

Frequency response measurement system for at- home laboratory work

FINAL REPORT

sdmay21-49

Client: Gary Tuttle

Advisers: Gary Tuttle

Team Members: David Gorzney, Rohan Gijare, Samuel Ferguson,
George Youngwirth, Kathryn Blesi, Tabitha Kiiru

Team Email: sdmay21-49@iastate.edu

Revised April 25, 2021

Table of Contents

1 Introduction	3
1.1 Acknowledgement	3
1.2 Problem and Project Statement	3
1.3 Operational Environment	3
1.4 Requirements	3
1.5 Intended Users and Uses	3
1.6 Assumptions and Limitations	4
1.7 Expected End Product and Deliverables	4
1.8 Development Standards & Practices Used	4
1.9 Applicable Courses from Iowa State University Curriculum	4
1.10 New Skills/Knowledge Acquired That Was Not Taught in Courses	4
2 Design	5
2.1 Similar Projects And Literature	5
2.2 Development Process	5
2.3 Design Schematic & PCB Prototype	5
2.4 Design Evolution	6
3 Testing	7
3.1 Unit Testing	7
3.2 Interface Testing	8
3.3 Acceptance Testing	9
4 Implementation	10
5 Closing Material	11
5.1 Conclusion	11
5.2 References	11
6 Appendices	12
6.1 Operation Manual	12
6.2 Alternative Designs	15
6.3 Code	15

List of figures/tables/symbols/definitions (This should be the similar to the project plan)

- 2.3 Figure 2.3.1: Frequency Generator Module Schematic
- 2.4 Figure 2.4.1: Device Block Diagram
- 3.1 Figure 3.1.1 Output waveform of the frequency generator using a simple RC filter circuit
- 3.2 Figure 3.2.1 Frequency Response Analysis Application GUI

1 Introduction

1.1 ACKNOWLEDGEMENT

Great acknowledgements are given to Professor Gary Tuttle of the Iowa State University Electrical Engineering department for his insight and knowledge to the topics revolving around the project specifications

1.2 PROBLEM AND PROJECT STATEMENT

- This project is a take home frequency generator. Throughout the COVID19 pandemic, students were left without a lab bench while taking heavy lab classes. Iowa State University figured out how to send the students take home exploration boards. The problem though, was that these boards cost way too much money. On top of that the boards were sold out as many other programs attempted to buy the same boards. The take home frequency generator project has the goal of giving students the hardware to measure the frequency response of their circuits for a price that is reasonable.
- The project is to consist of a printed circuit board and other components.
- There are going to be two probes for measurement along with the frequency input wire and the ground.
- The assembly will have a way to store values through software directly into the computer.

1.3 OPERATIONAL ENVIRONMENT

This product is designed to be used in an at home lab environment. The product is designed to be able to be put into a backpack and carried around so it will not be too fragile, though it is not rugged enough to be dropped or crushed. Being as it is a piece of electrical lab equipment, it is not designed to be wet or left out in the elements.

1.4 REQUIREMENTS

Functional

- A low cost component system that creates, measures and records the frequency responses and can be purchased to be operated at-home.
- The final product should have the following components incorporated into it :
 - a sweepable AC source with frequency ranging from 10 Hz to 1 MHz
 - a chip to measure RMS voltages
 - a microcontroller to automate the sweep and measurement
 - a memory module for storing the measured results
 - an interface for that can be used with a computer to control measurements and review measurements.

Non-functional

- Portable
- Easy to use
- Should cost less than \$25

1.5 INTENDED USERS AND USES

Our end product will be used by college students enrolled in EE201, EE230, and EE333 at Iowa State. It will be used as a way to complete laboratory assignments if a student is not able to physically be in the lab.

1.6 ASSUMPTIONS AND LIMITATIONS

Assumptions:

User will have -

- Windows machine to run Frequency Response Analyzer App
- An Arduino microcontroller with ADC capabilities.
- Breadboard and wires to connect terminals of the Arduino to the board and test circuit components.

Limitations:

- Should be completed by the end of spring semester 2021
- The Frequency range of the function generator will be between 10 Hz to 1 MHz
- Should be pre-soldered so that students with no soldering skills can use the device easily.
- Manufacturing cost should be \$100 or less

1.7 EXPECTED END PRODUCT AND DELIVERABLES

The project is a frequency response module that will be used by students for at home lab work. The PCB will be used to connect all of the various parts and will have circuit logic used to hold the system together. The probes will be used to send out the generated frequency, for ground, and for reading the information for the frequency response. The user manual details how to use the end product. This will be the last thing we make, once we know how the end product will work. This will be completed by April of 2021.

1.8 Development Standards & Practices Used

- The development of the prototype will be guided by the IEEE standards for technical reliability and soundness.
 - IEEE730: Unit testing
 - IEEE754: Measurements must be accurate to two decimal places
 - IEEE33-1927: Frequency specification must be accurate to 1Hz
- Solderless breadboards will be used as a construction base for the testing of all the components for the final product.

1.9 Applicable Courses from Iowa State University Curriculum

EE 201: Electric Circuits

EE 230: Electronic Circuits and Systems

EE 333: Electronic Systems and Design

1.10 New Skills/Knowledge acquired that was not taught in courses

PCB design (KiCAD)

Python programming

Arduino programming (C++)

Application specific MCU programming (AD9833 Function Generator)

Microchip unit testing (RMS to DC converter and Function Generator)

2 Design

2.1 Similar Products And Literature

In the market currently there is a board called the DAD (Diligent Analog Discovery) board. This board has many different features that relate to electronic circuit analysis. It has frequency response, phase response, and many more features. The difference between this product and the product we are trying to create is that the DAD board has many more bells and whistles which cause it to cost significantly more (\$399) and we planned to have the Frequency Response Device cost around \$25.

2.2 DEVELOPMENT PROCESS

We used the Waterfall methodology as our development process, which involves completing one task before moving on to the next. This allowed us to know where we were in the design process and the tasks we needed to complete. We first looked at the project's requirements and talked to our client about what he wanted the end product to look like. Using this information we created a general design of our device, adding small details as our client advised us, and eventually created a detailed design. Next step was programming the function generator, then performing unit testing on each component in order to determine that they work. We then moved to integration testing to test multiple units together. The last step of this methodology was system testing where we tested the system as a whole to make sure that the device meets all the design requirements.

2.3 DESIGN SCHEMATIC & PCB PROTOTYPE

The project included designing a schematic and designing a PCB layout of the model. Changes and alterations that occurred were reflected in the schematic and PCB design, and in the updated list of components. We created a schematic capture, associated footprint and defined the outline. However, we were unable to create a board layout due to time limitations and the fact that we were new to using KiCad.

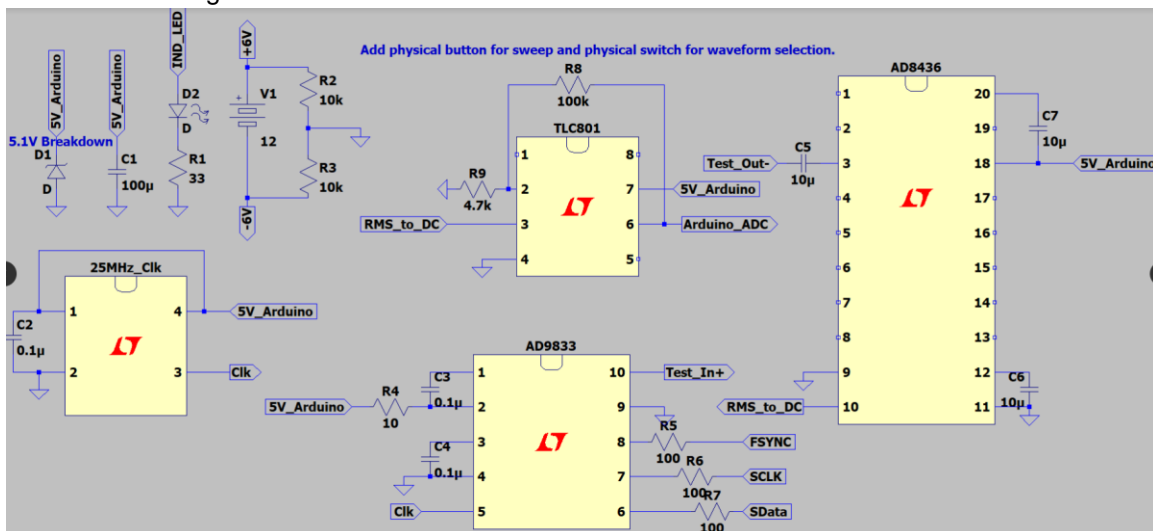


Figure 2.3.1: Frequency Generator Module Schematic

2.4 DESIGN EVOLUTION

For our original proposed design, we wanted the Frequency Response Device to have an embedded MCU that would control the function generator, sample the RMS to DC output and save the data points to a CSV. Since some of our group members already had experience with software development, we decided to use a PC to save and post process the data points. This helped reduce the development time of the device and overall cost of the device. Also having an application that can post-process the data and give time estimations of the analysis makes using the device much more convenient for the user. The one drawback of this design change is that the user is required to own an MCU such as an Arduino. However, students can check out Arduino's for free from ETG in Coover Hall.

We also wanted to give the user the ability to set specific frequencies using a potentiometer, but this was only a stretch goal to give the device more features and flexibility. The device was also supposed to have an LCD screen that would read the current frequency being output of the time remaining for the analysis, but this was replaced with the Frequency Response Analysis application.

We added $\pm 6V$ terminals so that the user can analyze active filter circuits instead of just passive filter circuits, but the 12V source must be a battery or a battery pack. The board also has a physical button to start the analysis and a switch to swap between sinusoidal and triangular waveforms. The final design of our PCB would be similar to the block diagram given below. We have all the mentioned features in our prototype breadboard design.

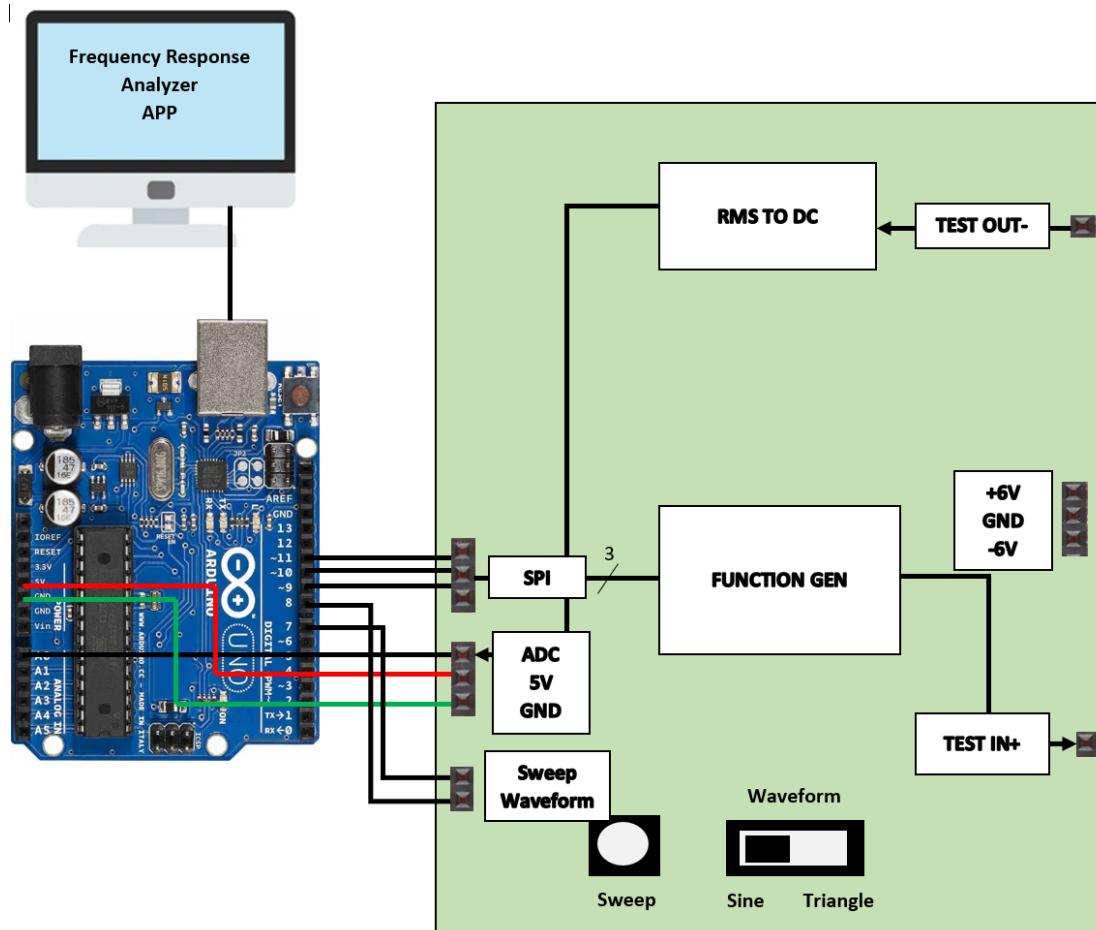


Figure 2.4.1: Device Block Diagram

3 Testing

3.1 UNIT TESTING

The unit testing of our project was broken down into three main groups: Testing for the function generator module, testing for the RMS to DC Converter module, and software testing for the User application and the Arduino code.

Function Generator Module -

To test the function generator module, we created an Arduino application that would program the function generator to do different waveforms at specified frequencies. This testing allowed us to learn what all the correct bit placements were to correctly program the AD9833 function generator. Once the AD9833 programming was understood, we then were able to complete an Arduino application. This application programs the function generator to sweep frequencies with the given waveform from 10Hz to 1MHz.

RMS to DC Converter Module -

We used a lab function generator to generate sinusoids at different V_{rms} values and frequencies and analyzed the given outputs for the RMS to DC converter (AD8436). This testing helped us get an idea of the accuracy of the RMS to DC converter and what the amplification of the function generator and the DC output would need to be.

Software Applications -

For testing of the Arduino and Frequency Response Analyzer application, we focused on the Arduino to PC communication. The serial driver in the Frequency Response Analyzer application needed to be programmed to have a timeout of 5 seconds. The serial driver also needed to be correctly unbound when the application was no longer using the comport. The Arduino would send the sampled magnitudes via comport to the PC. The first byte of data from each sampled value would begin with xC0. This would be used as a flag for the Frequency Response Analyzer application to ensure that the sampled comport messages were from the Arduino.

System.

Finally for system testing, we used the device on a couple of filter circuits with their corner frequencies calculated and compared the output waveforms of the Frequency Response Analyzer. For example, we tested the device with an RC filter circuit (1kohm & 10nF) with corner frequency of 15.91kHz and received this plot as the output which correctly corresponds to the expected frequency response.

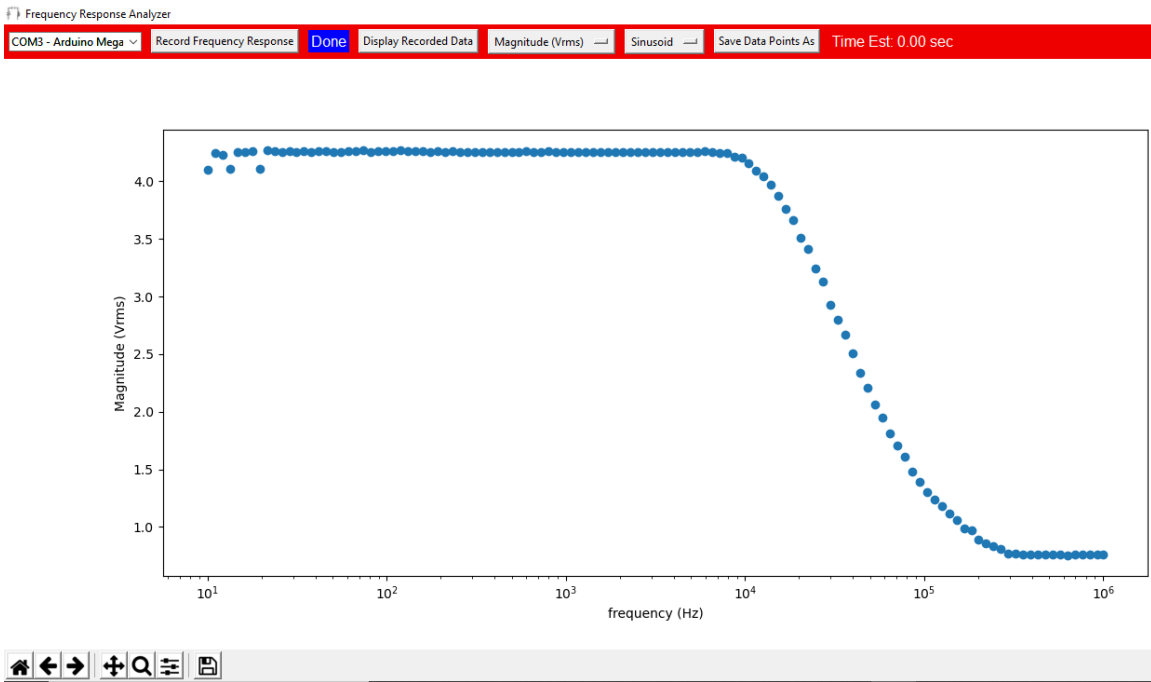


Figure 3.1.1 Output waveform of the frequency generator using a simple RC filter circuit

3.2 INTERFACE TESTING

For interface testing, we tested our Frequency Response Analyzer applications' user interface. For the comport selection window, the user can select any available comport. If the comport is unavailable, then the blue "Done" indication label will display "COM Unavailable". This would be the case if the comport was unplugged or being used by another application. The comport can also timeout if the user didn't hit the sweep button in time after pressing the "Record Frequency Response" button or the comport they selected was the improper port. In this case, the indicator label will then display "Timed Out". When the user presses the "Display Recorded Data" button, the application displays the most recent completed frequency response recording. This makes it so that the application doesn't fail if they have a failed attempt and try to display the data. The application will display "Done" in the indicator label to tell the user that the application is done sampling. There is also a "Time Est." label to indicate the amount of time remaining for the frequency response analysis.

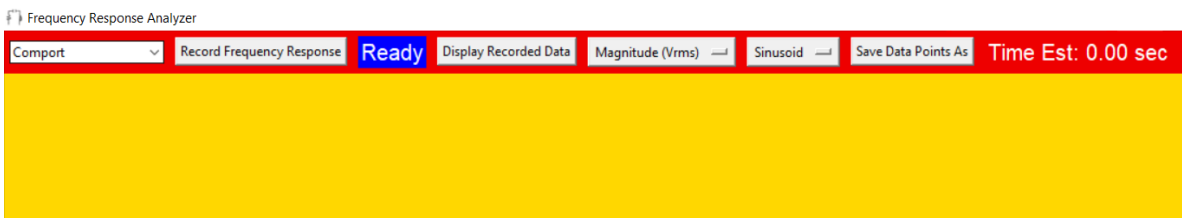


Figure 3.2.1 Frequency Response Analysis Application GUI

3.3 ACCEPTANCE TESTING

To demonstrate that the design requirements are being met, we compared the results obtained from testing to the expected and desired results. The device sweeps from 10Hz to 1MHz as given in the design requirements. We used an oscilloscope to probe the output of the device without a filter circuit and ran the device. The device accurately swept between the programmed frequencies from 10Hz to 1MHz. We then added filter circuits to the device, probed the filter output with the oscilloscope, and compared the oscilloscope magnitudes with the recorded magnitudes by the Frequency Response Analyzer. The Oscilloscope and the device had the same magnitudes and frequency response.

4 Implementation

For the implementation of the device, we started by designing the function generator module using the AD9833 IC. Before we designed the module, we did extensive research through the data sheets provided. The AD9833 IC was a digital application specific IC and was quite complicated to learn how to program with its proprietary programming message formats. We learned how to use the SPI serial interface, different bit placements, and data conversions for the IC. We used a testing schematic given to us on the datasheet for the function generator and tried programming it using an Arduino. Programming the AD9833 was the most time consuming portion of the project as it was hard to learn the bit placement of the programming messages with few examples online. Once we figured out how to program it to create a specific frequency and waveform, we needed to figure out how to constantly reprogram it to sweep through a range of frequencies. With some more research and testing, we were able to figure out how to reset the function generator and reprogram it for a different frequency quickly and easily.

Once we completed the function generator module, we started to work on a prototype Python application that would take a CSV of recorded data points and display them on a plot. The prototype mostly consisted of the graphical user interface (GUI) and did not include the time estimation indicator or the status indicator. Once the GUI was designed for the application, we created an Arduino application that would serialize ADC values of a given voltage and send it to the PC. For this portion of the design we added a flag byte to each serialized message of 0xC0. This flag would indicate that the message was correct and complete. We then programmed the Python application to read the serial port and get the correct serial messages. The messages were then calculated into actual voltage values based on the ADC range of 5V.

Now that the Python application could sample the magnitudes of the Frequency Response Analyzer device, we could start to design the RMS to DC converter module. We started the design by creating a sample test circuit from the AD8436 datasheet. Using a lab function generator, we tested the module with different magnitudes and frequencies to see the corresponding DC output voltages. Luckily the device had little to no offset with a coupling capacitor to the RMS input of the IC. We decided that the DC output was accurate enough to amplify itself instead of the function generator. We decided this because in order to amplify the function generator we would need a negative supply and designing the circuit without a negative supply voltage would be much simpler and cost effective.

Once we had a working function generator module and RMS to DC conversion module, we then could work on finalizing the software. Magnitude sampling was added to the frequency sweeping Arduino code and the Python application's frequency range was matched to the frequency sweep of the function generator. Finally, all the modules for the device were complete and we began system testing and implementation.

With the final testing of the device, we changed the plot from being linear to logarithmic. This really helped eliminate the clutter of data points at the lower frequencies. We also added an optional 9-12V battery supply that would allow the user to have +-4.5V or +-6V available for testing active filters. We then tested some filter designs to make sure the device was recording frequency responses correctly. The design was a success, so we began working on a PCB using KiCAD. Unfortunately we were all new to using KiCAD and ran out of time to design and order the PCB so that we could have a finished device.

5 Closing Material

5.1 CONCLUSION

We were able to complete robust and effective software for this project and a full working schematic for the design that met nearly all of the requirements. We unfortunately did not have enough time to complete a PCB design and order the PCB for the finalized device. With the two extra weeks we have in a normal semester, we believe we would've been able to fully complete the project and have a finalized device. Overall, with the current software and design materials we have available, another engineer or student should easily be able to replicate the working design for themselves.

5.2 REFERENCES

List technical references and related work / market survey references. Do professional citation style (ex. IEEE).

Schwartz, Eric M. "Digilent Analog Discovery (DAD) Tutorial." *University of Florida*, 2015, mil.ufl.edu/3701/DAD/DAD_tutorial.pdf.

Additional Information for the components used in the project:

ATmega328P Microcontroller Unit Data Sheet -

https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf

AD9833 Frequency Generator IC Data Sheet -

<https://www.analog.com/media/en/technical-documentation/data-sheets/AD9833.pdf>

AD8436 RMS Voltage Measurement IC Data Sheet -

<https://www.analog.com/media/en/technical-documentation/data-sheets/AD8436.pdf>

6 Appendices

6.1 APPENDIX I - OPERATION MANUAL

Using the Frequency Response Analyzer App:


1. Download the “Frequency Response Analyzer.zip” file.
2. Extract the “Frequency Response Analyzer.zip” file.
3. Right click the “Frequency Response Analyzer.exe”, create a shortcut, and place the shortcut in your desired location.
 - a. The “Frequency Response Analyzer.exe” must remain in the folder, but the folder may be moved anywhere in your directory.
4. Run application by double clicking the shortcut or on the “Frequency Response Analyzer.exe” file.

Using the Frequency Response Analyzer Arduino code:

1. Open the Freq_Resp_Anal_Arduino.ino with the Arduino IDE.
2. Change the defined pin numbers to correspond with the current MCU or Arduino model.
 - a. DATA must be a Digital, PWM, or SDATA pin.
 - b. CLK must be a Digital, PWM, or SCLK pin.
 - c. FSYNC must be a Digital or PWM pin.
 - d. IND_LED must be a Digital or PWM pin.
 - e. TRI_SW must be a Digital or PWM pin.
 - f. ADC must be an ADC pin.
 - g. SWEEP_BUT must be a Digital or PWM pin.

```
#define DATA 9 // SPI Data pin number
#define CLK 10 // SPI Clock pin number
#define FSYNC 11 // SPI Load pin number (FSYNC in AD9833 usage)
#define IND_LED 22 //Outputs signal for sweep indicator light
#define TRI_SW 24 //Input of waveform switch for triangle
#define ADC A0 //ADC pin
#define SWEEP_BUT 26 //Input of button activation for f sweep
```

3. Upload the Freq_Resp_Anal_Arduino.ino to your Arduino.

 Freq_Resp_Anal_Arduino | Arduino 1.8.13 (Windows Store 1.8.42.0)

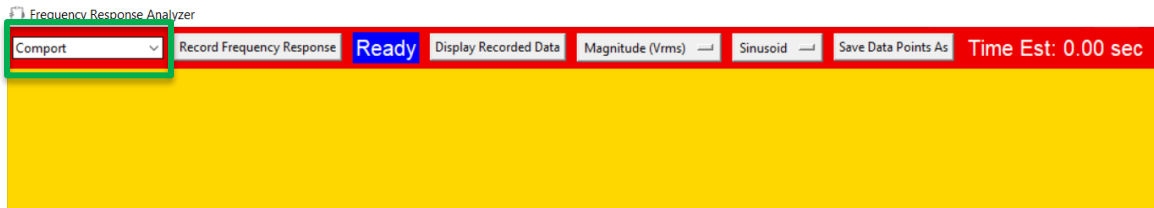
File Edit Sketch Tools Help



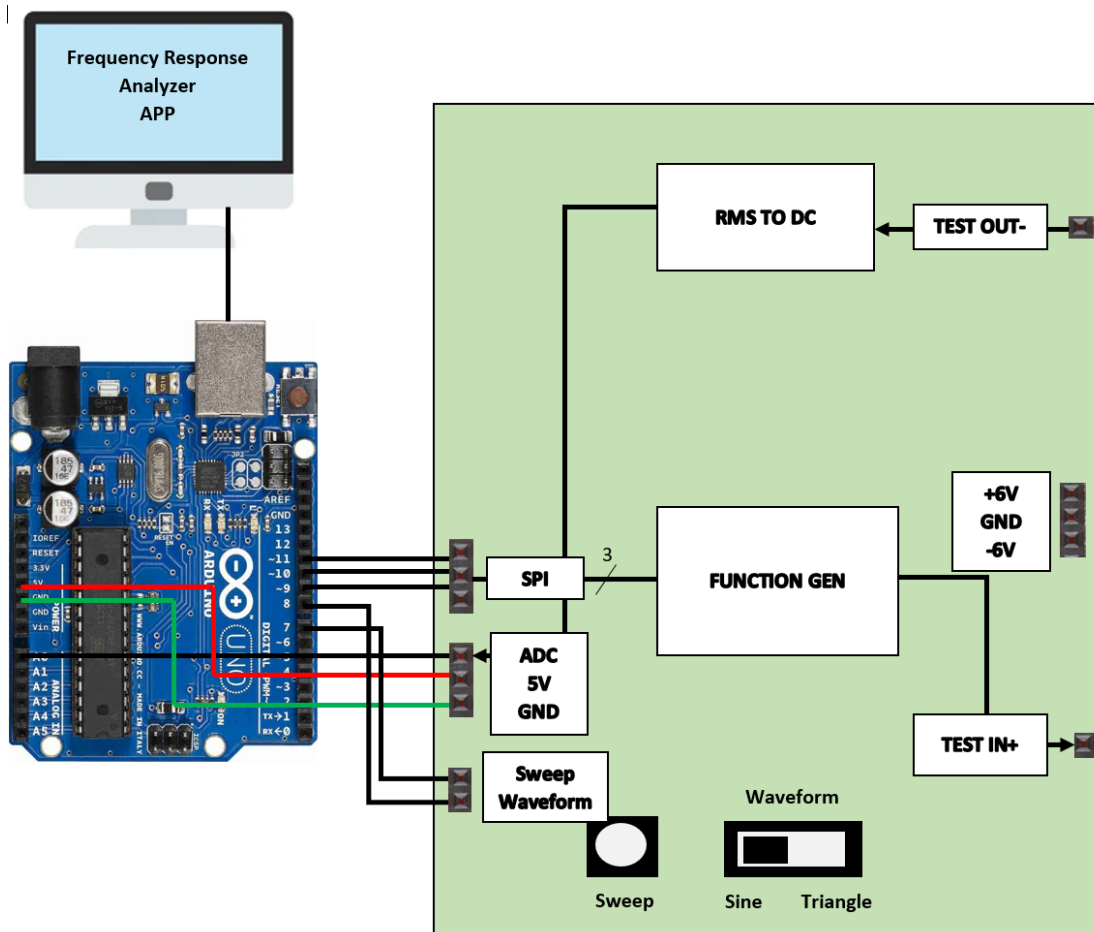
4. Connect Arduino pins to the corresponding schematic.

Running the Frequency Response Analyzer Device:

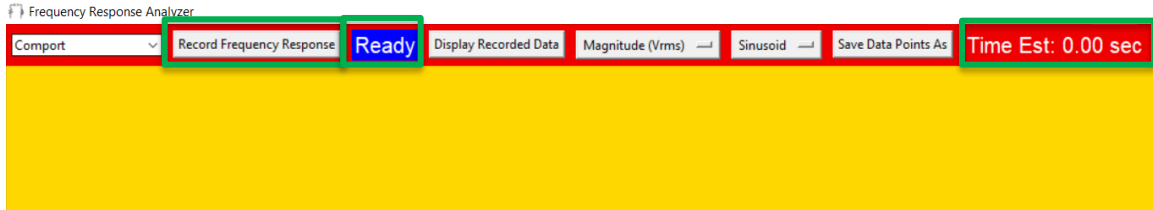
1. Run application by double clicking the shortcut or on the “Frequency Response Analyzer.exe” file.
2. Plug the Arduino into your PC.
3. Click the down arrow on the “Comport” dropdown box.



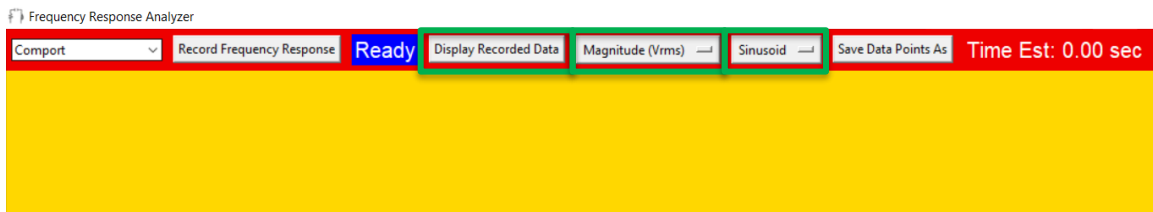
4. Select the Comport that corresponds to your Arduino.
5. Connect the ground of your circuit to the GND pin on the schematic or board.
6. Connect the Test In+ probe to the input of your filter circuit.
7. Connect the Test Out- to the output of your filter circuit.
8. Power the Vdd and Vss terminals of your active filter’s op-amps using the +6V and -6V terminals.



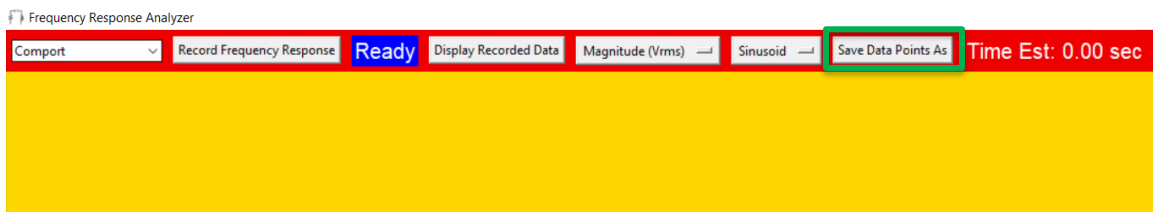
9. Move the waveform selection switch to the corresponding position for sinusoid or triangle waveforms.
10. Click “Record Frequency Response” button on the Frequency Response Analyzer App.
 - a. You have 5 seconds to click the Sweep Button on the Board or Circuit.
 - b. The indicator label should say “Reading” if data is being properly read by the application from the Arduino.
 - c. A time estimation should appear on the rightmost label in the application.
 - d. The indicator label will say “Done” once the data is done being recorded.



11. Press the “Display Recorded Data” button once the samples have been recorded.
 - a. This will display a graph corresponding to the data points taken.
 - b. You can select the format by selecting “Magnitude (Vrms)” or “Magnitude (dB)” and also selecting “Sinusoid” or “Triangle”.



12. You can save the data points into a csv by pressing the “Save Data Points As” button and selecting a destination.



13. To record another filter circuit, repeat steps 5-10.

6.2 APPENDIX II - ALTERNATIVE DESIGNS

Power Supply and Grounding Changed from Original Design

The original design of the Frequency Response Analyzer circuit did not have a supply for active filters. It was desired to use a wall-mounted 10V power supply to power the board, but this would create grounding issues if the board was connected to a PC that wasn't properly grounded. To avoid damages to the user's PC, we decided to use a 12V battery supply since the voltages are relative only to the terminals and not to common ground.

Data Recording Method Changed

Originally our client wanted the device to save the data points onto a physical SD card connected to the board and the board would have a MCU embedded into the device. We decided that we already had enough software experience to use a PC to do all the data recording and post processing. This would help reduce the size of the MCU programming files and reduce the cost of the design by requiring the user to already have their own MCU platform such as an Arduino.

Assembly Requirements Changed

Our client wanted to help reduce the cost of the final device by making the device easy to solder for novice electrical engineers. However, the function generator IC and the RMS to DC converter IC both only come in surface mount packages. This would require an adept engineer to solder. For this reason we planned the PCB to have IC sockets and have a technician from the ETG to solder the two surface mount ICs.

Inability to Meet Price Requirements for Device

The original price requirement for the Device was \$25, but the two primary ICs on the board itself cost \$25 (the function generator and the RMS to DC converter). To this end, we tried our best to reduce the cost of the overall design by requiring the user to have an Arduino with ADC capabilities and to supply batteries for the device.

6.3 APPENDIX III - CODE

ARDUINO CODE (C++)- `FREQ_RESP_ANAL_ARDUINO.INO`:


```

1  #include <math.h>
2  #include <stdint.h>
3  // Pins for SPI comm with the AD9833 IC
4  #define DATA 9 // SPI Data pin number
5  #define CLK 10 // SPI Clock pin number
6  #define FSYNC 11 // SPI Load pin number (FSYNC in AD9833 usage)
7  #define IND_LED 22 //Outputs signal for sweep indicator light
8  #define TRI_SW 24 //Input of waveform switch for triangle
9  #define ADC A0 //ADC pin
10 #define SWEEP_BUT 26 //Input of button activation for f sweep
11 #define MAX_FREQ 1000000 //1MHZ
12 #define DELAY 1500 //ms of delay between each freq change
13 #define FREQ_STEP_MULT 1.1
14 const long BAUDRATE = 9600;
15 int sweeping = 0;
16 unsigned long int output_frequency = 0;//Hz
17 int press_count = 0; //for debouncing button
18 int triangle = 0;
19
20 void write_to_func_gen(word data){
21     digitalWrite(CLK, LOW);
22     digitalWrite(CLK, HIGH);
23     digitalWrite(FSYNC, LOW);
24     for(byte i = 0; i < 16; i++){
25         if (data & 0x8000){
26             digitalWrite(DATA, HIGH);
27         }
28         else{
29             digitalWrite(DATA, LOW);
30         }
31         data = data << 1;
32         digitalWrite(CLK, HIGH);
33         digitalWrite(CLK, LOW);
34     }
35     digitalWrite(CLK, HIGH);
36     digitalWrite(FSYNC, HIGH);
37 }
38
39 void reset_func_gen(){
40     delay(100);
41     write_to_func_gen(0x0100);
42     delay(100);
43 }
44
45 unsigned long calc_freq_div(unsigned long int* freq){
46     //Serial.print("Entered freq: ");//For Testing
47     //Serial.println(*freq);//For Testing
48     //2^28/1MHz = 268.435
49     //2^28/25MHz = 10.737
50     double output_part1 = 10.737 * (*freq);
51     unsigned long output_part2 = round(output_part1);
52     //Serial.print("Div: ");//For Testing
53     //Serial.println(output_part1);//For Testing
54     unsigned long output = (unsigned long) output_part2;
55     return output;
56 }
57
58 void set_freq(unsigned long int* freq){
59     // 400 Hz Sine
60     unsigned long output_div = calc_freq_div(freq);
61     unsigned short freq1 = 0x0000;
62     unsigned short freq2 = 0x0000;
63     unsigned short wave = 0x2000;
64     freq1 = output_div | freq1;
65     freq1 = freq1 & 0x3fff;
66     freq1 = freq1 | 0x4000;
67     output_div = output_div >> 14;
68     freq2 = output_div | freq2;
69     freq2 = freq2 & 0x3fff;
70     freq2 = freq2 | 0x4000;
71     //Serial.println(freq1,16);//For Testing
72     //Serial.println(freq2,16);//For Testing
73     if(triangle == 1){
74         wave = wave | 0x0002;
75     }
76
77     write_to_func_gen(0x2100); //Ex: 0x2100
78     write_to_func_gen(freq1); //Ex: 0x636e
79     write_to_func_gen(freq2); //Ex: 0x4006
80     write_to_func_gen(0xc000); //Ex: 0xc000
81     write_to_func_gen(wave); //Ex: 0x2000
82 }
83

```

```

84 void sample_adc(void){
85     int adc_value = analogRead(ADC);
86     uint16_t adc_2byte = (uint16_t) adc_value;
87     adc_2byte = adc_2byte | 0xc000;
88     uint8_t adc_byte_2 = (adc_2byte >> 8);
89     uint8_t adc_byte_1 = (adc_2byte & 0x00ff);
90     Serial.write(adc_byte_2);
91     Serial.write(adc_byte_1);
92     //Serial.println(adc_value);//For Testing
93 }
94
95 void setup(void) {
96     Serial.begin(BAUDRATE);
97
98     pinMode(CLK, OUTPUT);
99     pinMode(DATA, OUTPUT);
100    pinMode(FSYNC, OUTPUT);
101    digitalWrite(CLK, HIGH);
102    digitalWrite(FSYNC, HIGH);
103    reset_func_gen();
104    pinMode(IND_LED, OUTPUT);
105    digitalWrite(IND_LED, LOW);
106    pinMode(TRI_SW, INPUT);
107    pinMode(SWEEP_BUT, INPUT);
108    pinMode(ADC, INPUT);
109 }
110
111 void loop(void) {
112     if(sweeping == 0){
113         int sweep_button_val = digitalRead(SWEEP_BUT);
114         if(sweep_button_val == 1){
115             press_count += 1;
116             if(press_count >= 3){
117                 sweeping = 1;
118                 digitalWrite(IND_LED, HIGH);
119                 press_count = 0;
120                 triangle = digitalRead(TRI_SW);
121                 output_frequency = 10;
122             }
123         }
124         else{
125             press_count = 0;
126         }
127     }
128     if(sweeping == 1){
129         if(output_frequency <= MAX_FREQ){
130             set_freq(&output_frequency);
131             delay(DELAY);//wait ms
132             sample_adc();
133             output_frequency = output_frequency * FREQ_STEP_MULT;//Frequency step const
134         }
135         else{
136             output_frequency = MAX_FREQ;
137             set_freq(&output_frequency);
138             delay(DELAY);//wait ms
139             sample_adc();
140             sweeping = 0;//Stopping sweep
141             output_frequency = 0;
142             reset_func_gen();
143             digitalWrite(IND_LED, LOW);
144         }
145     }
146 }
147 }

```

Python Code - Frequency Response Analyzer.py

```
1  # -*- coding: utf-8 -*-
2  """
3  Created on Fri Feb 19 16:33:00 2021
4
5  @author: Samuel Ferguson
6  """
7  import tkinter as tk
8  from tkinter import filedialog
9  import datetime as dt
10 from threading import Thread
11 from shutil import copyfile
12 import os
13 import time
14 import math
15 import matplotlib
16 matplotlib.use("TkAgg")
17 from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg, NavigationToolbar2Tk
18 from matplotlib.figure import Figure
19 import csv
20 import serial
21 import serial.tools.list_ports
22 try:
23     import ttk
24 except ImportError:
25     import tkinter.ttk as ttk
26
27
28 ###-----GUI Window-----###
29 root = tk.Tk()
30 root.title("Frequency Response Analyzer")
31 #root.geometry('700x700')
32 root.state("zoomed")
33 root.config(bg='gold')
34 root.iconbitmap(r'freq_resp_anal.ico')
35 ###-----GUI Frames-----###
36 topFrame = tk.Frame(root, bg='red2', height = 50)#Control Interface Frame
37 topFrame.pack(side = tk.TOP, fill = tk.X, expand = False)
38
39 ###-----Global Variables-----###
40 magnitude_format_options = ["Magnitude (Vrms)", "Magnitude (dB)"]
41 selected_magnitude_format = tk.StringVar()
42 selected_magnitude_format.set("Magnitude (Vrms)")
43 waveform_options = ["Sinusoid", "Triangle"]
44 selected_waveform = tk.StringVar()
45 selected_waveform.set("Sinusoid")
46 global plot_displayed
47 plot_displayed = False
48 global ser
49 ser = None
50 global connected_comport
51 connected_comport = None
52 global max_number_of_bytes
53 max_number_of_bytes = 0
54 global bytes_read
55 bytes_read = 0
56 global time_out_event
57 time_out_event = False
58 global num_freq_increments
59 num_freq_increments = 122
60 global max_freq
61 max_freq = 1000000 #Maximum frequency of func sweep
62 global input_vpk
63 input_vpk = 0.64 #Circuit input voltage without filter
64 global time_out
65 time_out = 5 #max number of seconds serial reader waits for bytes
66 """
```

```

66  ###-----Functionality-----###
67
68  ###-----Application Closing-----###
69  def on_close():
70      global ser
71      if ser != None:
72          if ser.is_open:
73              ser.close()
74      print("Closing program")
75      root.destroy()
76  ###-----Time Estimation-----###
77  def time_estimation():
78      global max_number_of_bytes
79      global bytes_read
80      global time_out_event
81      start_time = time.time()
82      bytes_read_til = bytes_read + 5
83      if max_number_of_bytes < 5 or bytes_read_til >= max_number_of_bytes:
84          return
85      while bytes_read < bytes_read_til and time_out_event == False:
86          pass
87      end_time = time.time()
88      time_difference = end_time - start_time
89      time_average = time_difference / 5
90      estimated_time = (max_number_of_bytes - bytes_read) * time_average
91      est_string = "Time Est: " + str(estimated_time)[:12] + " sec"
92      time_estimation_label.config(text=est_string)
93
94  ###-----Update Green Indicator-----###
95  def update_indicator_color_reading():
96      print('update color')
97      recording_indicator_label.config(text='Reading', bg = 'Green')
98      thread = Thread(target=record_magnitudes_from_com)
99      thread.start()
100  ###-----Record Magnitudes from COM Port-----###
101  def record_magnitudes_from_com():
102      comport = com_sel.get()
103      comport = comport[:5]
104
105      if len(comport) > 4:
106          comport = comport[:5]
107      try:
108          global ser
109          ser = serial.Serial(comport)
110      except serial.serialutil.SerialException:
111          print("Com port unavailable")
112          recording_indicator_label.config(text='COM Port Unavailable', bg = 'red')
113          ser = None
114          return
115      if ser != None:
116          print('Reading Serial Data')
117          print(comport)
118          global bytes_read
119          bytes_read = 0
120          global max_number_of_bytes
121          max_number_of_bytes = num_freq_increments
122          global time_out_event
123          time_out_event = False
124          ser.timeout = time_out
125          start_time = time.time()
126          cancel_time = start_time + time_out #timeout seconds
127          current_time_out = 0
128          magnitude_list = []
129          while bytes_read <= max_number_of_bytes and current_time_out <= cancel_time:
130              if bytes_read == 0:
131                  thread_2 = Thread(target=time_estimation)
132                  thread_2.start()
133              try:
134                  data = ser.read(2) #Read two bytes from serial in succession
135                  #print(data)
136              except serial.serialutil.SerialException:
137                  print("Com port unavailable")
138                  recording_indicator_label.config(text='COM Port Unavailable', bg = 'red')
139                  ser = None
140                  return
141              if not data:
142                  recording_indicator_label.config(text='Timed Out', bg = 'orange')
143                  print('Timed Out')
144                  time_out_event = True
145                  ser = None
146                  return

```

```

147         check_bits = int.from_bytes(data, byteorder="big", signed=False)
148         #Data check offset
149         adc_value = check_bits - 49152 #xC000 in Integer
150         #Convert adc value to voltage value
151         adc_value = (adc_value/1023) * 5 #10 bit adc with 5V in max
152         print(adc_value)
153         if adc_value > 0:
154             #add to list
155             magnitude_list.append(adc_value)
156             bytes_read += 1
157             start_time = time.time()
158             cancel_time = start_time + time_out
159             if bytes_read % 5 == 0 and bytes_read != 0:
160                 thread_2 = Thread(target= time_estimation)
161                 thread_2.start()
162         else:
163             current_time_out = time.time()
164     ser.close()
165     if current_time_out >= cancel_time:
166         recording_indicator_label.config(text='Timed Out', bg = 'orange')
167         print('Timed Out')
168         time_out_event = True
169     else:
170         recording_indicator_label.config(text='Done', bg = 'blue')
171         time_estimation_label.config(text="Time Est: 0.00 sec")
172         print('Done')
173         print(magnitude_list)
174         if os.path.exists('magnitude_data_points.csv') == False:
175             magnitude_csv = open('magnitude_data_points.csv', 'w', newline='')
176             magnitude_csv.close()
177         magnitude_csv = open('magnitude_data_points.csv', 'w', newline='')
178         magnitude_csv_write = csv.writer(magnitude_csv, delimiter=',')
179         #add date to CSV file
180         date_time = dt.datetime.now()
181         date_time_string = str(date_time.month) + '-' + str(date_time.day) + \
182             '-' + str(date_time.year) + ' ' + str(date_time.hour) + ':' + \
183             str(date_time.minute) + ':' + str(date_time.second)
184         print(date_time_string)
185         #TODO
186         #magnitude_csv_write.writerow(date_time_string)
187         #Add date to CSV
188         magnitude_csv_write.writerow(magnitude_list)
189         frequency_list = []
190         freq = 10
191         while freq < max_freq:
192             frequency_list.append(freq)
193             freq = freq * 1.1
194
195         frequency_list.append(max_freq)
196         magnitude_csv_write.writerow(frequency_list)
197         magnitude_csv.close()
198
199     ###-----Read Magnitudes from CSV-----###
200     def read_data_from_csv():
201         if os.path.exists('magnitude_data_points.csv') == False:
202             print('CSV does not exist')
203             return None
204         else:
205             magnitude_csv = open('magnitude_data_points.csv', 'r', newline='')
206             magnitude_csv_reader = csv.reader(magnitude_csv, delimiter=',')
207             data_list = []
208             row_list = list(magnitude_csv_reader)
209             data_list = row_list[0]
210             #print(data_list)
211             return data_list
212     ###-----Save Data As-----###
213     #save a copy of CSV to directory
214     def save_csv_as():
215         save_directory = filedialog.asksaveasfilename(title='Save Data Points', \
216             defaultextension=".csv", filetypes=(
217                 ("CSV", "*.CSV"), ("All Files", "*.*")))
218         if os.path.exists('magnitude_data_points.csv') == False:
219             print('CSV does not exist')
220             return None
221         else:
222             copyfile('magnitude_data_points.csv', save_directory)
223
224     ###

```

```

224  ###-----Create Plot-----###
225  def show_data_plot():
226      #Create plot outline
227      fig = Figure(figsize=(5,5), dpi=100)
228      fig_subplot = fig.add_subplot(111, xscale='log')
229
230      #Set plot axis labels
231      if selected_magnitude_format.get() == "Magnitude (Vrms)":
232          fig_subplot.set_ylabel("Magnitude (Vrms)")
233      else:
234          fig_subplot.set_ylabel("Magnitude (dB)")
235      fig_subplot.set_xlabel("frequency (Hz)")
236
237
238      #Add data points to plot and format
239      frequency_list = []
240      freq = 10
241      while freq < max_freq:
242          frequency_list.append(freq)
243          freq = freq * 1.1
244
245      frequency_list.append(max_freq)
246
247      #Read Magnitudes
248      magnitude_data_list = read_data_from_csv()
249      magnitude_list = []
250      magnitude_number = 0 #Counts number of read magnitudes
251
252      #Convert Vpk to Vrms based on waveform of func gen
253      if selected_waveform.get() == "Sinusoid":
254          input_voltage = input_vpk/math.sqrt(2)
255      elif selected_waveform.get() == "Triangle":
256          input_voltage = input_vpk/math.sqrt(3)
257      else:
258          input_voltage = input_vpk
259
260      number_of_samples = num_freq_increments
261      while magnitude_number < number_of_samples:
262          #Calculate dB and write magnitude value to data list for plot
263          if selected_magnitude_format.get() == "Magnitude (Vrms)":
264              magnitude_list.append(float(magnitude_data_list[magnitude_number]))
265          else:
266              db_magnitude = 20*math.log(float(magnitude_data_list[magnitude_number])/input_voltage,10)
267              magnitude_list.append(db_magnitude)
268          magnitude_number += 1
269      print(len(magnitude_list))
270      #Add Data Points to Plot
271      fig_subplot.scatter(frequency_list, magnitude_list)
272
273      #Draw Plot
274      global plot_displayed
275      if plot_displayed:
276          #draw new plot in place of old plot
277          widget_list = list(root.wininfo_children())
278          widget_list_length = len(widget_list)
279          widget_num = 1
280          if widget_list_length > 1:
281              while widget_num < widget_list_length:
282                  widget_list[widget_num].pack_forget()
283                  widget_num += 1
284          else:
285              root.geometry('655x600')
286              canvas = FigureCanvasTkAgg(fig, root)
287              canvas.draw()
288              canvas.get_tk_widget().pack(side=tk.BOTTOM, fill=tk.BOTH, expand=True)
289              toolbar = NavigationToolbar2Tk(canvas, root)
290              toolbar.update()
291              canvas._tkcanvas.pack(side=tk.TOP, fill=tk.BOTH, expand=True)
292              plot_displayed = True
293
294      ###-----Update COM Entry Combo Box-----###
295      #Search for available comports and display into combo box
296      def update_comports():
297          com_sel.config(values=serial.tools.list_ports.comports())
298

```

```

299  ###-----Control Interface-----###
300  com_sel = ttk.Combobox(topFrame, values=serial.tools.list_ports.comports(), \
301  postcommand=update_comports, takefocus="")
302  com_sel.grid(row=0, column=0, columnspan=1, padx=5, pady=5, sticky=tk.W)
303  com_sel.insert(0, "Comport")
304
305
306  record_data_button = tk.Button(topFrame, text = 'Record Frequency Response', \
307  command = update_indicator_color_reading)
308  record_data_button.grid(row=0, column=1, padx=5, pady=5, sticky = tk.W)
309
310
311  recording_indicator_label = tk.Label(topFrame, text='Ready', bg='blue', \
312  fg='white', font='bold')
313  recording_indicator_label.grid(row=0, column=2, padx=5, pady=5)
314
315
316  display_data_button = tk.Button(topFrame, text = 'Display Recorded Data', \
317  command=show_data_plot)
318  display_data_button.grid(row=0, column=3, padx=5, pady=5)
319
320
321  magnitude_format_sel = tk.OptionMenu(topFrame, selected_magnitude_format, \
322  *magnitude_format_options)
323  magnitude_format_sel["highlightthickness"]=0
324  magnitude_format_sel.grid(row=0, column=4, columnspan=1, padx=5, pady=5, sticky=tk.W)
325
326
327  waveform_sel = tk.OptionMenu(topFrame, selected_waveform, \
328  *waveform_options)
329  waveform_sel["highlightthickness"]=0
330  waveform_sel.grid(row=0, column=5, columnspan=1, padx=5, pady=5, sticky=tk.W)
331
332
333  save_csv_as_button = tk.Button(topFrame, text = 'Save Data Points As', \
334  command = save_csv_as)
335  save_csv_as_button.grid(row=0, column=6, padx=5, pady=5, sticky = tk.W)
336
337
338  time_estimation_label = tk.Label(topFrame, text="Time Est: 0.00 sec", fg="white", bg='red2', font='bold')
339  time_estimation_label.grid(row=0, column=7, padx=5, pady=5, sticky = tk.W)
340
341  root.protocol("WM_DELETE_WINDOW", on_close)
342  root.mainloop()

```