type="text" value="0" name="t7"></td>
        </tr>
        <tr class="tr">
          <td class="td"><input type="text" value="0" name="v8"></td>
          <td class="td"><input type="text" value="0" name="f8"></td>
          <td class="td"><input type="text" value="0" name="t8"></td>
        </tr>
        <tr class="tr">
          <td class="td"><input type="text" value="0" name="v9"></td>
          <td class="td"><input type="text" value="0" name="f9"></td>
          <td class="td"><input type="text" value="0" name="t9"></td>
        </tr>
      </table>
      </td>
      </tr>
     <tr>
     <td>
       <div>
       <input class="button" type="submit" value="Go" />
       <input class="button" type="submit" value="Stop" />
        </div>
      </td>
    </tr>
    </table>
    </form>

    </td>
   <td id="reset">

   <form action="/cgi-bin/reset.py" method "post">
   <div>
     <input class="button" type="submit" value="Reset" />
   </div>
   </form>

   </td>
   </tr>
   </table>
<!-- end content -->
</div>

<div id="footer">
  <br>
  <br>
</div>

<!-- Make a button which allows you to "git␣pull" from the web interface -->
<!--
```

```
<form action="/cgi-bin/git_update.bash" method "post">
<div>
  <input type="submit" value="git_update" />
</div>
</form>
-->

</body></html>
```

2.3 Code Listing - CGI Scripts

Listing 2: Update Website Script

```
#! /usr/bin/python2.7

import cgi
import cgitb
import update_voltage_frequency

# create instance of field storage to get values
parameters = cgi.FieldStorage()

# get the data from the fields
#update_voltage = parameters.getvalue('update_voltage')
#update_frequency = parameters.getvalue('update_frequency')

voltage_value = parameters.getvalue('voltage')
frequency_value = parameters.getvalue('frequency')

# Update the frequency and voltage
update_voltage_frequency.update_voltage(voltage_value)
update_voltage_frequency.update_frequency(frequency_value)

# This script is designed to count as a test
print "Content-type:text/html\r\n\r\n"
print "<html>"
print "<head>"
print "<title>CGI-Update-Procedure</title>"
print "<head>"
print "<body>"
print "<h1>Current Values:"
print "<h3>Voltage: %s V" % (voltage_value)
print "<h3>Frequency: %s Khz" % (frequency_value)
print "</body>"
print "</html>"

# Reprint index.html so that the user may change the voltage again
with open('/home/pi/senior_design/www/index.html', 'r') as file:
  line = file.readline()

  while line:
    #print "\""+line+"\""
    print line
```

```
    line = file.readline()
```

Listing 3: Update Voltage and Frequency Script

```
#! /usr/bin/python2.7

import spidev
import minigen
import voltage_regulator
import cPickle as pickle
import pga
import subprocess

#Designed to communicate with a Minigen connected to the GPIO pins
spi = spidev.SpiDev()

# Test function
def main():
  print 'running in test mode'

  # Test setting the frequency using spi
  #update_frequency(100)

  # Test setting the voltage using I2c
  update_voltage(5)

# define variables
#minigen_pickle_file = "/tmp/mini_pickle"
#digital_pot_pickle_file = "/tmp/pot_pickle"

# Update the voltage level to the specified value.
# This is not the voltage output by the minigen,
# Instead it is the voltage output by the circuit as a whole.
def update_voltage(voltage):
  # use pga script to set values of pga
  #p_amp = pga.pga()
  #p_amp.setGain(int(voltage))

  # call a script that will set the pga values
  #subprocess.call("pwd", shell=True)
  subprocess.call("./pga_calling_script.bash " + voltage, shell=True)

  #print 'voltage updated'

  # make an instance of the voltage_regulator class to handle the connection
  #vr = voltage_regulator.voltage_regulator()
  #vr = get_pickle_digital_pot()

  # ask vr to set the voltage to the given value
#  vr.set_voltage(voltage)

  # update pickled information
  ###set_pickle_digital_pot(vr)

  # preform cleanup actions
```

```
  #vr.close_regulator()

# Update the frequency to the specified value. Values are given in Khz.
def update_frequency(frequency):
  #print 'frequency␣updated'

  # make an instance of the minigen class to handle the connection
  m = minigen.minigen()
 # m = get_pickle_minigen()

  # ask the minigen to set the new frequency
  m.setFrequency(float(frequency)*1000)

  # update pickled information
  ###set_pickle_minigen(m)

  #close the conection
  m.close()

# attempt to grab pickeled information about minigen
# if no pickle is found, create new minigen object
def get_pickle_minigen():
  try:
    m = pickle.load( open( minigen_pickle_file , "rb" ) )
    #print "pickle␣loaded␣successfully"
  except:
    m = minigen.minigen()
    #print "new␣object␣created"

  return m

# attempt to grab pickeled information about dititial pot
# if no pickle is found, create new minigen object
def get_pickle_digital_pot():
  try:
    vr = pickle.load( open( digital_pot_pickle_file ,  "rb" ) )
  except:
    vr = voltage_regulator.voltage_regulator()

  return vr

# set pickeled information about minigen
def set_pickle_minigen(m):
  pickle.dump( m, open(minigen_pickle_file , "wb" ) )

# set pickeled information about digital pot
def set_pickle_digital_pot(vr):
  pickle.dump( vr, open(digital_pot_pickle_file , "wb" ) )

if(__name__ == "__main__"):
  main()
```

Listing 4: Voltage Control Script

```
import digital_pot

class voltageControl:


  def __init__(self):

    self. digitalPot = digital_pot.digital_pot()

    #Upper and lower Bound of DigitalPot
    self.upperResistance = 9870.0
    self.lowerResistance = 164.0
    self.maxStep = 128

    #Amplifier Circuit
    self.rf = 10000 #6575
    self.maxVoltage = 20
    self.minVoltage = 0
    self.inputVoltage = 1




  def close(self):
    self.digitalPot.close()


  def setVoltage(self, voltage):
    step = int(((((self.rf*self.inputVoltage)/voltage)-self.lowerResistance) * self.maxStep)/(self.upperResistance - self.lowerResistance))
    print step
    if( step > self.maxStep):
        step = self.maxStep
    self.digitalPot.setStep(step)




def main():
#   print 'running in test mode'
  vC = voltageControl()
  vC.setVoltage(10)


if(__name__ == "__main__"):
  main()
```

Listing 5: Voltage Control Script

```
#! /usr/bin/python2.7

import digital_pot

# This script adjusts the gpio pins to communicate
# with the variable resistors, adjusting them to produce the voltage we want
```

```
# ment to abstract away the communication with variable resistors
#
# uses the digital_pot class to communicate with the digital pots
class voltage_regulator:
  # initialize the connection
  def __init__(self):
    self.op0_r0 = 0
    self.op0_r1 = 1
    #self.op1_r0 = 0
    #self.op1_r1 = 1

  # variable resistors will use spi as well
  def set_voltage(self, voltage):
    # compute the resistor values
    self.compute_resistor_values(voltage)

    # set the variable resistor value
    #TODO use the digital_pot class

  # given the desired voltage, compute the necessary resistor values
  #
  # Vout = Vin(1 + (opx_r1/opx_r0))
  #
  # The above transfter function is for a single op amp
  # Vin is 1vpp from signal generator
  def compute_resistor_values(self, voltage):
    # fix r1 so there is only one variable
    self.op0_r1 = 1000

    # compute r1
    # function solved for r1
    #
    # opx_r0 = (Vin(opx_r1))/(Vout - Vin)
    self.op0_r0 = self.op0_r1/(voltage -1)

  # print out the calculated resistor values
  def print_resistor_values(self):
    print "resistor op0_r0: " + str(self.op0_r0)
    print "resistor op0_r1: " + str(self.op0_r1)

  # preform any cleanup actions (most likly closing i2c connection)
  def close_regulator(self):
    pass


# Test function
def main():
  print 'running in test mode'
  vr = voltage_regulator()

  vr.set_voltage(10)
  vr.print_resistor_values()

if(__name__ == "__main__"):
```

```
  main()
```

Listing 6: Minigen Control Script

```
#! /usr/bin/python2.7

import spidev
import time
import sys
# Designed to provide some of the functionality from SparkFun_MiniGen.cpp
# designed so that you only need to use set_frequency to set the frequency
class minigen:
  # initialize the connection with the minigen
  def __init__(self):
    self.spi = spidev.SpiDev()

    # open(bus, device)
    self.spi.open(0, 0)

    # minigen is driven at 40Mhz
    #self.spi.max_speed_hz = 15000000

    self.controlReg = [False]* 16
    self.controlReg[16-13] = True

    self.freqReg0 = [False]*32
    self.freqReg1 = [False]*32

    self.freqReg0[31-30] = True
    self.freqReg0[31-14] = True

    self.freqReg1[31-31] = True
    self.freqReg1[31-15] = True

    self.fudgeFactor = 1

##########      Control Register 16Bits
#   Bit Number    Name     Function
#   D15           Addr1    Always 0          D15 and D14 is the address of the control register
#   D14           Addr0    Always 0
#   D13           B28      When 1: allows a complete word to be loaded into a freq reg with two consecutive write. First contains 14 LSB, second contains
#                          14 MSB. (First two bits is freq reg addr)  Consecutive writes to the same freq register is not allowed, you must alternate.
#                          When 0: Configures the 28bit freq reg is act as two 14 bit regs. One contains 14 LSB, the other 14MSB. This allows for coarse, or fine
#                          grain tuning. HLB defines which to change.
#
#
#
#   D12           HLB      This allows the user to continiously load the MSB or LSB of a freq reg. Ignoring ther other 14 bits. When B28 = 1, this is ignored.
```

```
#                              When 1: Allows write to 14 MSB
#                              When 0: Allows write to 14 LSB
#
#
#
#    D11           FSEL        Selects either freq0 or freq1
#    D10           PSEL        Selects either phase0 or phase1
#    D09         reserved        0
#    D08          RESET      When 1: resets internal regs to 0. When 0: disables the reset function.
#    D07          Sleep1       Enables or disables MCLK
#    D06          Sleep12      Powers down on chip DAC
#    D05          OPBITEN      0
#    D04                       0
#    D03
#    D02           TODO
#    D01
#    D00

  def chooseFreq0(self):
    self.controlReg[15-11] = False
  def chooseFreq1(self):
    self.controlReg[15-11] = True
  def enableB28(self):
    self.controlReg[15-13] = True
  def disableB28(self):
    self.controlReg[15-13] = False
  def enableHLB(self):
    self.controlReg[15-12] = True
  def disableHLB(self):
    self.controlReg[15-12] = False

  def sendControlReg(self):
    controlRegNum = self.boolListToInteger(self.controlReg)
    self.spi.xfer([controlRegNum >> 8, controlRegNum & 0xFF])

  def getControlReg(self):
    return (self.boolListToInteger(self.controlReg))


  #Frequency Functions
  #
  #Frequency Registers are set up as one 1 32bit register with [31-30] and [15-14] defined to be the address
  #

  def setFreqRegister(self,freqReg, isMSB, num):
    if(num >  0x3FFF):
      return -1
    bitString = bin(num)[2:][::-1]
    if(isMSB == 1):
      x = 15
    else:
      x = 31
```

```
    for i in bitString:
      if int(i) == 1:
        if(freqReg == 0):
          self.freqReg0[x] = True
        else:
          self.freqReg1[x] = True
      else:
        if(freqReg == 0):
          self.freqReg0[x] = False
        else:
          self.freqReg1[x] = False
      x= x - 1
    return 1


  def setFreq0MSB(self,num):
    return self.setFreqRegister(0,1,num)

  def setFreq0LSB(self,num):
    return self.setFreqRegister(0,0,num)

  def setFreq1MSB(self,num):
    return self.setFreqRegister(1,1,num)

  def setFreq1LSB(self,num):
    return self.setFreqRegister(1,0,num)

  def setEntireFreqReg0(self,num):
    actualValue = self.calculateFrequency(num)
    if(actualValue > 0x3FFFFFFF):
      return -1
    self.setFreq0LSB(actualValue & 0x3FFF)
    self.setFreq0MSB(actualValue >> 14)
    return 1

  def setEntireFreqReg1(self,num):
    actualValue = self.calculateFrequency(num)
    if(actualValue > 0x3FFFFFFF):
      return -1
    self.setFreq1LSB(actualValue & 0x3FFF)
    self.setFreq1MSB(actualValue >> 14)
    return 1


  def getFreqReg0(self):
    return (self.boolListToInteger(self.freqReg0))
  def getFreqReg1(self):
    return (self.boolListToInteger(self.freqReg1))

  def sendFreqReg0MSB(self):
    sendFreqRegNum = self.boolListToInteger(self.freqReg0)
    self.spi.xfer([sendFreqRegNum >> 24, (sendFreqRegNum >> 16) & 0xFF])

  def sendFreqReg0LSB(self):
```

```
    sendFreqRegNum = self.boolListToInteger(self.freqReg0)
    self.spi.xfer([(sendFreqRegNum >> 8) & 0xFF, (sendFreqRegNum) & 0xFF])

  def sendFreqReg1MSB(self):
    sendFreqRegNum = self.boolListToInteger(self.freqReg1)
    self.spi.xfer([sendFreqRegNum >> 24, (sendFreqRegNum >> 16) & 0xFF])

  def sendFreqReg1LSB(self):
    sendFreqRegNum = self.boolListToInteger(self.freqReg1)
    self.spi.xfer([(sendFreqRegNum >> 8) & 0xFF, (sendFreqRegNum) & 0xFF])

  def setFrequency(self, freq):
    if(freq < 10000):
      return -1
    self.enableB28()
    self.chooseFreq1()
    self.sendControlReg()
    self.setEntireFreqReg0(freq)
    self.sendFreqReg0MSB()
    self.sendFreqReg0LSB()
    self.chooseFreq0()
    self.sendControlReg()
    return 1

  def setFrequency1(self, freq):
    self.disableB28()
    self.enableHLB()
    self.chooseFreq1()
    self.sendControlReg()

    calculatedValue = self.calculateFrequency(freq)
    if(calculatedValue > 0x3FFF):
      return -1

    MSB = (calculatedValue >> 14) & 0x3FFF
    self.setFreqRegister(0, 1, MSB)
    self.sendFreqReg0MSB()
    self.disableHLB()
    self.sendControlReg()
    LSB = calculatedValue & 0x3FFF
    self.setFreqRegister(0,0,LSB)
    self.sendFreqReg0LSB()

    self.chooseFreq0()
    self.sendControlReg()

  def calculateFrequency(self, num):
    #print "Calculated Value: " + str(int(num/(.0596)))*self.fudgeFactor
    return int(num/(.0596))*self.fudgeFactor

  def close(self):
    self.spi.close()

  #Converts a boolean array to a number
```

```
  def boolListToInteger(self, lst):
    return int(''.join(['1' if x else '0' for x in lst]),2)

# Test function
def main():
  print 'running in test mode'
  m = minigen()
  m.setFrequency(float(sys.argv[1]))
  print "Frequency Register: " + bin(m.getFreqReg0())
  m.close()


if(__name__ == "__main__"):
  main()
```

Listing 7: PGA Control Script

```
#! /usr/bin/python2.7

import RPi.GPIO as GPIO
import time
import os
import sys

class pga:

  def __init__(self):
    #Switch user
    #sudoPassword = 'root'
    #command = 'sudo -i'
    #p = os.system('echo %s|sudo -S %s' % (sudoPassword, command))

    #Default Gx pins
    self.pinGx = [4, 17, 27]

    #6910-3
    #self.gainList = [0, -1, -2, -3, -4, -5, -6, -7]

    #6910-2
    #self.gaintList = [0, -1, -2, -4, -8, -16, -32, -64]

    #6910-3
    self.gainList = [0, -1, -2, -5, -10, -20, -50, -100]


    GPIO.setmode(GPIO.BCM)
    GPIO.setwarnings(False)
    self.updatePins()


  def updatePins(self):
    GPIO.cleanup()
    for pos in range(0,3):
      GPIO.setup(self.pinGx[pos], GPIO.OUT)

```

```
  def setPinG0(self, pin):
    self.pinGx[0] = pin
    self.updatePins()

  def setPinG1(self, pin):
    self.pinGx[1]= pin
    self.updatePins()

  def setPinG2(self, pin):
    self.pinGx[2] = pin
    self.updatePins()

  def setGainList(self, list):
    if(len(list) == 8):
      self.gainList = list
      return True
    return False

  def getGainList(self):
    return self.gainList

  def printGainList(self):
    print self.gainList

  # Only Allows for gains set in the given list
  def setGain(self, gain):
    try:
      binNum = '{:03b}'.format(self.gainList.index(gain))
      #print binNum
      binNum = binNum[::-1]
      for pos in range(0, 3):
        if int(binNum[pos]) == 1:
          GPIO.output(self.pinGx[pos], 1)
        else:
          GPIO.output(self.pinGx[pos], 0)
    except ValueError:
      print "Gain not Found"


def main():
  print "Running PGA test"
  p = pga()
  p.setGain(int(sys.argv[1]))


 #self.gainList = [0, -1, -2, -5, -10, -20, -50, -100]
if(__name__ == "__main__"):
  main()
```

Listing 8: Reset Script

```
#! /usr/bin/python2.7

# TODO the purpose of this script is to reset the minigen to a default state

```

```python

# This script is designed to count as a test
print "Content-type:text/html\r\n\r\n"
print "<html>"
print "<head>"
print "<title>Reset-Procedure</title>"
print "<head>"
print "<body>"
print "</body>"
print "</html>"

# reprint index.html
with open('/home/pi/senior_design/www/index.html', 'r') as file:
  line = file.readline()

  while line:
    #print "\""+line+"\""
    print line
    line = file.readline()
```