EE 224 Data Collection Project

Final report

Team Number: SDMay18-31

Client/Adviser: Professor Julie Dickerson

**Leif Bauer** Hardware Engineer, Scribe

**Christopher Caldwell** - Front-end Engineer, Documentation Lead

**Eric Joyce** - Firmware Engineer, Meeting Facilitator

**Martin Szuck** - Front-end Engineer

**Nicholas Starr** - Hardware Engineer, Adviser Contact

**Tyler Tran** - Firmware Engineer, Report Writer, Technical Resource Lead

Team Email: [sdmay18-31@iastate.edu](mailto:sdmay18-31@iastate.edu)  
Team Website: [sdmay18-31.sd.ece.iastate.edu](http://sdmay18-31.sd.ece.iastate.edu/#)

Revised: 04/21/2018

Table of Contents

[**List of Figures/Tables/Symbols/Definitions**](#_ewr6306mkcdo) **5**

[**1 Introduction**](#_uldci6239q0a) **7**

[1.1 Acknowledgement](#_2sz5apts9ou7) 7

[1.2 Problem and Project Statement](#_a98l5abq88c0) 7

[1.3 Operational Environment](#_2thegrgmlv30) 7

[1.4 Intended Users and Uses](#_132u383ethtt) 7

[1.5 Assumptions and Limitations](#_nvt07go5mjk1) 8

[Assumptions](#_fyoy26x1o17o) 8

[Limitations](#_f6id3u4czc1m) 8

[1.6 Expected End Product and Deliverables](#_q8bc725my2d9) 8

[**2 Design and Analysis**](#_b5iyvmrjus00) **8**

[2.1 Functional Requirements](#_2s8eyo1) 8

[2.2 Technology Considerations](#_579fvrc2v3zs) 9

[2.3 Related Products and Literature](#_gdfo6ox4i2cf) 10

[2.4 Labs](#_al948e190uhf) 10

[2.5 Possible Risks and Risk Management](#_rujilskhl4lb) 11

[2.6 Project Tracking Procedures](#_a4blnf2wqusj) 11

[**3 The Implementation**](#_i9wt1ldun88) **11**

[3.1 Circuit](#_5swv11cpn0rj) 11

[3.2 Front End](#_6gffi79b6yt8) 12

[3.3 Firmware](#_efw9gqlcor8o) 12

[**4 Hardware and Software Testing**](#_26in1rg) **13**

[4.1 Software](#_gg9d9vc1t71p) 13

[4.2 Hardware](#_8u828bkmd3lf) 13

[**5 Closing Material**](#_2jxsxqh) **14**

[5.1 Conclusion](#_z337ya) 14

[5.2 References](#_3j2qqm3) 14

[**6 Appendices**](#_gp4wbv362qzp) **15**

[6.1 Appendix I: User Manual and Getting Started Guide](#_67zck9orbl43) 15

[Basic Structure](#_qj4tnj29p0tt) 15

[Navigation Buttons](#_gdy7pgrn5nb3) 16

[ADC Configuration](#_muyra9qmmbrk) 17

[Digital Filter Configuration](#_fwmh0sxzvmx7) 19

[Data Collection Screen](#_elu87iy465ii) 20

[Digital Filtering](#_qzguechd0dop) 21

[6.2 Appendix II: Functional Diagrams](#_vwhv3w997h0j) 25

[6.3 Appendix III: Hardware Simulations](#_l1638jtiao4z) 27

[1st Order Low Pass Filter](#_xnuil5ty5o4q) 27

[5th Order Low Pass Filter](#_byj2v1onmrq) 28

[1st Order High Pass Filter](#_vvibzndz9ap5) 29

[5th Order High Pass Filter](#_6gpyllumo7e4) 30

[2nd Order Band Pass Filter](#_m5msd2jh0ga0) 31

[6th Order Band Pass Filter](#_vtg07onk2v4o) 32

[60 Hz Notch Filter](#_ijvf89s0ym7p) 33

[6.4 Appendix IV: Hardware Specifications](#_zl1tc8jpyso) 34

[Sensor Setup:](#_b5s6nspj6ls4) 34

[Hardware Setup:](#_7hbg8bzfxubp) 34

[Analog Sensor:](#_r85cn83nprl) 34

[I2C Sensor:](#_d5l4e7h7mlm) 35

[SPI Sensor:](#_2boenk3l67xl) 36

[Firmware Setup:](#_y8nw2r5dtqh0) 36

[Tunable Filter Measurement Method](#_7zz9keqkqesf) 37

[Setup:](#_sqtmfrh5kumv) 37

[Testing (Low Pass):](#_pb14w2vbrmkx) 37

[Testing (High Pass):](#_kgjkt53efspp) 37

[Testing (Band Pass):](#_k9qtyxtwjy4p) 38

[Measured Tunable Filter Characteristics](#_rb6m996xnfb0) 38

[1st Order Low Pass Filter](#_40lgk710e8yh) 38

[5th Order Low Pass Filter](#_vh7fz7xpdqbm) 39

[1st Order High Pass Filter](#_18vq7u9kzhlw) 39

[5th Order High Pass Filter](#_6bosacl7ice7) 40

[2nd Order Band Pass Filter](#_gfia667prgb1) 40

[6th Order Band Pass Filter](#_9e1qcqus4p9y) 41

[Firmware Testing Method](#_rd85v759qpwk) 41

[Muxes](#_m08j91ipcu3n) 41

[ADC 0 Muxes/Capture](#_7kkuhxk1w2mg) 41

[ADC 1 Muxes/Capture](#_awq5y0fsdblk) 41

[ADC 2 Muxes/Capture](#_5jbi8p6ixzra) 42

[ADC 0/1/2 Parallel Input](#_uwc5kb7xdjxy) 42

[Potentiometer Mux/Programming](#_qlxe0vlyxovw) 42

[Serial Communication Interfaces](#_lhzp80an540z) 43

[I2C](#_oin7d2nhswn) 43

[SPI](#_tqn0k0mflkuk) 43

[Filter Circuit Diagrams](#_pen7vy6xdho9) 43

[1st Order Low Pass Filter](#_g8djmmw7cbn5) 43

[5th Order Low Pass Filter](#_mpvenx2yk2ea) 44

[1st Order High Pass Filter](#_ttrvscrn30l4) 44

[5th Order High Pass Filter](#_ubnywo1f0co6) 45

[2nd Order Band Pass Filter](#_86zbhbijmuuy) 45

[6th Order Band Pass Filter](#_nti5zl5fr31) 46

[60 Hz Notch Filter](#_17216cfu1gb9) 46

[Passthrough](#_th2pz56asg1s) 47

[6.5 Appendix V: Lab Design Outlines](#_kjl4sgt3r5o) 48

[Synthesis of Sinusoidal Signals using Tuning Forks](#_4vlfnf37i7g9) 48

[Pre-Lab:](#_ppoiheyjrcv5) 48

[1 Overview](#_sge0l8qmho56) 48

[2 Post-Collection Processing](#_jpgli61g3k4r) 48

[3 Labeling Plots in MATLAB](#_ooios2rl5up) 48

[4 Frequency view of signals in Matlab](#_nrje0j1hziy0) 49

[5 Lab Procedure Signal Collection](#_em5ybfekqgib) 49

[Fourier Transform Lab:](#_u3v6pe35dym5) 52

[Introduction](#_1dl6qi2djdze) 52

[Background](#_9oy86ogsdt4a) 52

[Data Collection](#_cae1dz96r942) 53

[Pulse Rule Calculation](#_46stkucosnya) 53

[VAD Lab Outline:](#_debin70yax2) 54

[Data Collection:](#_3g1fuk4h7lkm) 54

[MATLAB:](#_jptx8i5895y) 54

[Report Extras:](#_6kno0etmhtv7) 55

[Bonus Vocab:](#_y3glbxkst3mg) 55

[Noisy IR Sensor Lab:](#_3w6g7aw0yfab) 55

[Abstract:](#_agptpyxeulru) 55

[Introduction:](#_m0sc77302bfk) 55

[Process:](#_eck9ao6xx0us) 56

[Step 1: Configuration](#_h11p6prvo03) 56

[Step 2: Data Collection](#_x9hksq3uu0pc) 56

[Step 3: First, we’ll handle construction of FIR filters in MATLAB.](#_qzo2rhvblffc) 56

[Step 4: Implementing digital filters on the device.](#_k63e8bs61qjt) 57

[Automation System Lab:](#_47q5ppad6k4q) 57

[Intro:](#_yn3s8bdsm68c) 57

[Background:](#_gz9623bvu5la) 57

[Setup:](#_8yaa5pkwuv9) 58

# List of Figures/Tables/Symbols/Definitions

* Figure 1 - GUI Basic Structure
* Figure 2 - GUI Navigation Buttons
* Figure 3 - GUI ADC Sampling Configuration
* Figure 4 - GUI ADC Filter Configuration
* Figure 5 - GUI Digital Sensor Configuration
* Figure 6 - GUI Data Collection
* Figure 7 - GUI Digital Filtering Selection
* Figure 8 - GUI FIR Filter Design
* Figure 9 - GUI IIR Filter Type Selection
* Figure 10 - GUI IIR Filter Bandwidth Selection
* Figure 11 - Filter Circuit Block Diagram
* Figure 12 - Python Front End Data Diagram
* Figure 13 - Firmware Task Diagram
* Figure 14 - 1st Order Low Pass Filter Simulation Schematic
* Figure 15 - 1st Order Low Pass Filter Simulation Response
* Figure 16 - 5th Order Low Pass Filter Simulation Schematic
* Figure 17 - 5th Order Low Pass Filter Simulation Response
* Figure 18 - 1st Order High Pass Filter Simulation Schematic
* Figure 19 - 1st Order High Pass Filter Simulation Response
* Figure 20 - 5th Order High Pass Filter Simulation Schematic
* Figure 21 - 5th Order High Pass Filter Simulation Response
* Figure 22 - 2nd Order Band Pass Filter Simulation Schematic
* Figure 23 - 2nd Order Band Pass Filter Simulation Response
* Figure 24 - 6th Order Band Pass Filter Simulation Schematic
* Figure 25 - 6th Order Band Pass Filter Simulation Response
* Figure 26 - 60 Hz Notch Filter Simulation Schematic
* Figure 27 - 60 Hz Notch Filter Simulation Response
* Figure 28 - Analog Sensor Pin Setup
* Figure 29 - I2C Sensor Pin Setup
* Figure 30 - SPI Sensor Pin Setup
* Figure 31 - 1st Order Low Pass Filter Tunable Characteristics
* Figure 32 - 5th Order Low Pass Filter Tunable Characteristics
* Figure 33 - 1st Order High Pass Filter Tunable Characteristics (Rev 2 Measurement)
* Figure 34 - 5th Order High Pass Filter Tunable Characteristics (Rev 2 Measurement)
* Figure 35 - 2nd Order Band Pass Filter Tunable Characteristic
* Figure 36 - 6th Order Band Pass Filter Tunable Characteristics
* Figure 37 - 1st Order Low Pass Filter Schematic
* Figure 38 - 5th Order Low Pass Filter Schematic
* Figure 39 - 1st Order High Pass Filter Schematic
* Figure 40 - 5th Order High Pass Filter Schematic
* Figure 41 - 2nd Order Band Pass Filter Schematic
* Figure 42 - 6th Order Band Pass Filter Schematic
* Figure 43 - 60 Hz Notch Filter Schematic
* Figure 44 - Passthrough Filter Schematic
* Figure 45 - Intro Lab Tuning Fork
* Figure 46 - Intro Lab Tuning Fork Table
* Figure 47 - Pulse Lab PPG Signal
* Figure 48 - Pulse Lab PPG Sensor Functionality

# 1 Introduction

## 1.1 Acknowledgement

We would like to thank Professor Julie Dickerson of the ECpE department for providing technical advice for the project in the capacity of faculty adviser. Additionally, we would like to thank Matthew Post and ETG for providing advice for real world considerations for the design of the project. We would also like to thank ETG for help in picking and ordering parts and supplies for the PCB and sensors.

## 1.2 Problem and Project Statement

In our Electrical Engineering curriculum, Signals and System I (EE 224) serves as the introductory course for signal processing and provides a theoretical basis for communications theory. The concepts covered in EE 224 are foundational to the field of electrical engineering - specifically in our understanding of how signals behave when passed through various systems.

Currently, the laboratory component of this course focuses on introducing these concepts through Matlab code snippets and pseudo-analog signals (vectors that correspond to a signal that has been sampled at such a rate that it "appears to be" or "behaves as" an analog signal). Both of these issues detract from the student's learning environment.

Our proposed solution to this problem is to introduce a new lab interface that will simplify certain components of the lab, while highlighting course content to make it more available to students. We plan to do this by introducing a data acquisition device called CyDAQ that brings in real-world signals taken in by a variety of analog and digital sensors, passes the signals through analog filters implemented on a PCB or digital filters generated on a lab computer and applied by the front end. From here, data can be collected and exported to a .csv/.mat file or displayed in figures. These files are then ready for further processing and analysis in MATLAB.

## 1.3 Operational Environment

The environment this product will operate in is the Introduction to Signal Processing Laboratory. This room is a busy laboratory in the ECpE department due to being booked for several 200 and 300 level courses throughout the semester. Because of the large number of students passing through the classroom, the device needs to be easily configurable and resettable by students or by a lab TA.

## 1.4 Intended Users and Uses

This product is intended to provide an interactive and experiential introduction to Signal Processing by providing a data capture toolbox to collect and process real-world signals. Students will be the primary intended users of the device. We hope that more students will find the content presented in EE 224 easier to learn, and will be more able to apply concepts of Signal Processing to real world issues and tasks.

## 1.5 Assumptions and Limitations

### Assumptions

CyDAQ will support the single user use case.

CyDAQ will be primarily used for the EE 224 lab.

The users of CyDAQ shall have a basic understanding of computer interfaces.

CyDAQ shall have access to the USB port from a lab computer to power the device and for communication to the lab computer.

### Limitations

CyDAQ shall cost no more that $300 per unit. (Client Resources)

The CyDAQ shall only have to support three analog input ports. (To support multiple sensors at once)

CyDAQ shall only have to support two digital input ports. (One for and one for SPI)

## 1.6 Expected End Product and Deliverables

* A software interface for collecting data from a variety of sensor types which must be able to control sampling rate, bandwidth, and allow for a selection of basic filtering options.
* An inexpensive hardware board that allows multiple off the shelf sensors to be installed. The hardware used must be inexpensive and commercially available.
* A set of laboratory outlines that teaches students how to use the hardware and software, as well as a set of laboratory outlines that will apply topics and concepts learned in lecture.

# 2 Design and Analysis

## 2.1 Functional Requirements

* The device shall receive different types of signals from different sensors.
* The device shall switch the signal path between different types of filters.
* The device shall have a port for connection to a computer.
* The device shall provide an interface to access collected data.
* The device shall be able to receive commands to adjust settings:
  + Sample Frequency
  + ADC channels to collect data
  + Analog filters to use on ADC channels
* The UI shall provide means of modifying filter settings (bandwidth, filter type, etc).
* The UI shall provide means to perform an FFT on received data.
* The UI shall provide a time-domain display of modified data.
* The UI shall provide a frequency-domain display of modified data.
* The Controller shall be able to store settings and data locally.
* The device shall have keyed ports so that polarity cannot be reversed.

Below are the standards that we will be using for our project:

**Coding Standards:**

*Barr Group Embedded Systems Standards for C* [1]

*PEP 8 - Style Guide for Python Code* [2]

*Matlab 2.0 Style Guidelines* [3]

## 2.2 Technology Considerations

Our design involves using a microprocessor to process incoming signals and route them through the appropriate discrete circuits (channels). This opens up the possibility of recording data in real time that students are able to work with. This recorded data is usable in the form of exported .mat and .csv files from our front end. These files can be imported directly into MATLAB and are an improvement over using simulated signals in MATLAB.

By packaging analog circuits into a “black box”, the students can see and recognize the circuit, but do not need to debug a circuit while simultaneously trying to understand new course content; they can “play” with circuit characteristics and see consistent results.

The major theme of EE 224 is to be the first major exploration of frequency domain mathematics and analysis in the Electrical Engineering curricula. The only direct Electrical Engineering prerequisite is Introductory Circuits (EE 201). If a student is following the flowchart (referencing 2017-18), they will be in their fourth semester and be just entering their first full semester of Electrical/Computer Engineering coursework. In the flowchart it is paired with EE 230, early in the curriculum. Circuits and Frequency Domain Analysis are fundamental to the development of the student and are the major building blocks they will be working with for the rest of their academic careers.

The CyDAQ takes filtering circuits that they have seen in circuits courses and puts them in a black box. This allows the EE 224 labs, involving the CyDAQ, to focus on the effects of filters on incoming signals. By taking this “black box” approach to analog systems, the students will not need to debug a circuit while simultaneously trying to understand new course content. Instead, they can “play” with analog system characteristics and achieve consistent results.

## 2.3 Related Products and Literature

Courses like Signals and Systems I are common for electrical engineering majors across the country. While several schools have pushed to include a heavier focus on digital signal processing, it is not uncommon for schools to also include courses similar to EE 224 that deal with a mix of digital and analog systems. Schools like Georgia Tech, University of Illinois, University of Michigan, and Purdue have similar courses. Like EE 224, these courses rely heavily on MATLAB, but we believe that CyDAQ could be utilized to improve labs at these schools as well.

While other commercial software packages for data acquisition already exist (e.g. LabVIEW, SignalExpress), introducing these to the EE 224 lab would not be helpful. Use of these packages would shift focus away from the systems the signals are going through. Similar to the issue with using MATLAB, students would be spending unnecessary time working with new software packages. Our user interface is designed specifically to work with the device we’ve designed, and as such, should require minimal time for troubleshooting and setup on a weekly basis.

A question that has been posed is why we haven’t looked into the usefulness of National Instruments’ myDAQ device. We believe that myDAQ is less useful for EE 224, because it is prohibitively expensive and requires students to learn Python scripting or Labview on top of new signals and systems content. To replicate our product, myDAQ would require students to create additional analog filtering circuits, which does not align with the course objectives.

We believe that myDAQ is an excellent product for a class that is interested in teaching circuits and signals in the same course. With our separation of the topics, myDAQ is not a good use of financial funds and not conducive to the learning goals of this course. Plus, with myDAQ’s expensive sensor accessories, future lab development could be cost prohibitive. Once a set of sensors has been purchased, it would be difficult to justify purchasing more sensors to update the lab.

## 2.4 Labs

The core signal processing concepts that shall be addressed in this lab are derived from existing EE 224 lab manuals. The majority of the current EE 224 labs use MATLAB to generate signals that are then analysed by students. Our project aims to replace the MATLAB signal generation with signals gathered from real-world sensors. While the lab outlines that will be created in this project are derived from existing labs and sections that involve simulated signals will be replaced with manipulations of the real world signals gathered by our device. Listed below are the current labs that are used in the course:

**Lab List**

* Tuning Fork Lab
* Signal Generation Lab
* Chirp Lab
* Simulink Fourier Analysis
* Fourier Series Lab
* Image Intro Lab
* Finite-Impulse-Response Lab
* Digital Image Processing Lab
* AM Communications Lab
* Band Pass Filter Lab

A potential approach that could have been used for the collection of real signals, would be measuring electrical signals generated with function generators. The data could be collected using an oscilloscope through a Signal Express interface. This data could then be imported into Matlab for analysis. This approach was abandoned due to the unreliability of maintaining a Signalexpress environment across many lab sessions.

We decided that the device should be able to modularly take signals from several sensors and allow for control of those signals through various signal processing devices. These signals are then sent through a microcontroller to the computer through a UART connection. On the computer the signals are then visualized through a GUI. The signal processing is also controlled through the GUI.

## 2.5 Possible Risks and Risk Management

One of the biggest challenges will be to ensure that the device improves the lab.

* Testing out different types of sensors and ensuring that they match the key concepts may present several difficulties.
* Ensuring that the device is intuitive to use may present some difficulties
* We may have some issues with writing labs to incorporate the necessary course topics.
* We may have some issues with designing the circuit and PCB to act in a reliable manner
* We must ensure that the cost of the device is less than $300

## 2.6 Project Tracking Procedures

Our group has used Gitlab to track the progress of our development. We have utilized several different features in Gitlab such as milestones for deliverables and issues for smaller day-to-day tasks. This has allowed us to track our progress efficiently and complete our goals in a timely manner.

# 3 The Implementation

## 3.1 Circuit

CyDAQ utilizes the TI Tiva C TM4C123G Launchpad platform and is designed to accept up to three different types of analog inputs, two different types of digital inputs, one type of I2C input, and one type of SPI input. CyDAQ connects analog sensors through a filter network, while digital sensors are connected directly to the TI Launchpad. The filter network allows for the choice of one of eight filters for each sensor connected. The filters included are a 1st order low-pass filter, a 5th order low-pass filter, a 1st order high-pass filter, a 5th order high-pass filter, a 2nd order band-pass filter, a 6th order band-pass filter, a 60 Hz notch filter, and a passthrough filter. Each of the filter’s cutoff frequencies or center frequencies are tunable other than the 60 Hz notch filter and the passthrough filter. After the signals are routed through the filter network, they are read by the TI Launchpad using analog inputs. See Figure 1 for the circuit filter network design.

## 3.2 Front End

The chosen GUI development kit is Tkinter due to its popular use in Python GUI development and the team’s familiarity with it. The GUI generates a configuration file that will store user settings. This allows a user to not lose their configuration upon application termination. Along with this, the application outputs data in two file formats (.CSV and .MAT) allowing for easy MATLAB interfacing. The front end has the ability to select between different analog and digital sensor ports and set the appropriate settings related to sampling rate, bandwidth, and filter type. The front end application can construct IIR and FIR filters to apply to collected data before entering MATLAB. The team used: SciPy, NumPy, Matplotlib.pyplot, JSONpickle, and pyserial to implement different aspects of the front end. See Figure 2 for the front end data diagram to understand how the data will flow through the Python front end.

## 3.3 Firmware

A firmware component is required for this project as the signal capture device needs to manage the routing of signals from input devices through several filter networks, control capture settings, and communicate with the front-end. The firmware team has chosen to use the FreeRTOS project as the base threading layer to handle context switching between tasks for the capture device. See Figure 3 for the relation between the firmware tasks.

The firmware’s functionality is divided into three tasks: UART TX, UART RX, and digital sensor tasks. The digital sensor task is responsible for retrieving data from digital sensors connected to the capture device and buffering the read data from these devices.

The UART TX and UART RX tasks are responsible for communicating with the front-end component over UART. The UART TX task takes the buffered data from the ADC, I2C, and SPI sensors and packages them for transfer to the front end. The UART RX task receives configuration information, such as sampling rate, from the front end and enables said configurations if they differ from the current state. The data packets are sent on UART over USB which provides a reliable communication channel with the front end. Packets sent between the firmware and front-end are verified using an appended CRC on both ends. If the front-end does not receive a response from the firmware, a timeout will occur and the user will be notified of the error. The firmware also uses a watchdog timer to periodically restart the device when it is not in use.

# 4 Hardware and Software Testing

## 4.1 Software

Both the front-end and firmware have undergone systems tests to minimize bugs. Functional testing was used to verify that each of the components of the design functioned properly. To ensure software functionality, development was performed on feature branches for each proposed addition. As part of the code review process, feature testing was performed to confirm that the addition met the design requirements. Then integration testing was performed to ensure that the feature did not break previous functionality. Once these two review processes were completed, we merged the proposed featured into our master branch. The front-end was evaluated by our client and faculty advisor to ensure that its design is simple and intuitive.

## 4.2 Hardware

Each of the filters have been simulated using PSpice simulation tools and frequency analysis has been performed to ensure that the filters meet our original design intentions (see Appendix III). These filters were also simulated with the unity gain stage to ensure that op-amps were set up correctly. Then, these filters along with their op-amps were tested on circuit boards at specific frequencies to test their amplitude response. The demultiplexer component has also been tested on its own to ensure minimum cross-talk between channels.

A microcontroller has been used to take in data to verify the functionality of the circuit system on a printed circuit board (PCB). Each of the components were tested using testing points on the PCB to ensure expected performance. The filter tunability has been tested to ensure that each of the filters operates within the functional frequency range of the device.

The filter’s potentiometers were tested using a picoscope to ensure proper I2C communication between the firmware and the potentiometers. The firmware was also tested in combination with the PCB to ensure that each of the data signal paths function properly.

The potentiometer tunability range was tested using the following setup. A waveform generator was used to input a signal into the sensor port. The output root mean square (RMS) voltage was measured using a multimeter, and an oscilloscope. The oscilloscope was used to verify that the signal maintained its shape. In order to determine the cutoff frequency of the filters, the waveform generator frequency was set to a frequency within the filter’s passband. At that frequency the RMS voltage was measured. Then, the frequency was swept until the RMS voltage reached half of the passband voltage. This frequency should be the -3dB cutoff frequency for the filter, and is recorded as such. Then the filter was tuned to 15 other points in its range, and the cutoff frequency was measured for each of these points. For the points in between these recorded points, a linear interpolation was used (more specific setup in Appendix IV).

# 5 Closing Material

## 5.1 Conclusion

The team successfully created three revisions of the CyDAQ PCB. The hardware team was able to verify all eight analog filters, filter muxes, and filter tuning. The firmware was able to collect data from digital sensor and sample analog signals at a rate of 8000 Hz. We also were able to receive and apply settings from the front-end. The front-end team was able to create a simple and intuitive GUI to configure settings, collect and visualize data, and apply digital filters.

The goal of this project is to bridge the conceptual gap between EE 224 concepts and real world signals. Our design allows students to quickly and easily obtain signals from the real world using analog and digital sensors. The CyDAQ also allows students to route signals through a series of analog and digital filters to see their effects on different signals. The use of a bare-bones custom built GUI allows students to explore the core course concepts with little to no experience with MATLAB.

## 5.2 References

Post, M. (2017, September). Personal interview.

Dickerson, J. (2017, September). Personal interview.

[1] General Rules. (2016, May 13). Retrieved October 24, 2017, from <https://barrgroup.com/Embedded-Systems/Books/Embedded-C-Coding-Standard/General-Rules>

[2] Johnson , R. (2014, March 27). MATLAB Style Guidelines 2.0. Retrieved October 24, 2017, from <https://www.mathworks.com/matlabcentral/fileexchange/46056-matlab-style-guidelines-2-0>

[3] Nicholaus Smith | Jul 17, 2013. (2017, January 31). The Engineer's Guide To High-Quality PCB Design. Retrieved October 24, 2017, from <http://www.electronicdesign.com/embedded/engineer-s-guide-high-quality-pcb-design>

[4] PCB Design Layout Rules Recommendation. (n.d.). Retrieved October 24, 2017, from <http://www.electronics-project-design.com/PCB-Design.html>

[5] PEP 8 -- Style Guide for Python Code. (n.d.). Retrieved October 24, 2017, from <https://www.python.org/dev/peps/pep-0008/>

# 6 Appendices

## 6.1 Appendix I: User Manual and Getting Started Guide

### Basic Structure

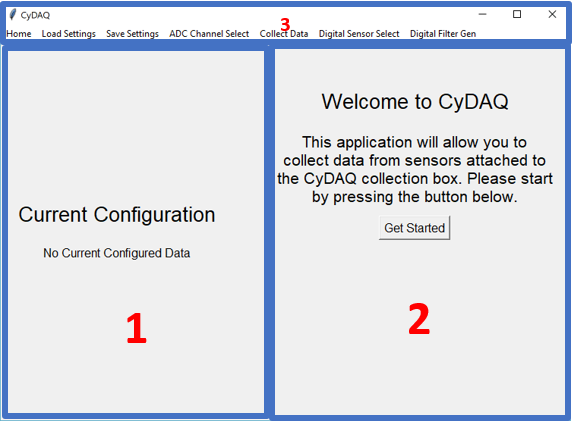


Figure 1: GUI Basic Structure

1) Left Panel: Displays the current sensor configuration that will be loaded onto the CyDAQ hardware. Updated as the settings are configured.

2) Right Panel: Primary user interaction panel. Panel contents gets changed as the user moves through levels of the UI.

3) Navigation Toolbar: Provides navigation buttons to CyDAQ features.

### Navigation Buttons

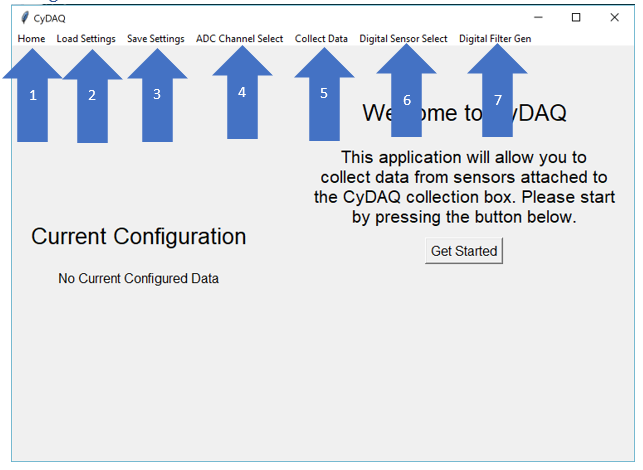


Figure 2: GUI Navigation Buttons

The navigation bar allows the user to change the CyDAQ feature they wish to use.

1) Home, brings the user back to the Welcome Page (Shown Above)

2) Load Settings, launches a prompt for the user to select a settings file (JSON). The stored settings will be loaded into the current CyDAQ session.

3) Save Settings, launches a prompt for the user to save the current CyDAQ settings to a file for future use. Stored as a JSON file.

4) ADC Channel Select, launches the analog sensor configuration screen.

5) Collect Data, launches the data collection screen.

6) Digital Sensor Select, launches the digital sensor configuration screen.

7) Digital Filter Gen, launches the digital filtering screen.

### ADC Configuration

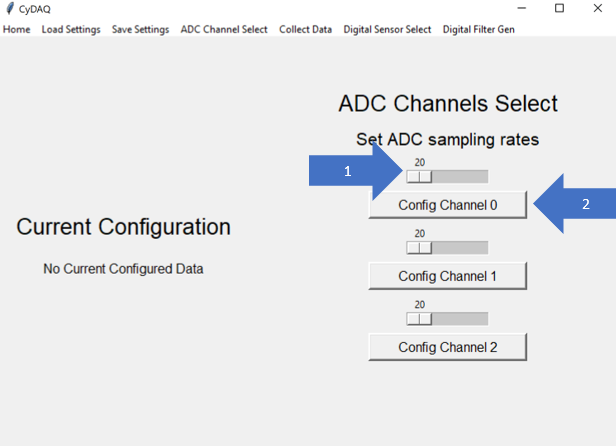


Figure 3: GUI ADC Sampling Configuration

The CyDAQ device has three ADC channels that can be used simultaneously. The device is limited to a max collective sampling rate of 8000 samples per second. An example of a valid ADC configuration: ADC0 = 200 samples per second, ADC2 = 3000 samples per second.

1) Slider: Sets the sampling rate for an ADC channel. Restricted to a range between 20 and MAX\_VAL, where MAX\_VAL is 8000 – the sampling rate currently configured for all other analog sensors.

2) Button: Configures the indicated ADC channel with the sampling rate indicated by the slider above the button. Launches the filter path configuration page.

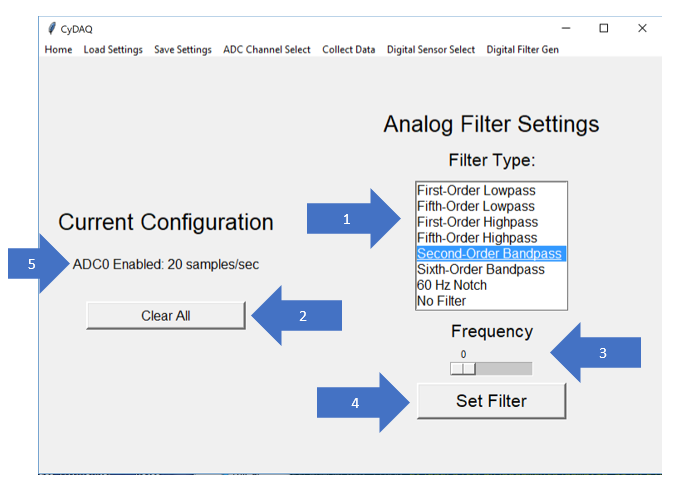


Figure 4: GUI ADC Filter Configuration

The above page configures the filter route for the current ADC channel, indicated by arrow 5. Each ADC channel can be configured to route through one of eight analog filter paths on the CyDAQ device. For all paths, beside the “60 Hz Notch Filter” and the “No Filter” path, a frequency setting must be provided. For the highpass and lowpass filter, the frequency is a corner frequency. The frequency setting for the bandpass filters are a corner frequency.

1) List of available filter paths. If another channel is configured to use a path, that filter will be removed from the list.

2) Clears the current configuration that will be sent to the firmware.

3) Frequency adjustment slider, allows frequency adjustments between 0 and 8000.

4) Applies the filter configuration settings to the ADC channel under configuration.

5) Current ADC channel under configuration.

### Digital Filter Configuration

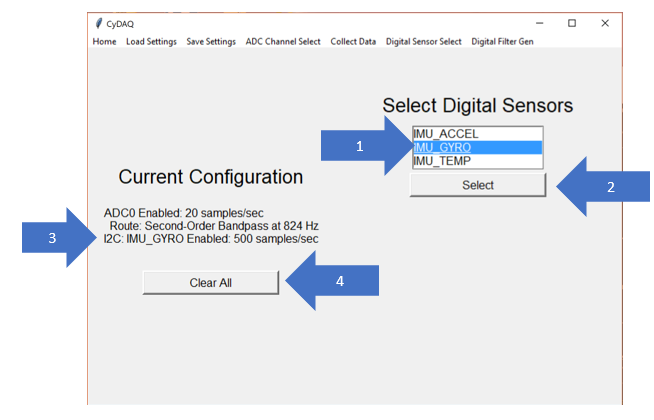


Figure 5: GUI Digital Sensor Configuration

The digital sensor configuration page allows a user to select a digital sensor to use. Since the CyDAQ device has two digital sensor ports, one for SPI and one for I2C, a user can have 1 of each type of sensor configured simultaneously. These sensors have fixed sampling rates.

1) List of available digital sensors.

2) Button: applies the digital sensor setting.

3) Settings text indicates that I2C port is set for the IMU\_GYRO sensor.

4) Clears the current configuration that will be sent to the firmware.

### Data Collection Screen

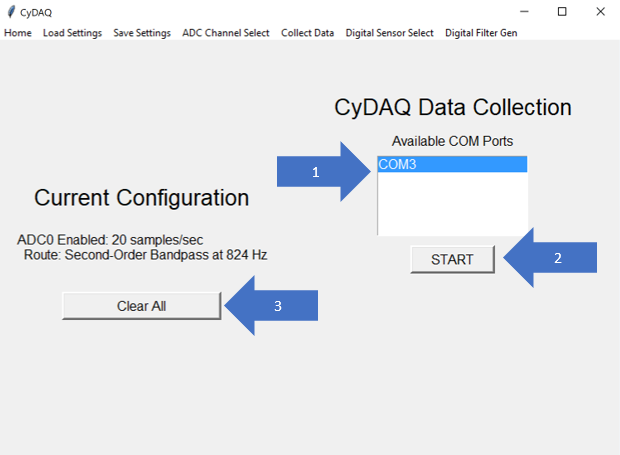


Figure 6: GUI Data Collection

The data collection screen allows the user to select a USB device to collect data from. If there is more than 1 COM port in the device list, the user will need to find the correct CyDAQ device (Stellaris) from the Device Manager. Once the user starts the data collection, the configuration data will be sent to the CyDAQ device, and the device will poll data with the configured settings until the user hits the stop button. Once the user hits the stop button, a File Explorer prompt will show up allowing the user to save the collected data to a CSV or MAT file. A prompt will be given for each sensor configured.

1) List of available COM ports attached to the lab computer, will update periodically as devices are removed and added.

2) Start/Stop Button: Starts and stops data collection. Stopping collection will launch a file save prompt.

3) Clears the current configuration that will be sent to the firmware.

### Digital Filtering

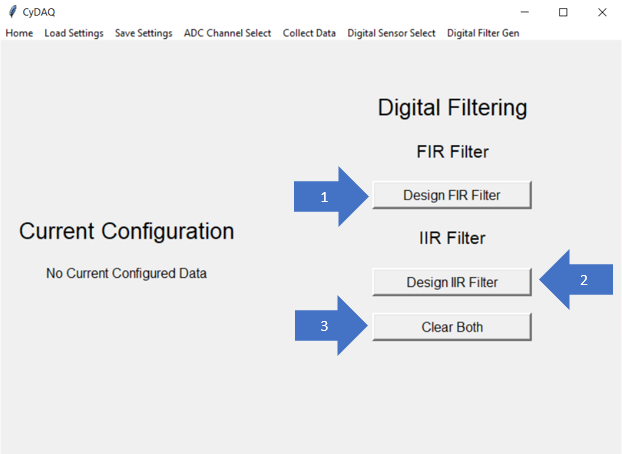


Figure 7: GUI Digital Filtering Selection

The digital filtering screen allows a user to open a CyDAQ data file, formatted as a MAT or CSV, and perform digital filtering on the stored data. It allows the user to select and design either a IIR or FIR filter.

1) Button, launches a prompt for the user to select an input file, then launches the FIR filter design page.

2) Button, launches a prompt for the user to select an input file, then launches the IIR filter design page.

3) Clears the current digital filtering settings.

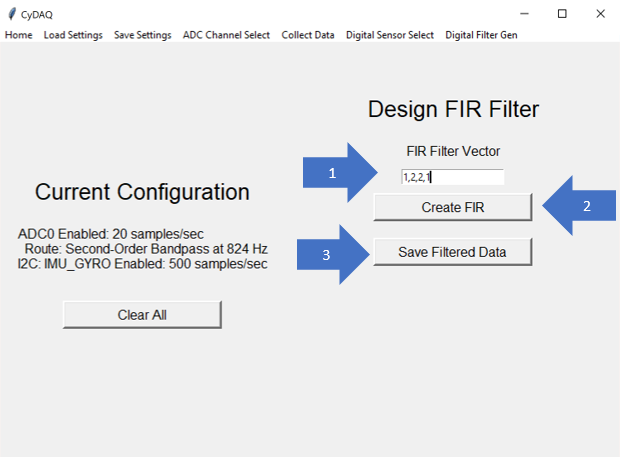


Figure 8: GUI FIR Filter Design

The FIR filter design page allows the user to input the impulse response of the FIR filter they want to design. Once the coefficients are provided, the user can create the filter, plot the filter, and plot the filtered signal. The user can also save the filtered data to a MAT or CSV file.

1) Input Field, allows the user to input FIR filter coefficients as a series of comma separated values.

2) Button, designs a filter with the given coefficients, plots the filter response, and plots the filtered signal.

3) Button, saves the last filtered signal.

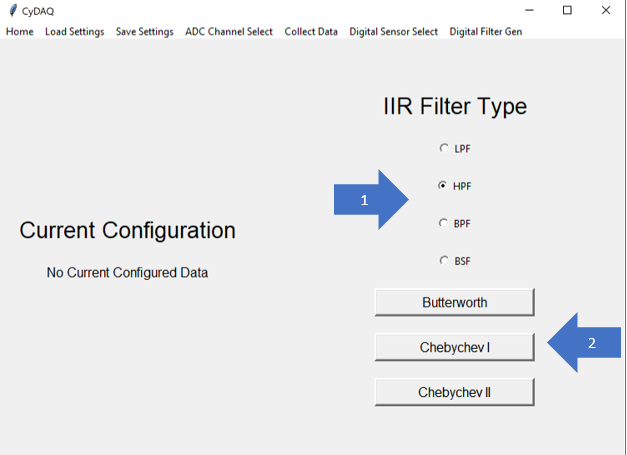


Figure 9: GUI IIR Filter Type Selection

The IIR filter selection field allows a user to select the type of filter they wish to design. They can select Low-pass, High-pass, Band-pass, or Band-stop filters. They also must select the type of characteristics they wish their filter to take on, such as a Butterworth filter or a Chebychev filter.

1) Choice Buttons, selects the type of filter the user wants to design.

2) Buttons, selects the filter characteristics that the user want their filter to take on, such as maximizing response flatness in the passband (Butterworth). Launches the frequency selection screen.

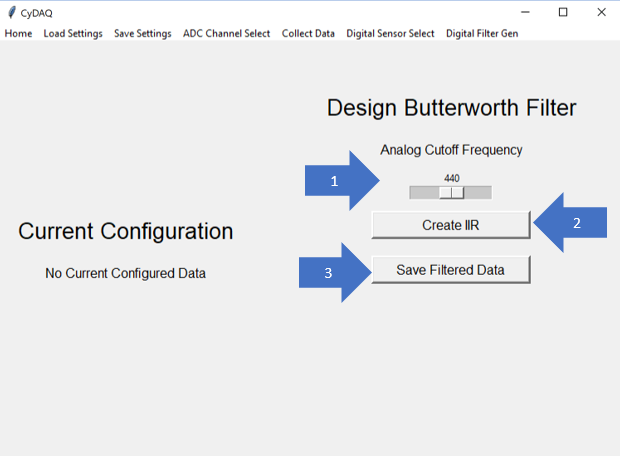


Figure 10: GUI IIR Filter Bandwidth Selection

The IIR filter design page allows the user to select the corner/center frequency for the IIR filter they are designing. Once a frequency is selected, the user can create the filter, plot the filter, and plot the filtered signal. The user can also save the filtered data to a MAT or CSV file.

1) Slider, allows the user to select the corner/center frequency for their IIR filter.

2) Button, designs a filter with the given frequency, plots the filter response, and plots the filtered signal.

3) Button, saves the last filtered signal.

## 6.2 Appendix II: Functional Diagrams

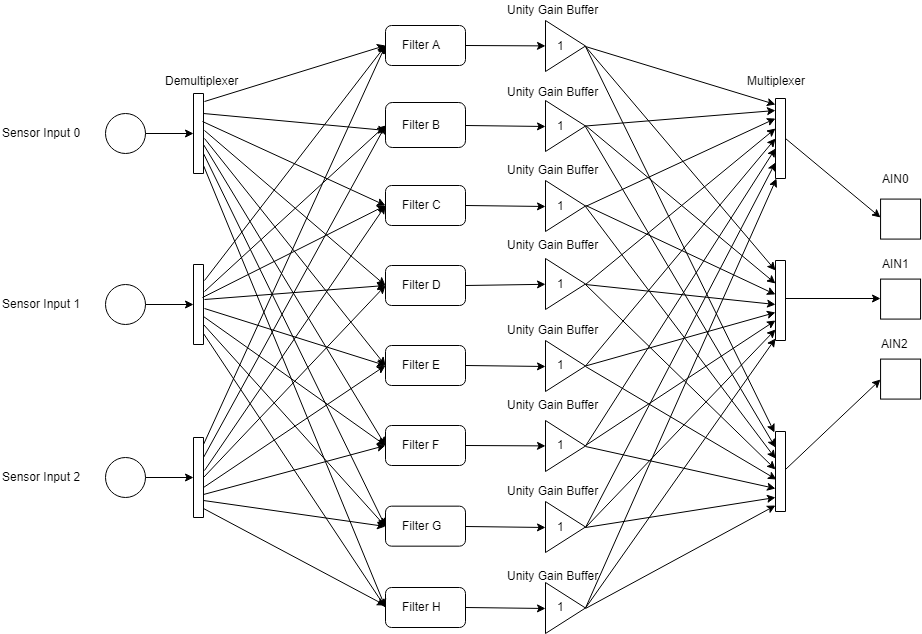


Figure 11: Filter Circuit Block Diagram

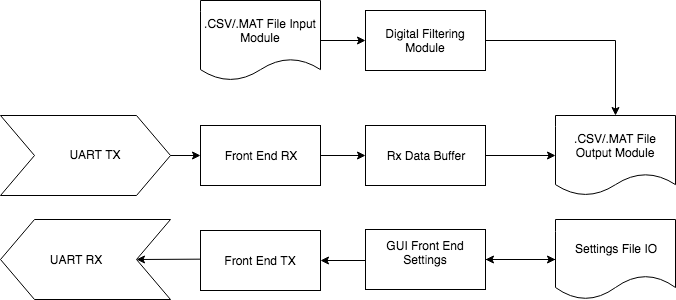


Figure 12: Python Front End Data Diagram

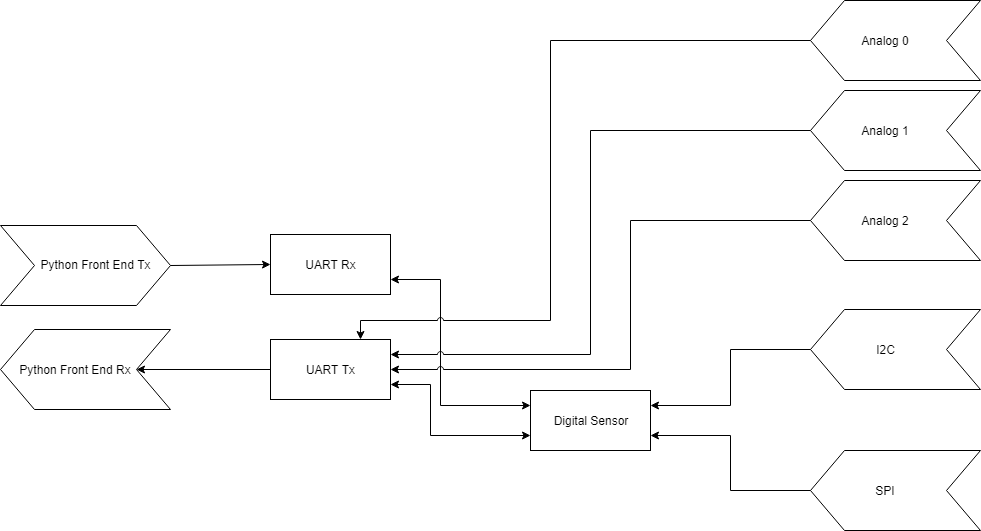


Figure 13: Firmware Task Diagram

## 6.3 Appendix III: Hardware Simulations

### 1st Order Low Pass Filter

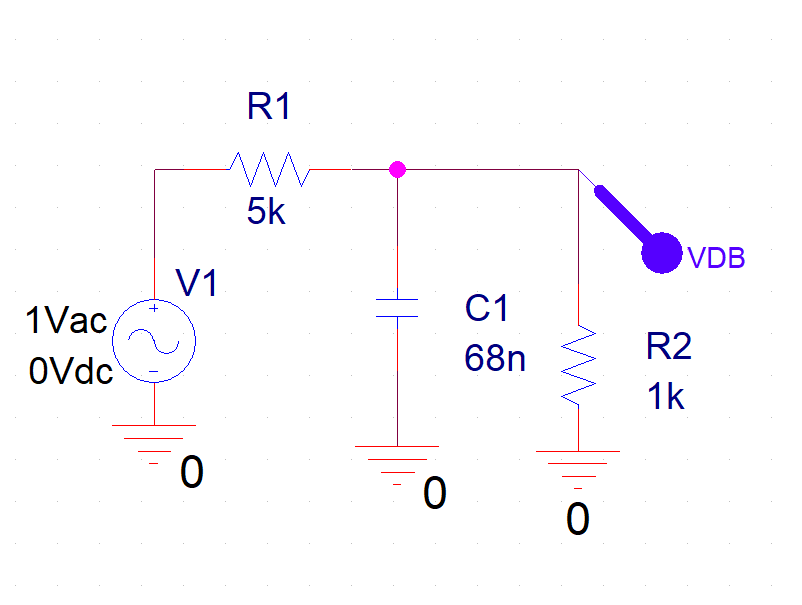


Figure 14: 1st Order Low Pass Filter Simulation Schematic

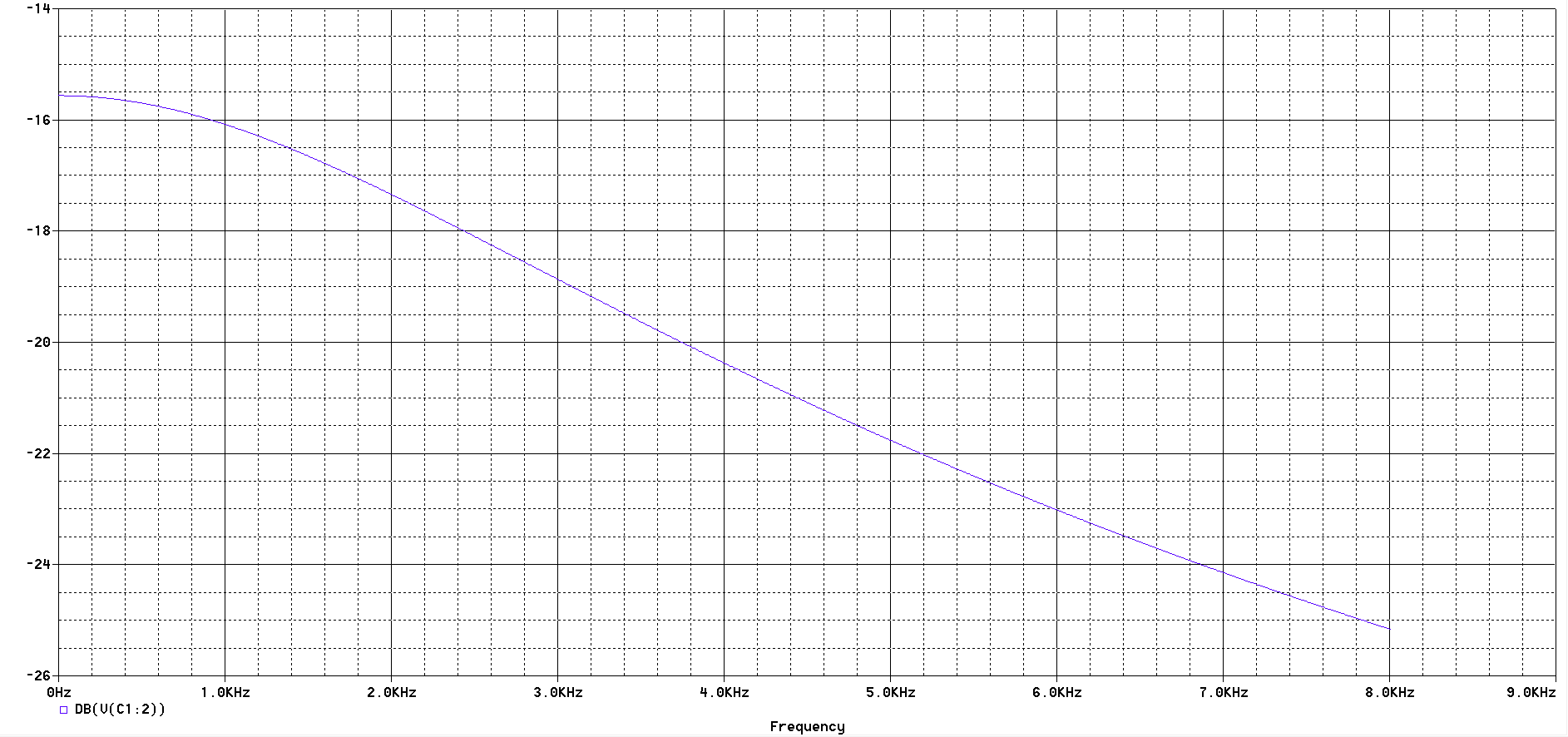


Figure 15: 1st Order Low Pass Filter Simulation Response

### 5th Order Low Pass Filter

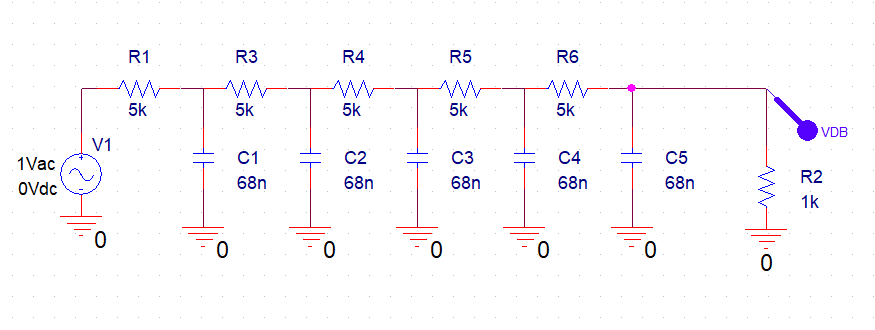


Figure 16: 5th Order Low Pass Filter Simulation Schematic

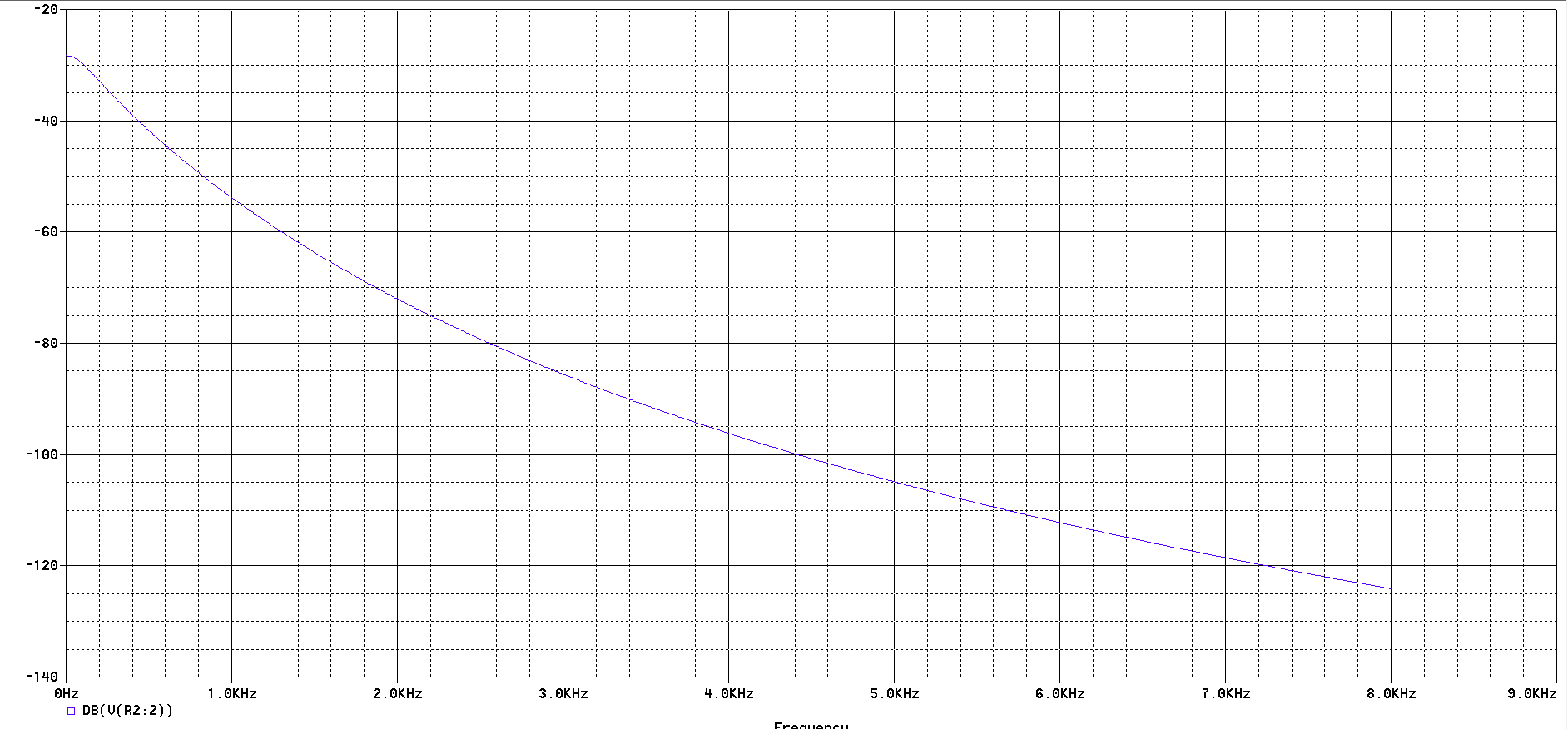


Figure 17: 5th Order Low Pass Filter Simulation Response

### 1st Order High Pass Filter

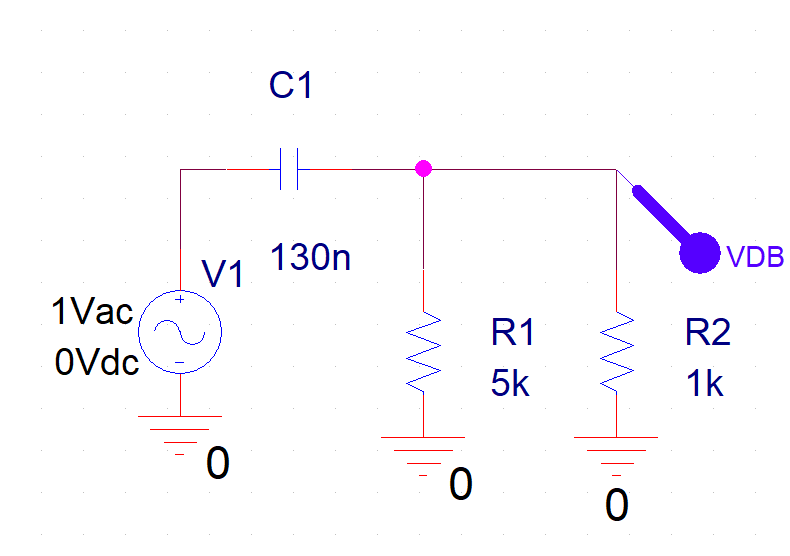


Figure 18: 1st Order High Pass Filter Simulation Schematic

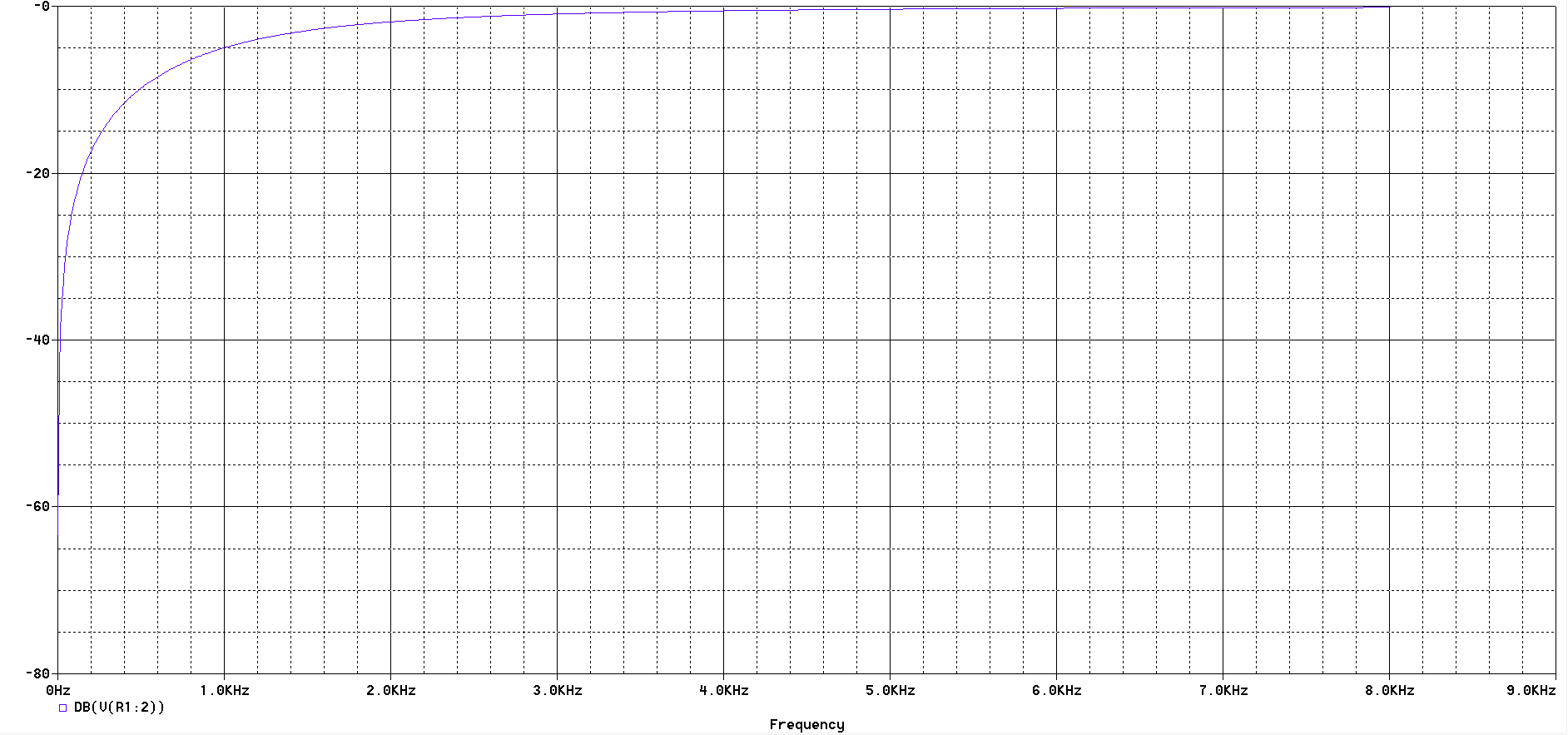


Figure 19: 1st Order High Pass Filter Simulation Response

### 5th Order High Pass Filter

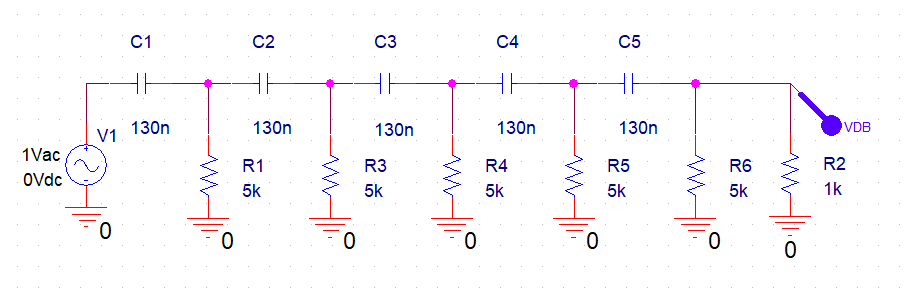


Figure 20: 5th Order High Pass Filter Simulation Schematic

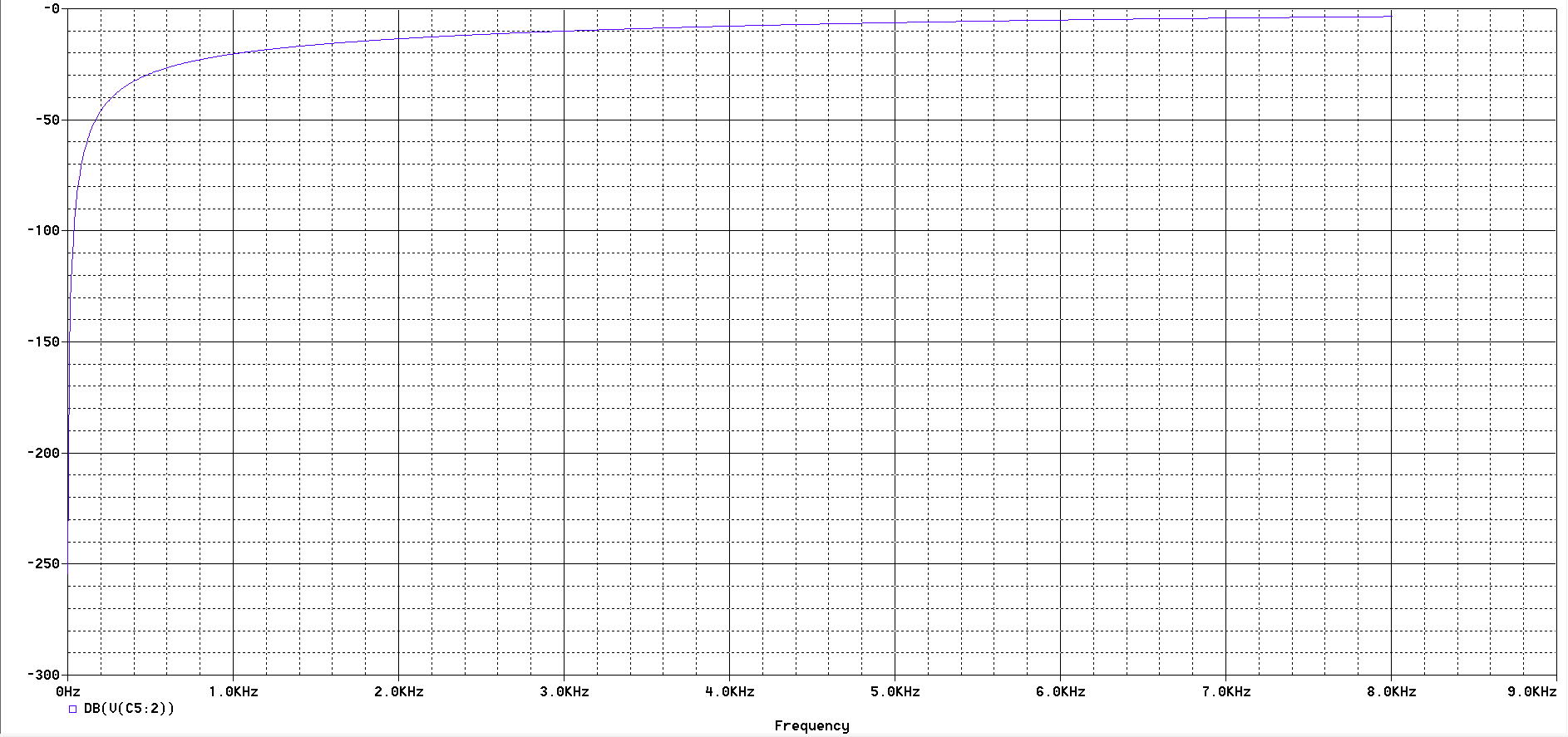


Figure 21: 5th Order High Pass Filter Simulation Response

### 2nd Order Band Pass Filter

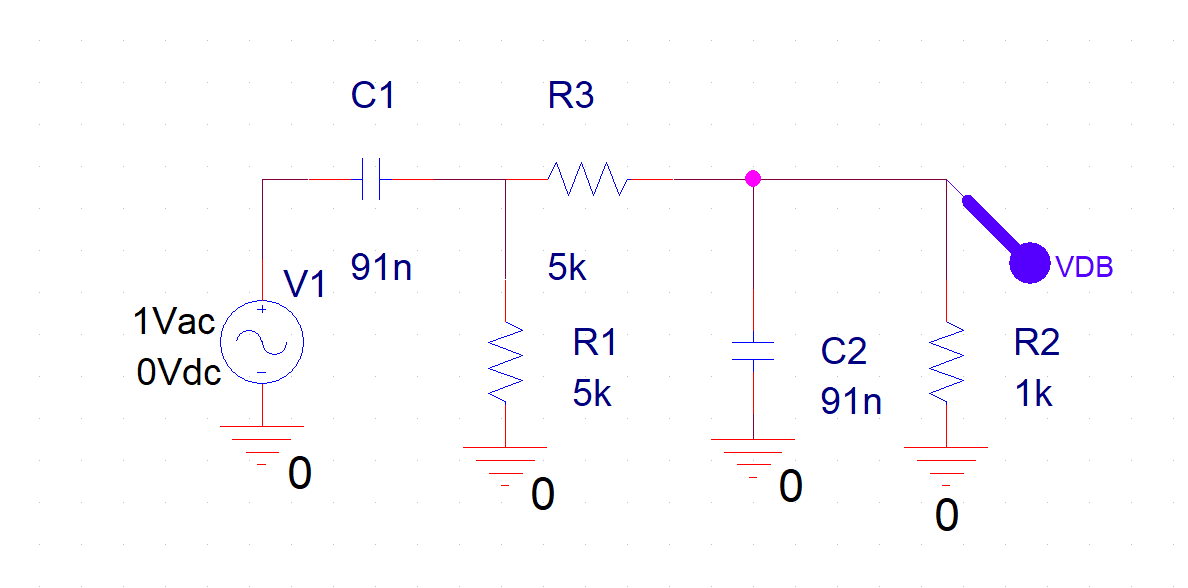


Figure 22: 2nd Order Band Pass Filter Simulation Schematic

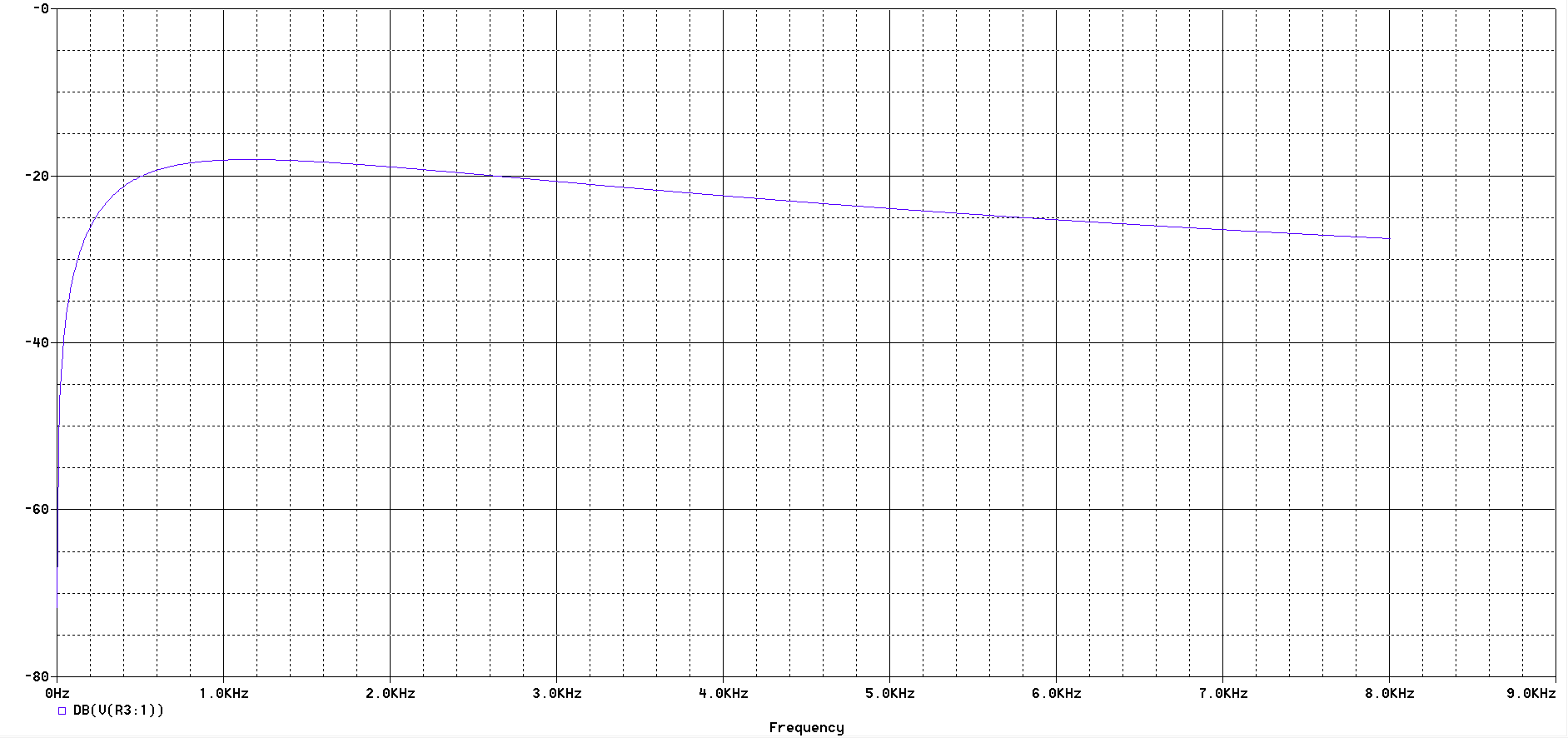


Figure 23: 2nd Order Band Pass Filter Simulation Response

### 6th Order Band Pass Filter

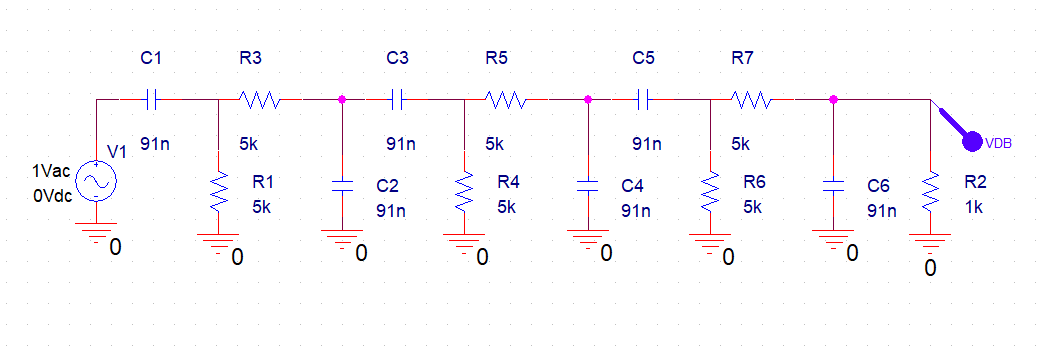


Figure 24: 6th Order Band Pass Filter Simulation Schematic

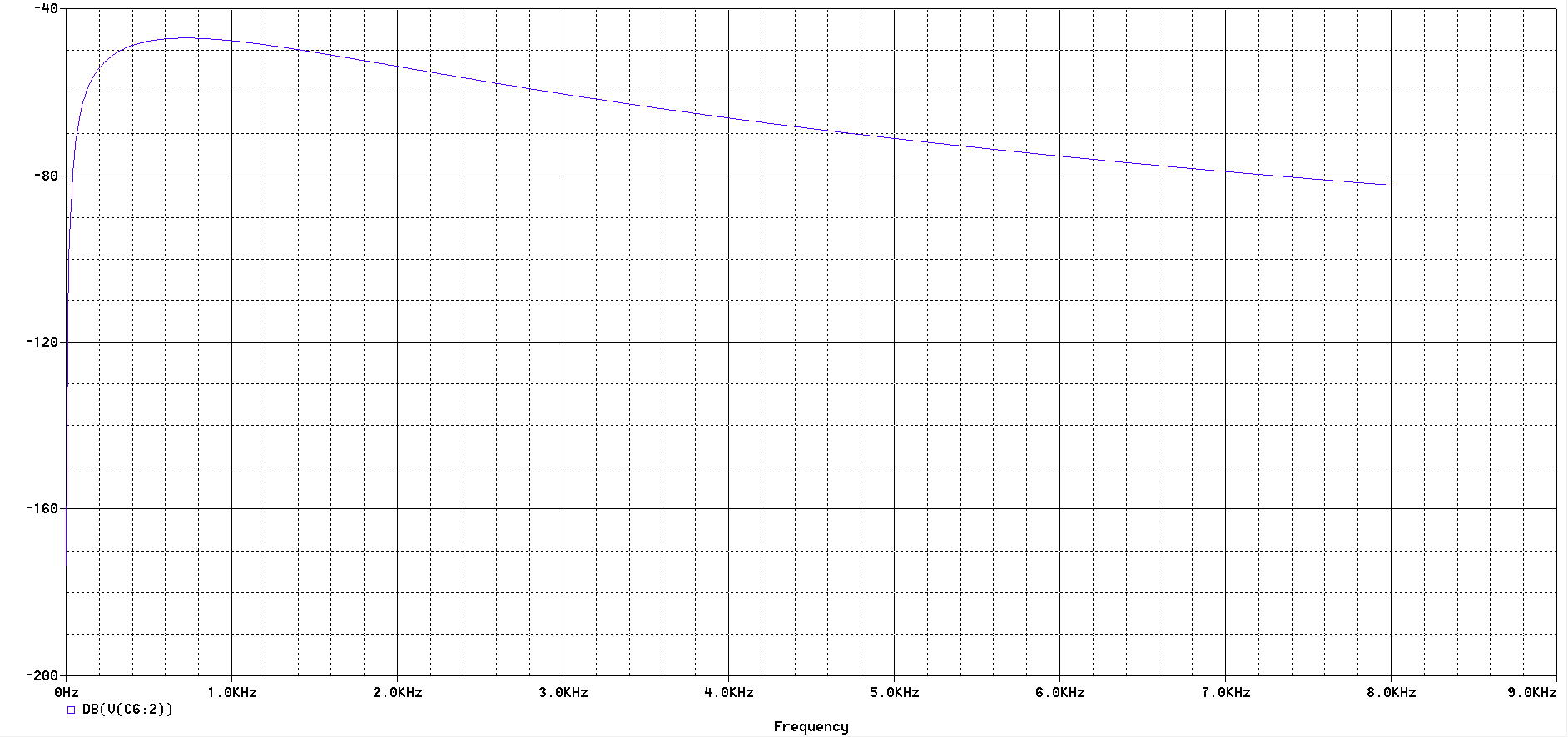


Figure 25: 6th Order Band Pass Filter Simulation Response

### 60 Hz Notch Filter

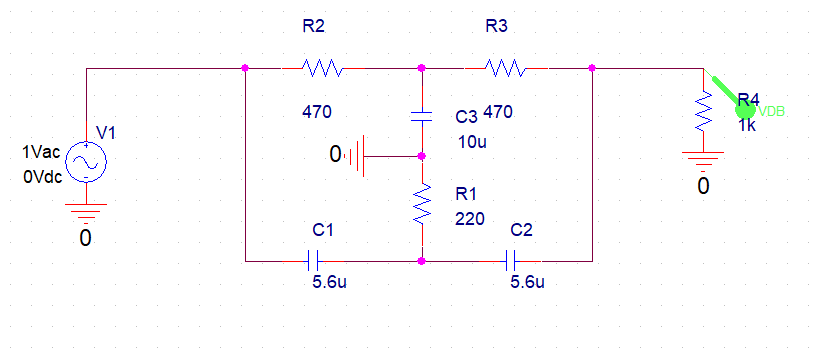


Figure 26: 60 Hz Notch Filter Simulation Schematic

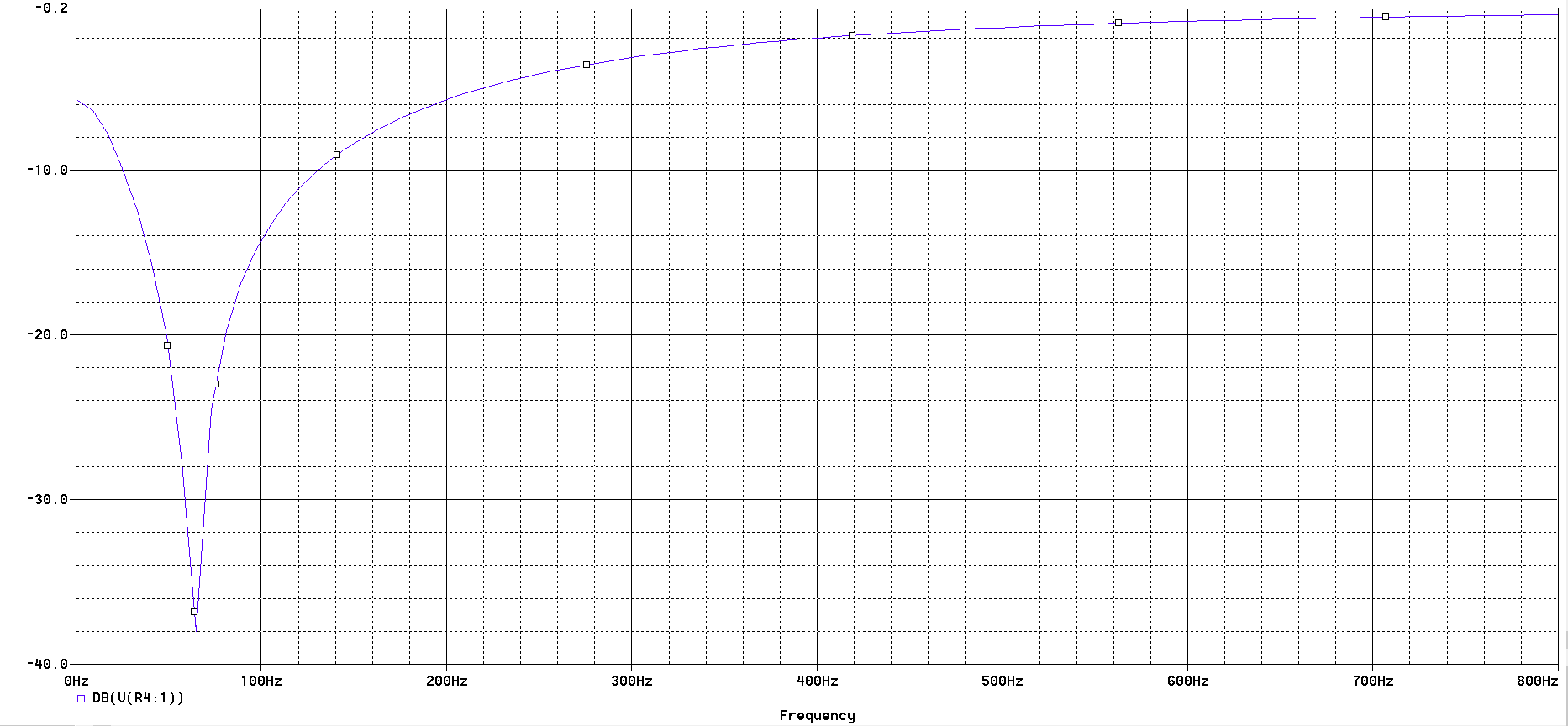


Figure 27: 60 Hz Notch Filter Simulation Response

## 6.4 Appendix IV: Hardware Specifications

Max Total Sensor Current: 230 mA

Sensor Applied Voltage: 3.3V

### Sensor Setup:

#### Hardware Setup:

##### Analog Sensor:



Figure 28: Analog Sensor Pin Setup

Pin 1: VCC

Pin 2: GND

Pin 3: DATA

##### I2C Sensor:



Figure 29: I2C Sensor Pin Setup

Pin 1: I2CSDA

Pin 2: VCC

Pin 3: GND

Pin 4: I2CSCL

Note: The logic voltage level must be 3.3V.

##### SPI Sensor:



Figure 30: SPI Sensor Pin Setup

Pin 1: SSIOCLK

Pin 2: SSIOTX

Pin 3: SSIORX

Pin 4: VCC

Pin 5: SSIOFSS

Pin 6: GND

Note: The logic voltage level must be 3.3V.

#### Firmware Setup:

For digital sensors, new drivers must be written to collect data from the new sensor. Follow the method described in the sensor’s documentation to write functions to collect data and check whether an error has occurred. Use the functions provided in the I2C driver found in sdmay18-31/src/firmware/src/drivers/ for all data transfers between the microcontroller and the sensor. This is a FreeRTOS driver, so it must be ran inside a FreeRTOS task. After the sensor driver has been tested, the sensor must be added to the “sensors\_e” enumeration which can be found in sdmay18-31/src/firmware/src/drivers/shared\_defs.h. Last, the data collection function created in the sensor driver will need to be added to the digital sensors task. Add a case to the switch in the task “digital\_sensor\_task(void\* params)” located in sdmay18-31/src/firmware/src/digital\_sensor\_task.c. The case should collect data from the sensor and place it in the data queue. Use the IMU sensor cases as examples of how to correctly collect and queue data samples.

### Tunable Filter Measurement Method

#### Setup:

1. Turn on Multimeter, Oscilloscope, and function generator.

2. Calibrate oscilloscope probe by connecting ground to ground on face of oscilloscope, and probe to test 2. Tune the oscilloscope probe until it looks like a square wave using the blue screw driver on the screw at the base of the probe.

3. Set function generator to 1.9Vpp, change offset to 1.65V, change output mode to High Z.

4. Set Multimeter to VAC measurement

5. Connect grounds of multimeter, oscilloscope, and function generator to the board ground.

6. Connect function generator positive lead to sensor input

7. Connect oscilloscope probe and multimeter positive input to circuit output pin on launchpad.

#### Testing (Low Pass):

1. Set potentiometer values

2. Record multimeter value in passband region (low frequency for low pass, high frequency for high pass)

3. Tune function generator frequency until multimeter records half of the value in the passband region

4. Record critical frequency

#### Testing (High Pass):

1. Set potentiometer values

2. Check on oscilloscope that DC offset is setup correctly

3. Record multimeter value in passband region (low frequency for low pass, high frequency for high pass)

4. Tune function generator frequency until multimeter records half of the value in the passband region

5. Record critical frequency

#### Testing (Band Pass):

1. Set potentiometer values

2. Check on oscilloscope that DC offset is setup correctly

3. Record multimeter value in passband region (tune frequency until you reach relative maximum)

4. Tune function generator frequency lower until multimeter records half of the value in the passband region

5. Record critical frequency

6. Tune function generator frequency higher until multimeter records half of the value in the passband region

7. Record critical frequency

8. Center frequency is the average of the two critical frequencies

### Measured Tunable Filter Characteristics

#### 1st Order Low Pass Filter

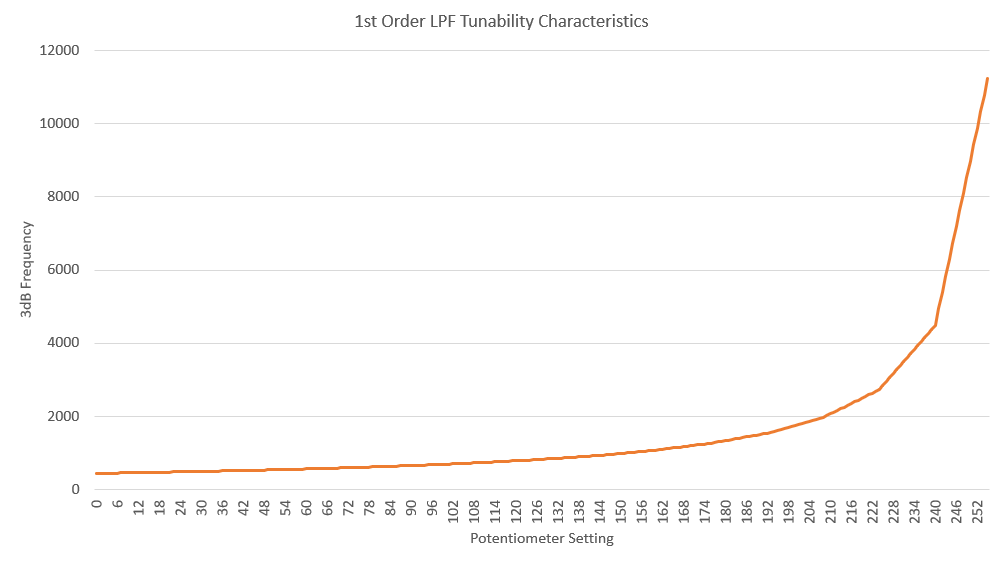


Figure 31: 1st Order Low Pass Filter Tunable Characteristics

#### 5th Order Low Pass Filter

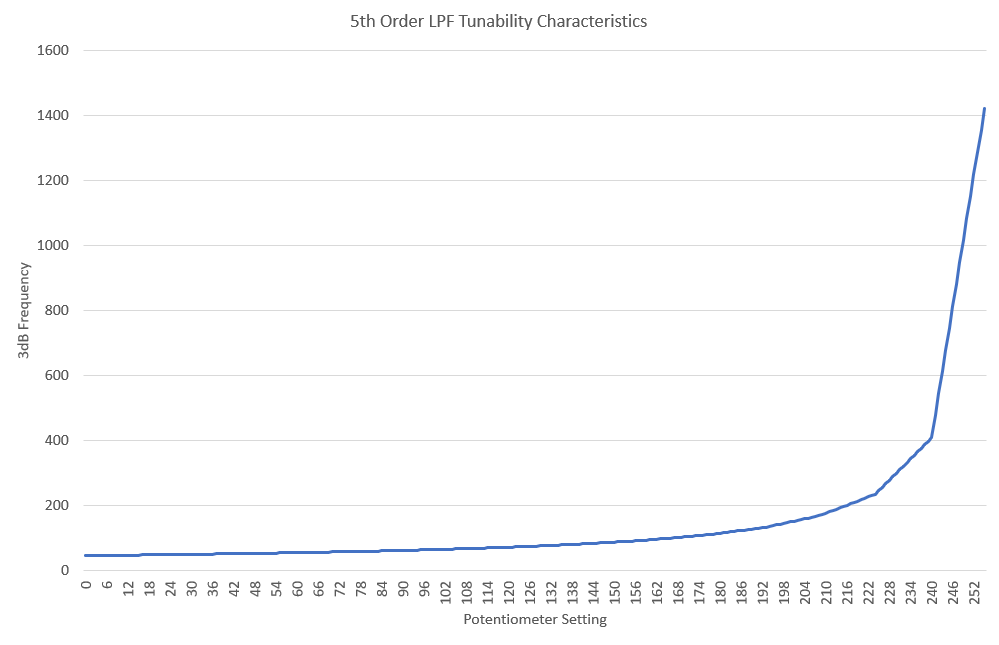


Figure 32: 5th Order Low Pass Filter Tunable Characteristics

#### 1st Order High Pass Filter

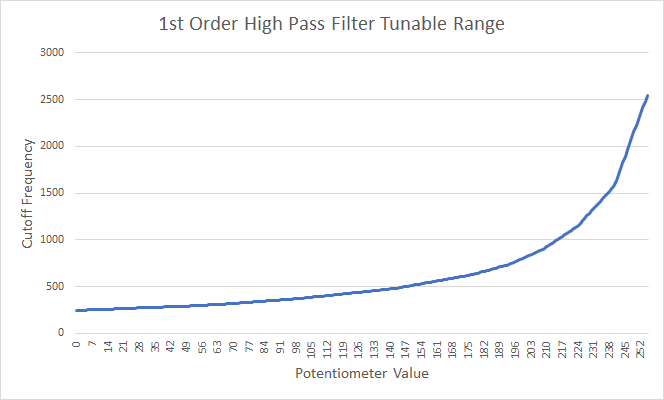


Figure 33: 1st Order High Pass Filter Tunable Characteristics (Rev 2 Measurement)

#### 5th Order High Pass Filter

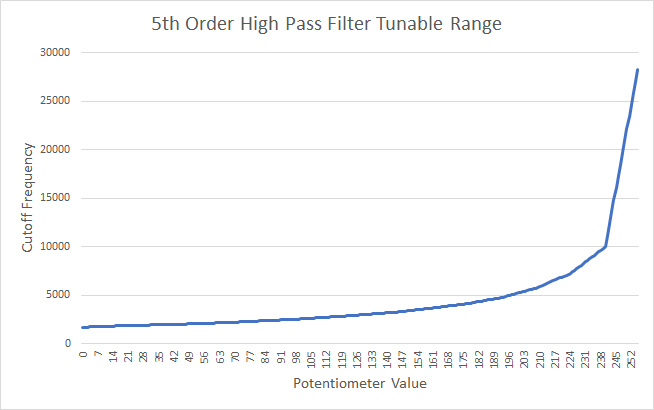


Figure 34: 5th Order High Pass Filter Tunable Characteristics (Rev 2 Measurement)

#### 2nd Order Band Pass Filter

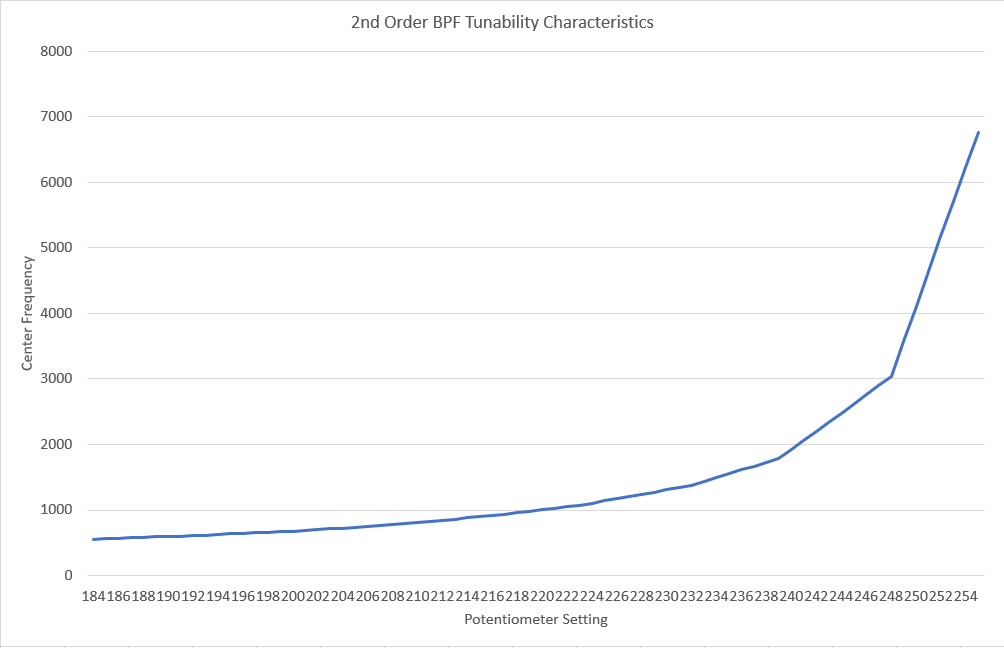


Figure 35: 2nd Order Band Pass Filter Tunable Characteristics

#### 6th Order Band Pass Filter

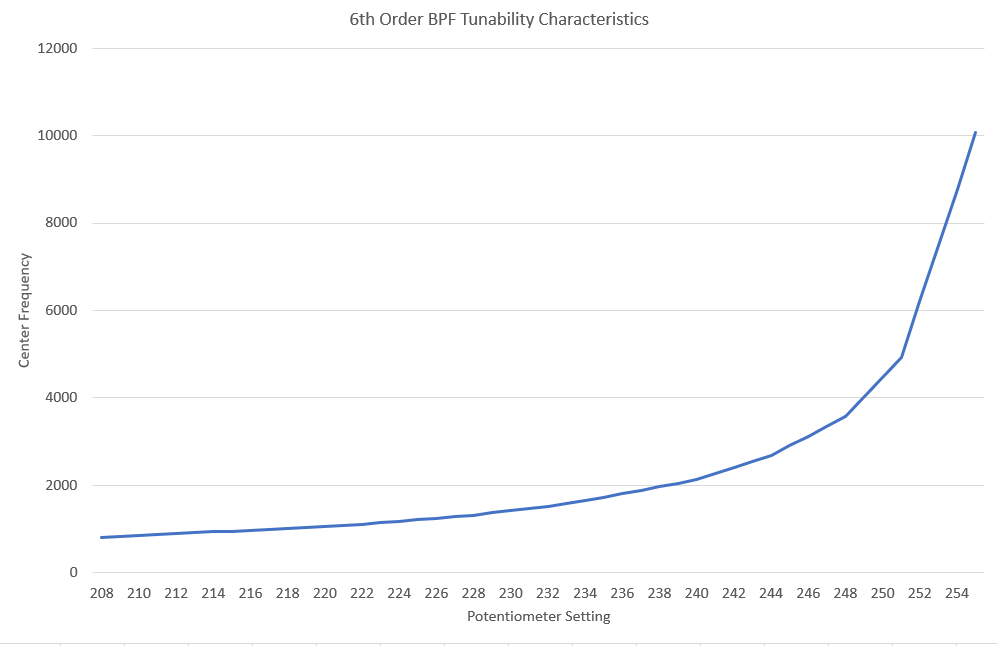


Figure 36: 6th Order Band Pass Filter Tunable Characteristics

### Firmware Testing Method

#### Muxes

##### ADC 0 Muxes/Capture

Configure the ADC muxes for following filters. Verify using the oscilloscope and the ADC capture firmware.

* 60 Hz Notch Filter
* 6th Order BPF
* 1st Order LPF
* 5th Order LPF
* 2nd Order BPF
* 5th Order HPF
* 1st Order HPF
* Pass Through

##### ADC 1 Muxes/Capture

Configure the ADC muxes for following filters. Verify using the oscilloscope and the ADC capture firmware.

* 60 Hz Notch Filter
* 6th Order BPF
* 1st Order LPF
* 5th Order LPF
* 2nd Order BPF
* 5th Order HPF
* 1st Order HPF
* Pass Through

##### ADC 2 Muxes/Capture

Configure the ADC muxes for following filters. Verify using the oscilloscope and the ADC capture firmware.

* 60 Hz Notch Filter
* 6th Order BPF
* 1st Order LPF
* 5th Order LPF
* 2nd Order BPF
* 5th Order HPF
* 1st Order HPF
* Pass Through

##### ADC 0/1/2 Parallel Input

Verify all three sensor inputs can function simultaneously. Verify with the oscilloscope and the ADC capture firmware.

* ADC 0/1
* ADC 0/1/2

##### Potentiometer Mux/Programming

Program the potentiometers for the following filters. The firmware verifies that the wiper values are written correctly.

* 6th Order BPF
* 1st Order LPF
* 5th Order LPF
* 2nd Order BPF
* 5th Order HPF
* 1st Order HPF

#### Serial Communication Interfaces

##### I2C

Send/Receive an I2C command and verify that the information is transmitted using a logic analyzer.

* Send
* Receive

##### SPI

Send/Receive an SPI command and verify that the information is transmitted using a logic analyzer.

* Send
* Receive

### Filter Circuit Diagrams

#### 1st Order Low Pass Filter

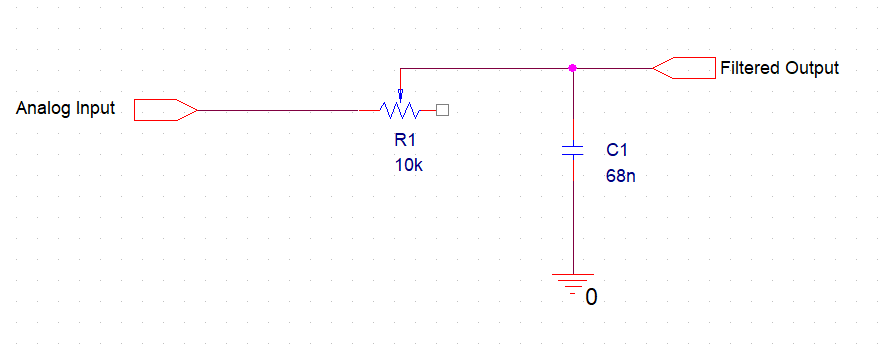


Figure 37: 1st Order Low Pass Filter Schematic

#### 5th Order Low Pass Filter

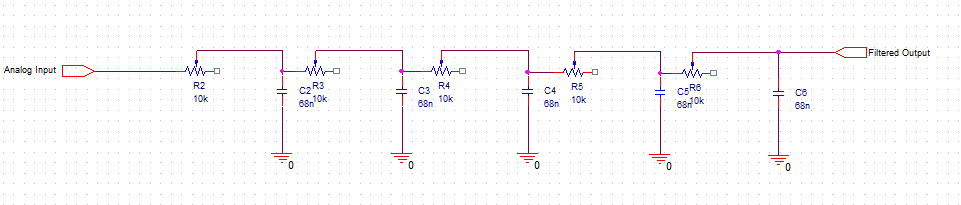


Figure 38: 5th Order Low Pass Filter Schematic

#### 1st Order High Pass Filter

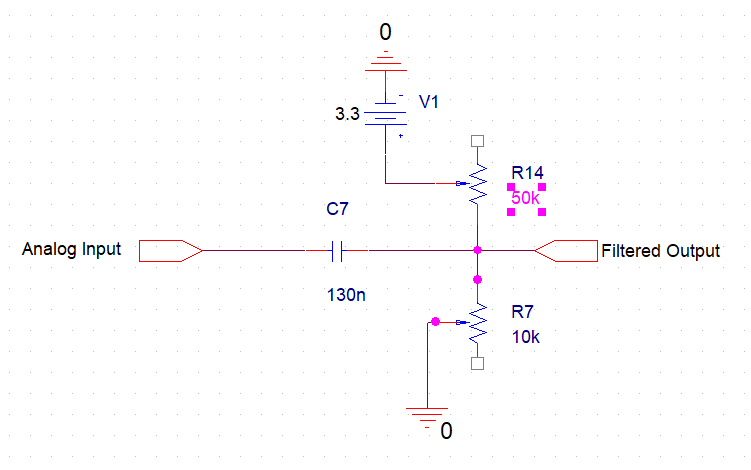


Figure 39: 1st Order High Pass Filter Schematic

#### 5th Order High Pass Filter

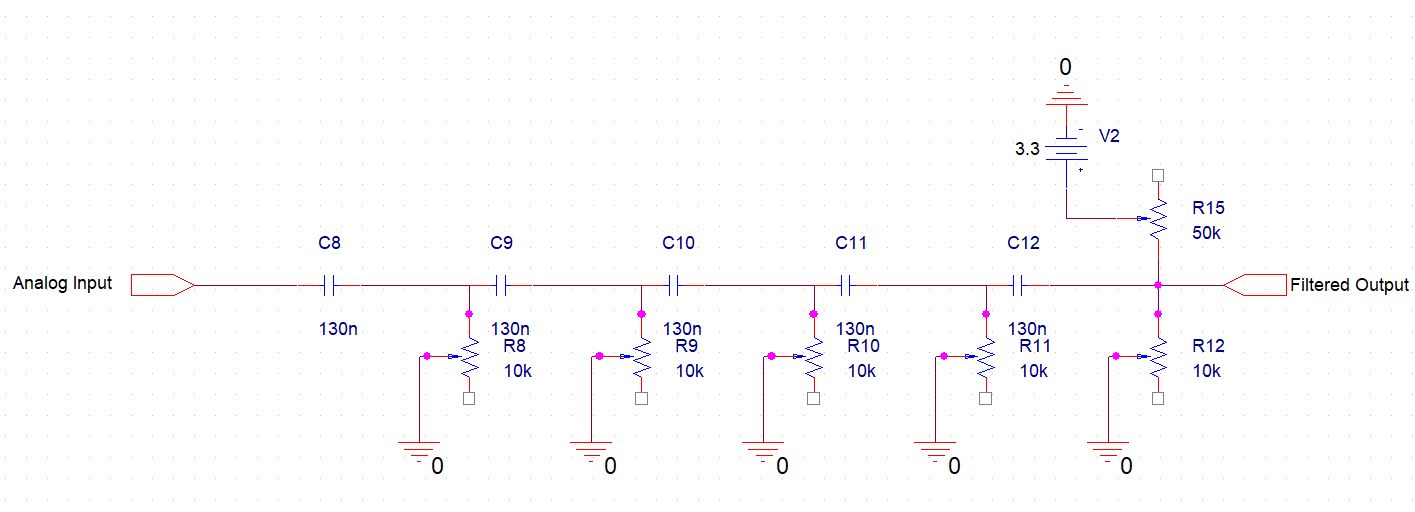


Figure 40: 5th Order High Pass Filter Schematic

#### 2nd Order Band Pass Filter

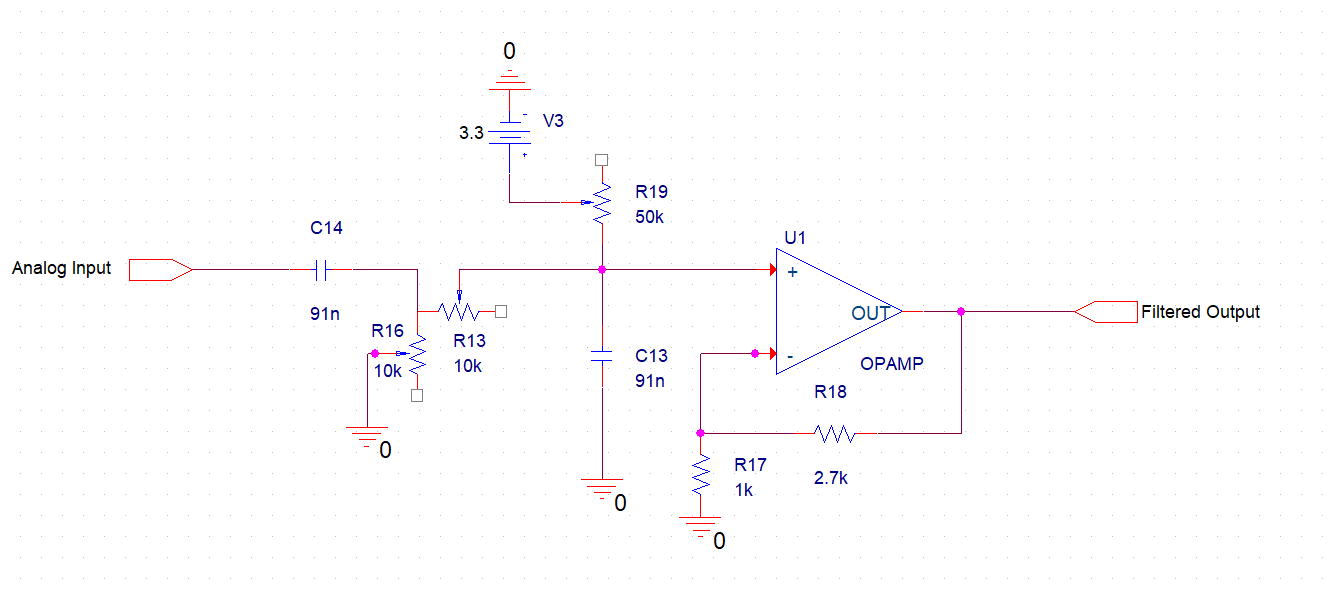


Figure 41: 2nd Order Band Pass Filter Schematic

#### 6th Order Band Pass Filter

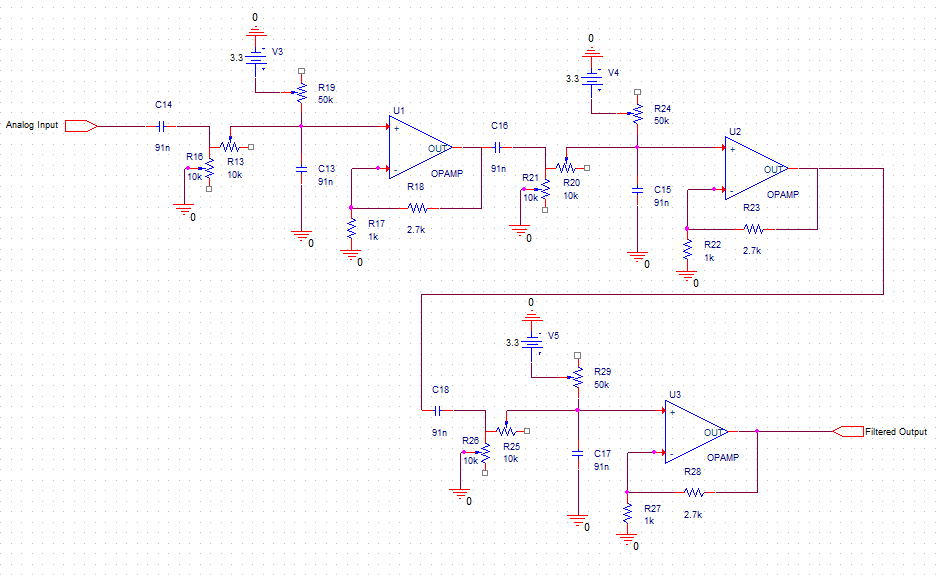


Figure 42: 6th Order Band Pass Filter Schematic

#### 60 Hz Notch Filter

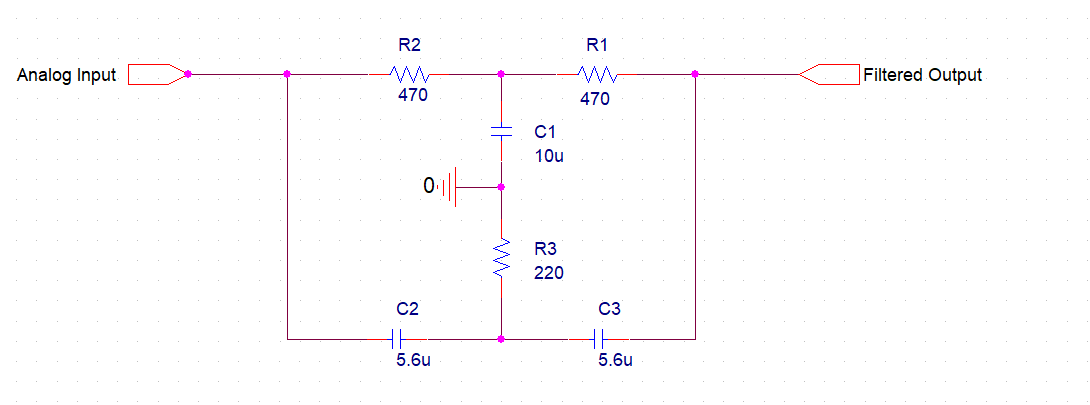


Figure 43: 60 Hz Notch Filter Schematic

#### Passthrough

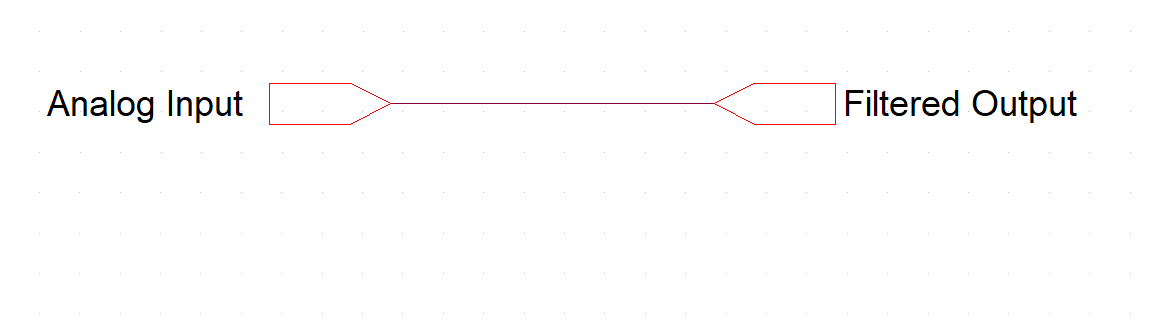


Figure 44: Passthrough Filter Schematic

## 6.5 Appendix V: Lab Design Outlines

### Synthesis of Sinusoidal Signals using Tuning Forks

Tuning forks are physical systems that generate sinusoidal signals. When a tuning fork is exposed to vibration, it disturbs nearby air molecules, creating regions of higher-than-normal pressure (called compressions) and regions of lower-than-normal pressure (called rarefactions).

#### Pre-Lab:

Read the lab handout and section 2.7 in the textbook on tuning forks. Consider the following question. This lab uses tuning forks with frequencies ranging from 128 Hz to 4000 Hz. Assuming that the stiffness of the tuning forks is the same, comment on the mass of the tuning forks as the frequency increases. Verify your hypothesis when you get to the lab.

#### 1 Overview

In this lab, you will be using the microphone sensor connected to the CyDAQ and your computer to digitize a sinusoidal signal from different tuning forks. The microphone will convert the sound signal generated by the tuning fork to an electrical signal, which in turn is converted to a sequence of numbers stored in a digital file which can be displayed on computer screen as a sinusoidal curve. Characteristics of the sound wave, such as its period T and frequency can be determined from this curve. Knowing the waves period, its frequency f is easily computed using the formula:  
 f = 1 /T

#### 2 Post-Collection Processing

In this section, you will write an M-File function that takes input data signal, the sampling rate, and related time vector at a specific sampling rate and stores it as an output file in .wav format. The input signal will be the tuning fork signal when you strike it and expose to vibration.

#### 3 Labeling Plots in MATLAB

It is very important to be able to label all plots and graphs turned in with your lab reports. Basic functions for labeling axes and putting titles on your graphs are given in this section.  
xlabel, ylabel: these commands put a text label on the x and y axes of a plot. For example, to label the x- axis as time in msec: xlabel(‘time (msec)’); The characters within the quotes are plotted on the active graph.  
title: this command puts a title above the graph. For example, to label a plot ‘Time vs. Tuning Fork Response’, use: title(‘Time vs. Tuning Fork Response’);  
axis: this command sets the beginning and end values for the graph axes. For example, if the time should start at 2 sec and end at 5 sec and the recorded signal should range between 0 and 10 volts, use the command: axis([2, 5, 0 10]); axis([xmin xmax ymin ymax]).

#### 4 Frequency view of signals in Matlab

Signals can also be viewed in terms of frequency as well as time. The Fourier Transform is used to show what frequencies are present in a signal. We will cover exactly how this works in class in the next few weeks. The command below plots the frequency spectrum of a signal where Fs is the sampling frequency:  
freqz(tuningfork1,[1],Fs);

#### 5 Lab Procedure Signal Collection

Part 1:  
A) Collect audio signals. Using the microphone sensor connected to ADC0 and collect signals for two different tuning forks or generated tones at frequencies at or under 2028 Hz using a sampling frequency of 4056 samples/sec. Do not use any filters. Repeat at 8000 samples/sec sampling frequency. Note: Save the files as .mat and give each signal a different name to avoid overwriting.  
Next, collect data at 8000 samples/sec using the 6th Order Bandpass filter centered close to your tone/tuning fork’s frequency.

B) Measure the frequency of these signals using a plot of the signal. Include the plots that you used in your lab report. Be sure to label your graph and all the time units using the methods given in section 3 above. Show how you measured the signal. To have the proper time index on the plot, the sampling interval (the space between samples) used in this signal must be calculated (Ts=1/Fs). The time vector must then be scaled to give the proper time values.

C) Compare the measured frequency with that on the tuning fork. What are possible sources of error? Use the zoom and cursor functions in Matlab figures to improve your observations.

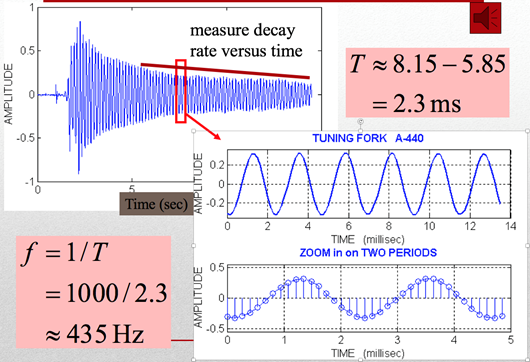


Figure 45: Intro Lab Tuning Fork

D) Measure how the energy in the signal falls off over time for each of the tuning forks. Provide an annotated figure that shows how you measured the signal. How does the bandpass filtered signal compare to the unfiltered signal, sampled at the same rate?

E) Compute the magnitude spectrum of the signals in step A using the method given in section 4 of this lab. Verify that the peaks occur at the frequency of the tuning fork. Turn in your plots. (Note: If your signal is not close, then your time window for collection is probably too long.) What is the amplitude in decibels of the signal in each case? Compare the tuning fork frequency estimated by the frequency spectrum with the one estimated using the time signal.

F) Comment on the other signals present in the spectrum. Why don’t you see a nice clean delta function like in the notes?

G) Search on your browser for Matlab’s butter and buttord functions. Could these help? Implement a bandpass butterworth filter with the following parameters:  
 Let f = your tuning fork/tone frequency in Hz   
Passband Frequencies: f - 30 Hz to f + 30 Hz  
Stopband Frequencies: f – 60 Hz and f + 60 Hz  
Passband Ripple: 3 dB  
Stopband Ripple: 35 dB  
Apply your filter your filter using Matlab’s filter function:  
 filtered\_signal = filter(b, a, data);  
Apply freqz and compare it to the other plots. How does it compare? Better? Worse?

H) Give some reasons for why the CyDAQ has a max sampling rate? What are some system level limitations that prevent this device from being faster? Note, if you used the DAQ in the computer, you would be able to sample around 11,025 samples/sec.  
  
  
5. Lab Report   
A) Answer all questions in section 4 of this lab and include all requested plots and calculations of the frequency and magnitude decay estimation with clearly labeled graphs. Put them in a pdf file and submit electronically using the appropriate lab account. Fill in the table below as part of your report:

|  |  |  |  |
| --- | --- | --- | --- |
| Signal Source | Frequency Estimate from Time Plot | Magnitude Decay Rate V/sec | Frequency Estimate from frequency plot |
| Tuning fork labeled 426.6 at 11,025 samp/sec | 430 Hz | .02 | 428 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Figure 46: Intro Lab Tuning Fork Table

B) Include all your MATLAB code in an appendix of your report. You do not need to include the tuning fork signals.

### 

### Fourier Transform Lab:

#### Introduction

You are working for a struggling smart watch company. Your boss, fearing for his job, does a market survey and finds that consumers are buying smart watches with activity tracking features like pulse rate monitoring. You are tasked with developing an algorithm to calculate pulse rate from a plethysmography waveform. You will be exploring the Fourier transform and then using what you’ve learned to develop an algorithm to find the pulse rate.

#### Background

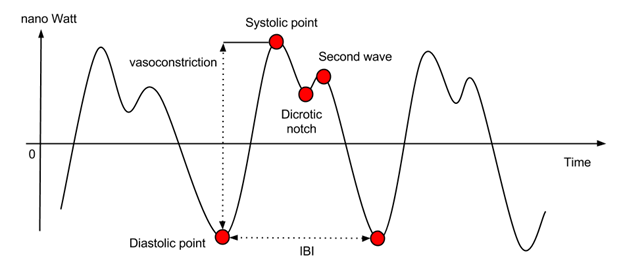


Figure 47:Pulse Lab PPG Signal

<https://support.empatica.com/hc/en-us/article_attachments/202917049/PPG2.png>

A plethysmogram (PPG) is a signal that measures the volume of blood traveling through a given part of the body. When a person’s heart beats the volume of blood changes as blood is pumped through the body creating a pseudo periodic signal. There are two phases of the cardiac cycle: the diastole phase and systole phase. The diastole phase is a period of low pressure when the blood flows into the heart. These low pressure points correspond to the local minima of the PPG signal (diastolic point and dicrotic notch shown in Figure 1). The systole phase is a period of high pressure when the blood flows out of the heart. These high pressure points correspond to the local maxima of the PPG signal (systolic point and second wave shown in Figure 1). The inter-beat interval (IBI) is used to calculate the pulse rate.

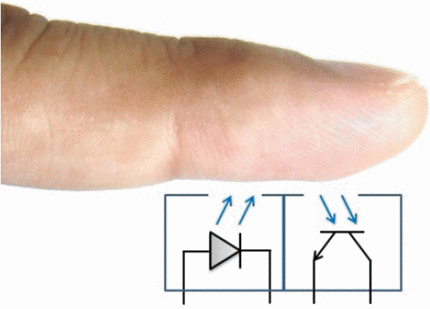


Figure 48:Pulse Lab PPG Sensor Functionality

https://ieeexplore.ieee.org/ielx7/7036195/7042935/7043525/html/img/7043525-fig-1-large.gif

The PPG sensor that will be used in lab has an LED emitter and receiver to measure the volume of blood in the finger. Blood absorbs certain wavelengths of light, so the volume of blood in the finger corresponds to the amount of light reflected out of the finger.

#### Data Collection

Since the PPG signal will be measured on a human, assumptions can be made about frequency of signal. Assuming that a human’s pulse rate will be between 30 and 300 beats per minute (BPM), calculate the cutoff frequencies for a band pass filter that will remove unwanted frequency components. Show all your work. Note: The creator of the PPG sensor already implemented an analog low pass filter so don’t worry about filtering the signal yet.

Again, using the assumption that the maximum pulse rate is 300 BPM, what would be the minimum sampling rate required to accurately calculate one’s pulse rate? Show all your work.

Now collect data from the PPG sensor using a sample rate of 100 Hz so that you can clearly see the different characteristics of the signal. Be sure to stand still when collecting the data. Note: Everyone’s PPG signal looks different so don’t worry about making the signal look exactly like Figure 1, but do make sure that the signal is periodic.

#### Pulse Rule Calculation

Once you’ve collected a clean PPG signal, import the data into MATLAB. Calculate and plot the Fourier transform (FT) to view the signal’s frequency spectrum using the fft function. Remember that the FT calculates a complex number, so use abs to get the magnitude of signal. Why is the fft symmetrical? Plot only the first half of the fft and include it in your report. Be sure to calculate the frequency on the x-axis.

Now calculate the pulse rate by finding the frequency that contains the most energy. Hint: use where is the index at which has the max value. What is the calculated pulse rate? Is this correct?

Remove the DC component from the fft by setting where is the FT of the PPG signal. Calculate the pulse rate again. Is it correct now?

**Be sure to include all MATLAB code used as well as the answers to all the above questions in your report.**

### VAD Lab Outline:

In this lab, students will use both the CyDAQ device and MATLAB in order to implement a simple form of Voice Activity Detection (or VAD) in a noisy environment.

#### Data Collection:

* Microphone Sensor
* Utilize on-board sensor to filter out spectral content outside of typical speaking range (~ 80Hz to 260Hz)
  + Thoughts: Why do we filter at this range? What benefit does this offer in a real life application?

#### MATLAB:

* Determine appropriate signal magnitude for voice detection
* Implement thresholding at this value (a)
  + Thoughts: Why do we care about thresholding
* Import Voice recording as ‘data’ vector
* Write a function taking arguments (data, a), output signal y
  + If magnitude>=a for some number of consecutive samples,
    - Display to command window “Voice Activity Detected”
    - Signal passes as expected in y
  + Else (if magnitude<a for some number of consecutive samples)
    - Sample is replaced by zero
    - Display “No Voice Activity Detected”
* Harder Test: Generate random noise vector (randn(length(data))), add this to our recorded voice data
  + Thoughts: What are we modeling here?
* Try various scaling values for noise. What happens as noise increases? Why do we care? Can we do anything in software to lessen issues caused by artificial noise?

#### Report Extras:

What sorts of applications could this be used for in real life?

#### Bonus Vocab:

* FEC (Front End Clipping): clipping introduced in passing from noise to speech activity;
* MSC (Mid Speech Clipping): clipping due to speech misclassified as noise;
* OVER: noise interpreted as speech due to the VAD flag remaining active in passing from speech activity to noise;
* NDS (Noise Detected as Speech): noise interpreted as speech within a silence period.

### Noisy IR Sensor Lab:

#### Abstract:

#### 

FIR Filtering of discrete time data can be performed using convolution. By applying the idea of discrete time convolution, students will be able to construct various filters to be applied to a noisy incoming signal. Using an IR sensor as our signal source, we can look at applying the following filters through convolution in MATLAB:

* Moving average
* First difference filter

Additionally we can look at applying digital FIR filters to the input data using the interface to be provided by our device, to create the following filters:

* Low-Pass Filter
* High-Pass Filter

#### Introduction:

In this lab, we will experiment with creating FIR filters, and applying them to a real-life input. In this lab, we consider the case of a Mars rover. A rover is an automated motor vehicle which is capable unmanned exploration upon its arrival on Mars. In order to make this exploration possible, the rover needs to be capable of taking in all sorts of input data from the surrounding environment. One particularly important type of data the rover needs to be able to bring in would be whether there are obstructions or other objects nearby, and ideally how far away these objects are from the rover.  
  
Using the inexpensive SHARP GPY0A21YK IR sensor appears to be a cost effective way of implementing this functionality. Clearly, using inexpensive components in an expensive project such as a Mars rover will introduce noise to our measurements. Luckily, we can filter the incoming data such that it is more useful for determining where objects are actually located in relation to the rover.In this lab, you are asked to explore the application of discrete-time convolution to the development of FIR filters in sampled data, as well as to design digital filters using the provided user interface, and apply these to the incoming data.

#### Process:

For this experiment, students are expected to do the following:

1. Configure the device to bring in IR Sensor data
2. Record data
3. Construct FIR filters in MATLAB, which are then applied to our recorded data
4. Construct FIR filters on the device, which can be applied to our recorded data

#### Step 1: Configuration

--- This section will provide instructions on how to set our device up for use with the IR Proximity sensor ---

#### Step 2: Data Collection

--- This section will provide students with guidance on how to collect the data used for this lab, detailing which distances to measure from, which sort of surfaces work well for more accurate measurements, etc ---

Some reflection questions here: Have the student generate a plot with the raw data. What do they notice? Have the student reflect on the noise contained in their recording. Where does this come from? What are possible sources of error? What could potentially be done differently to reduce the amount of noise that is present in the reading?

#### Step 3: First, we’ll handle construction of FIR filters in MATLAB.

We can call y[n] = (b\_0)(x[n])+(b\_1)(x[n-1])+(b\_2)(x[n-2])+...  
Or, \sum\_0^N=(b\_i)(x[n-i]). We know that this is just discrete convolution.  
We can apply a moving average filter in matlab:

>> x = input data;

>> h= ones(1,N)/N;

>> conv(x,h);

Plot this resulting data. Students reflect on the result. What happens as N becomes very large? Clearly our result becomes more robust to noise. What are drawbacks of using high numbers for N? Have students plot for N = {1, 3, 7, 21, 200}. Have students comment on their observations.  
  
What other sorts of filters can be implemented with convolution? Consider a first difference filter, h=[1, -1]. This finds the difference between two consecutive data points. Have students apply this to the recorded data, plot the result, and describe their findings. Unlike the previous filter, this filter is not very robust to noise. What could be some real life applications of a filter like this?

#### Step 4: Implementing digital filters on the device.

Using Matlabs designfilt command, have students generate various filters, and input the filter coefficients into the digital input filters on the device. Were the results what was initially expected? Why do you or do you not think so? Which filter type seems to be the best for filtering noisy measurements?

Include the following in the report:

All answers to questions,

All plots requested.

Elaboration on all digital filters created with designfilt

Thoughtful reflection on the real-life uses and applications of FIR filtering for measurements.

### Automation System Lab:

#### Intro:

Within the past several years, the prevalence of “smart” homes and the interconnection between networked devices has been on the rise. Considering this, it should come as no surprise that signal processing is playing an increasing role in the day-to-day life of the average consumer. In this lab, we’ll work with both the CyDAQ and the MATLAB tool Simulink to develop and simulate a home automation system.

#### Background:

We would like to be able to minimize our spending on running the air conditioner in a home. By leaving the air conditioner running while no one is using it, we could potentially be losing out on a couple hundred dollars per summer. While this is certainly not great for an average sized home, it becomes even worse as we consider the implications for large office buildings with several air conditioned rooms.

You have been tasked with developing a simple system to determine when a room is empty (and thus, should not be running the AC). As a design constraint, you are able only to use a light sensor to determine this.  
  
As such, we wish to construct a system that will bring in readings from a light sensor, determine whether we are seeing natural or artificial light, and decide what actions need to be taken pertaining to cooling the room.

#### Setup:

Using light sensor

* Develop a method to determine which sensor values correspond to artificial light readings
* Identify when lights are on or off
* Analyze frequency domain results to see artificial light frequency characteristics

Next, we consider the possibility of natural light coming in through a window or skylight.

Using simulink:

* Pass in readings from the light sensor
* Corrupt with additive gaussian noise (modeling natural light)
* Re-evaluate system to ensure accuracy even in presence of modeled natural lighting
* Analyze frequency domain results using FFT