FeAtHEr-Cm

Release 0.1.1

BoB LeSuer (BoBthechemist)

Nov 01, 2021

CONTENTS

FeAtHEr-Cm is an alternative approach to teaching the traditional 2nd semester of analytical chemistry: instrumental methods. The focus is on problem-based learning that involves the design and implementation of rudimentary instruments from the core areas of electrochemistry, spectroscopy and chromatography. In addition to content knowledge, students acquire key technological process skills - physical computing with microcontrollers and software programming - that are transferable to many professional-development pathways.

CHAPTER

ONE

START HERE

Welcome. If you are here, you were either invited or spend an awful lot of time randomly clicking links on the internet...

`FeAtHEr-Cm` is a platform for building a better understanding of scientific instrumentation. It is my attempt at developing what I call an *Instrumental Approach to Chemical Analysis.*[*0] Traditionally, the second semester of analytical chemistry covers instrumentation used in the discipline. Recent advances in technology, electronics and programming allow us to explore how scientific measurements are made by *designing our own instruments* and using them to explore chemical systems. FeAtHEr-Cm helps us do that.

1.1 Why FeAtHEr-Cm?

When exploring platforms that could be useful for introducing chemists it scientific instrumentation design, I wanted something that was relatively inexpensive, easy to operate, established, and supported by a larger community. The feather platform by Adafruit Industries checks all of those boxes *plus* provides a cross-compatible platform with a vast array of tools already available. The flexibility comes at a cost, however, and that is the array of choices that can be made is a bit overwhelming. To bring some focus to the project, I have made two design decisions:

- The microcontroller board that will serve as the brains of any FeAtHEr-Cm instrument is the M4 Express and there is no intention to make activities compatible with other microcontrollers.
- All programming will be done in Circuit Python

The primary reason for selecting the M4 is that this microcontroller contains two 12-bit digital-to-analog outputs, which simplifies many designs (not least of which is the potentiostat). The analog inputs are also 12 bit which is a very nice feature in education-based instrumentation. By limiting software design to circuit python we not only decrease the programming learning curve (only one language is needed to control and communicate with the instrument) but it increases the accessibility of the instrument by being a plug and play option. Note that a big downside to the M4 is that the analog to digital converter (ADC) isn't as good as it needs to be, which is why *the instrument you build is not suitable for research applications*. There are tweaks made (and being made) to address the problems so that the potentiostat remains easy to build and configure while still serving as a meaningful learning platform.

As for that odd capitalization, it is something I've done for a long time. I'm not alone and there are many chemists who do this.

[0] Yes, this title is homage to Ewing's 1954 text and no, it is not at all suggestive that I am (or ever will be) comparable to Ewing as a scientist.

1.2 What is a Feather?

From the Adafruit website:

> Feather is a flexible and powerful family of microcontroller main-boards (Feathers) and daughter-boards (Wings) with a wide range of capabilities. They can be plugged into a breadboard for quick prototyping, have built-in battery connectors for your on-the-go projects, and most have built-in lipoly chargers!

We use the Feather platform because it's cheap, powerful and flexible.

1.2.1 What about *Mathematica*?

You'll note that while I mentioned above that all programming will be done using Python, there is a fair amount of Mathematica code floating around these pages. I've been using Mathematica for many years now, and it is a language with which I'm comfortable. Therefore, when I need to do something quickly, I'll do it using that language. Additionally, there's a very good chance that students at an American college or university have access to Mathematica, and even if you don't, you can gain access by running the program on a Raspberry Pi (which includes a free for educational use license). Despite being a closed source package, Mathematica is fairly accessible and serves as a useful tool for some high-order programming, data manipulation and visualization. This project will not rely on Mathematica, but examples written in this language will be used to demonstrate how one might integrate these projects into a larger system.

1.2.2 Want to dive in?

If you are excited about this project and want to start exploring right away, then I suggest purchasing an M4 Feather along with Adafruit's parts pal. That will give you nearly all the parts you need, and many that you'll find fun to play with. The only additional components I suggest would be alligator clips and a quad op amp - the MCP6004 - which isn't sold by Adafruit but you can find it at other vendors. I purchase from Newark but you might want to try Digikey as they are also a reseller for Adafruit products and you might be able to get all of the components from one vendor.

1.3 A work in progress

This project is in the early stages of development. If you are visiting, you will likely find links that do not work, documentation that seems to require a significant amount of prerequisite knowledge, and many outlines and placeholders. My intention for a *version 1.0* of this project is an instrument with activities for each of the three main areas of chemical instrumentation (spectroscopy, electrochemistry and chromatography). I am currently working on the electrochemistry modules.

1.4 An open, collaborative effort

I am building this project using github and read the docs for a few important reasons.

- Both platforms provide open access to the content, so that others may view, use and modify what is presented here
- Both platforms work nicely together, allowing me to generate both software and instructional content in one location and have it display in a pretty format without much input on my part.
- This approach allows community members to contribute to the project either as content generators or content reviewers.

I imagine that users of the content will find typographical and content errors and wish to propose corrections; others may have suggestions on how to better present content. With Github, those users should be able to make those requests in a documented, public fashion. If the changes are accepted, they get pushed immediately to the project.

1.5 What to explore right now

The current area of activity is building the potentiostat, creating software for controlling it and developing activities for exploring how a potentiostat is built and operated. Have a look first at *Current operation* which shows the data that can be acquired with the potentiostat and how it is analyzed. Then visit *The Voltage Divider* and the 2 or three sections that follow in order to get a feel for how the simulation software is used to introduce the electronics concepts needed to build the potentiostat. Lastly, you can look at *Building the potentiostat* which presents the schematic among other things. All other sections will be in various stats of disarray.

CHAPTER

TWO

TIPS & TRICKS

Here you'll find various bits of information that are useful for working with the FeAtHEr-Cm system.

2.1 Obtaining the source code

The source code is maintained on the FeAtHEr-Cm Github site. On the right hand side of this page, you'll find a link to the releases with the latest one displayed. If you want that one (chances are, you do) go ahead and click on the link, read the release information and download the source code zip file. It is also possible to click this link to go directly to the latest release.

The zip file contains more information than you need (such as the raw files used to construct the documentation website). You will want the material found in the *source* directory. Copy all of the contents of this directory *except for client.py* onto your microcontroller. Note that this process will overwrite any code you currently have in *cody.py* so store that in some location if it is necessary. The *client.py* code is for your computer and allows for communication between the instrument and you.

An example of communicating is in the Example Data section of the Potentiostat module.

CHAPTER

THREE

THE FEATHER-CM TEXTBOOK

3.1 Roadmap for version 1.0

Initially, content is not going to be self-standing and will leverage other textbooks that adequately cover theory and pre-requisite material. Texts such as Skoog's *Principles of Instrumental Analysis* and Harris' *Quantitative Chemical Analysis* are sufficiently ubiquitous that a student will likely have access to one or both of those texts. Harvey's *Analytical Chemistry 2.1* provides adequate background material on chemical theory and is freely accessible. These texts will be referenced when background material is necessary. Rather than reinvent the wheel, there will be little attempt to cover material that is already available in Harvey's text, so once it becomes necessary to introduce content that is presented in one of the "closed" textbooks, the focus will be on expanding on where Harvey lets off.

3.2 Building and using a potentiostat

An instrumental approach to voltammetric measurements will center on the construction of a potentiostat, which requires a basic understanding of circuits involving resistors, capacitors and operational amplifiers. Circuit analysis at the level of Kirchhoff's voltage/current laws is not necessary and will not be presented. Coverage of operational amplifiers will be limited to the two ideal principles of op amps (no current draw from the inputs and the voltage difference between the inputs is zero).

The potentiostat used in this course is built with problem-based exploration as a foundational design principle. Not only are we ignoring important electrical engineering design concepts (which are beyond the scope of this program), but the resulting potentiostat will allow individual components to be probed and analyzed. For example, the components used to define a virtual ground are also used to introduce the concept of a voltage divider and explore why voltage dividers need to be buffered with an operational amplifer.

Lastly, there is an excellent circuit simulation package available on the internet that is incorporated into the text. It allows for just-in-time exploration of concepts that are introduced and will allow students to compare actual results to theoretical results.

The expected student trajectory is as follows:

1. Electronics principles
   - Understand what a voltage divider is and how it is created using two resistors
   - Explore relationship between resistor magnitude, power consumed and output voltage
   - Analyze the effect of an external load on a voltage divider
   - Compare simulated and actual performance metrics with the bob173.
2. An overview of operational amplifiers and their utility
   - Understand that an operational amplifier can compare the relative values of the input voltages

- Explore the role of op amps in buffering an input
- Evaluate how performance of a real operation amplifier compares to the ideal behavior of drawing no current. - This section will likely be the first entry into (circuit) python programming.
- Explore how operational amplifiers can add multiple inputs together.
- Explore how operational amplifiers can magnify the intensity of a signal.

3. Combining the elements - a basic adder potentiostat

- Learn how to read a schematic
- Explore python programming needed to communicate with the bob173
- Calibrate voltage and current scales of the bob173
- Perform a cyclic voltammetry experiment and analyze results

4. Going further - improving the potentiostat

- Reducing noise
- Incorporating iR compensation
- Exploring instrument limitations
- Exploring a "new" chemical system.

3.3 Building a colorimeter

One of the most prevalent instrument design activities is the construction of a rudimentary spectrometer. Rather than provide yet another example, the proposed instrument design activities will focus on several under-represented concepts:

- Exploring how an instrument controls/operates a source and records information from a detector
- Analyze non-idealities due to instrument design (a primary example is stray light)
- Investigating signal transduction for novel data acquisition strategies (one example might be transducing detector response to an audio signal)

One design idea is to intentionally incorporate source flicker into the instrument. Then create a system that provides dual-beam capabilities and explore how effective this approach is at reducing such noise.

Concepts covered in this module will include electronics (transimpedance amplifiers), programming (digital and analog IO), deviations from Beer's Law and signal transduction. It may also be an interesting segue into 3D printing design.

3.4 Building a thermal conductivity detector

Prior to exploring chromatography instrumentation, we need to have a reasonable detector. It is possible that one could use concepts in the previous sections (spectroscopy, voltammetry) to design a detector, and that could/should be part of a future direction. Here we are interested in exploring a detector that is often described in instrumental analysis courses, the thermal conductivity detector. Designing and testing the detector will provide an opportunity to further develop our electronics understanding through the development of a Wheatstone bridge. The goal would be to develop an instrument that can, for example, analyze a binary gas sample. The approach we are revisiting was first described in J. Chem Ed. 28, 576 (1951) and it is time we revisit this approach.

3.5 Creating a rudimentary Gas Chromatograph

The previous module presented an approach to analyzing gaseous mixtures, which suggests that by adding a means to separate mixtures prior to TCD analysis should result in a functioning GC. There are DIY GC designs on the internet that use packed columns (one of them actually uses kitty litter) which might be sufficient to perform alcohol separations.

CHAPTER

FOUR

INTRODUCTION: INSTRUMENTAL APPROACH TO CHEMICAL ANALYSIS

In this introductory module, you will jump in to instrumental design by building an instrument - a papercraft spectrograph - that you connect to the camera of your phone. You'll then learn about classifying data types using the concept of *data domains* and as a result, develop a good understanding of *what is a scientific instrument* and *what is a measurement*. You then shift gears to an introductory exploration of the microcontroller used to make most of your scientific instruments along with the programming language, CircuitPython. Lastly, this module introduces some practical electronics topics such as soldering, which will allow you to put together your instrument kits.

Warning: See a problem? Have a suggestion? Please raise an issue and share your thoughts there.

4.1 A Papercraft Spectroscope

We start this module by building a papercraft spectroscope that can be used with your phone. The design was developed by Public lab and full details on the design can be found on their website. Here, you will find the basic materials and suggested activities that relate to subsequent topics in the course.

Before building the instrument, it is worthwhile to discuss the basic components of a scientific instrument. Most, if not all, scientific instrumentation can be broken down into five parts:[Rayson2004]

- a source that generates the desired excitation or perturbation of the sample
- a sample compartment suitable to contain the sample in its appropriate form and quantity
- a discriminator that separates multiple signals emitted from the sample
- a detector that is capable of detecting the aforementioned signal at an appropriate level and within a suitable timeframe
- an output in a visual form appropriate for the characteristic that has been measured

4.1.1 Bill of Materials

You will need the following items:

- `The papercraft box`
- scissors
- razor blade or craft knife
- tape, either clear or black.
- `a piece of paper with a narrow slit` (more detailed instructions about the slit can be found here)
- a marker or some other way to darken the inside of the spectroscope box
- an old DVD (which will be destroyed)

4.1.2 Abbreviated procedure

Instructions and guidance are provided on the printed materials. Cut out the spectroscope and fold as instructed. Secure the slit to the appropriate location in the box. Carefully separate the two layers of the DVD by prying the edge of the disc with the razor blade. Take care to avoid/minimize the amount of aluminum sheet that remains on the *bottom* plate of the disc. The bottom side has grooves that can serve as a diffraction grating. It is coated in a purple dye that can be removed, if desired, by gently rinsing in alcohol. Cut the diffracting disc into the appropriate size and attach to the spectroscope. Once complete, tape the spectroscope to a phone so that the camera can capture an image of the diffracted light.

Fig. 4.1: Completed spectroscope attached to a phone.

4.1.3 What to do now?

Your spectroscope can be used to differentiate various light sources. In addition to typical sources such as fluorescent and incandescent light bulbs, consider the sun, streetlights, computer/phone screens, and LEDs.

Compare and contrast the spectra of various light sources. Are there aspects of the different sources that are consistent? How can you use the tools on your phone to record and compare various light sources? Is it possible to identify one light source as *stronger* than another with this instrument?

Here, we refer to the instrument as a *spectroscope* whereas the original authors use the term *spectrometer*. Research the difference between the terms and be prepared to argue which term is more suitable for the instrument you have fabricated.

Create a block diagram of the spectroscope. Identify each of the five components, noting that some aspects of a scientific instrument may be missing or be represented by the same aspect of the instrument.

Warning: See a problem? Have a suggestion? Please raise an issue and share your thoughts there.

4.2 Instrument description strategies

Before building instrumentation, it is useful to have some general approaches to describing instrumentation. In doing so, we will have an additional tool for breaking down instrumentation into individual components, allowing us to better understand the role of these components and how they contribute to the accuracy and precision of a measurement. Upon completing this section, you should have a better understanding of

- Block diagrams of scientific instruments
- The concept of data domains
- What a scientific instrument is
- What a measurement is

4.2.1 Block Diagrams

More details to come. This section covers the concepts presented in Rayson's article. [Rayson2004] Learning objectives are:

- Describe the five components of a scientific instrument
- Draw a block diagram of an instrument
- Compare and contrast different scientific instruments using a block diagram
- Evaluate potential impact of component changes in the modification of a scientific instrument

Implementation notes. Assuming students are familiar with a spectrometer, it is useful to use this instrument to introduce block diagrams. After this introduction, students can be asked to draw a block diagram of the papercraft spectroscope that they recently designed. Comparing the two instruments, students can often identify that the spectroscope combines the source and sample components.

Because the first instrument students design is the potentiostat, it is useful to look at Rayson's block diagram of this instrument (Figure 2 in his article).

4.2.2 Data Domains

More details to come. This section covers the concepts presented in Enke's article. [Enke1971] Learning objectives are:

- Define data domains
- Understand that a transducer convers data from a non-electrical to electrical domain (and vice versa)
- Evaluate the argument that a scientific instrument performs (at least) two data domain conversions
- Describe a scientific instrument in terms of a sequence of data domain conversions
- Understand that a measurement involves both a difference detector and a reference standard quantity
- Assess the accuracy of a measurement

Implementation notes. Data domains build upon block diagrams by first focusing on the last three blocks (discriminator -> detector -> display) of a visible spectrometer. Data are initially in non-electronic domain (light intensity of a particular wavelength) that is transduced by a photodetector into a voltage or current. This electrical signal is then further transduced into a number for visual display by the end user. While the process can occur in two steps in principle, a more realistic path involves data amplification and filtering (both of which involve data domain conversions within the electrical domain) as well as multiple data domain conversions in the non-electrical domain (a good example is the discriminator, which converts light intensity into wavelength-dependent light intensity). A number of the examples provided in the paper are outdated (unsurprising given that the paper is 50 years old at the time of this writing); however, the strategies proposed remain relevant.

Warning: See a problem? Have a suggestion? Please raise an issue and share your thoughts there.

4.3 The M4 Express

The Adafruit Feather M4 Express is a microcontroller that provides sufficient resources and computing power to create pedagogically meaningful scientific instrumentation. It can be programmed using either C++ or Python, and this text will use the latter - specifically CircuitPython - for all projects.

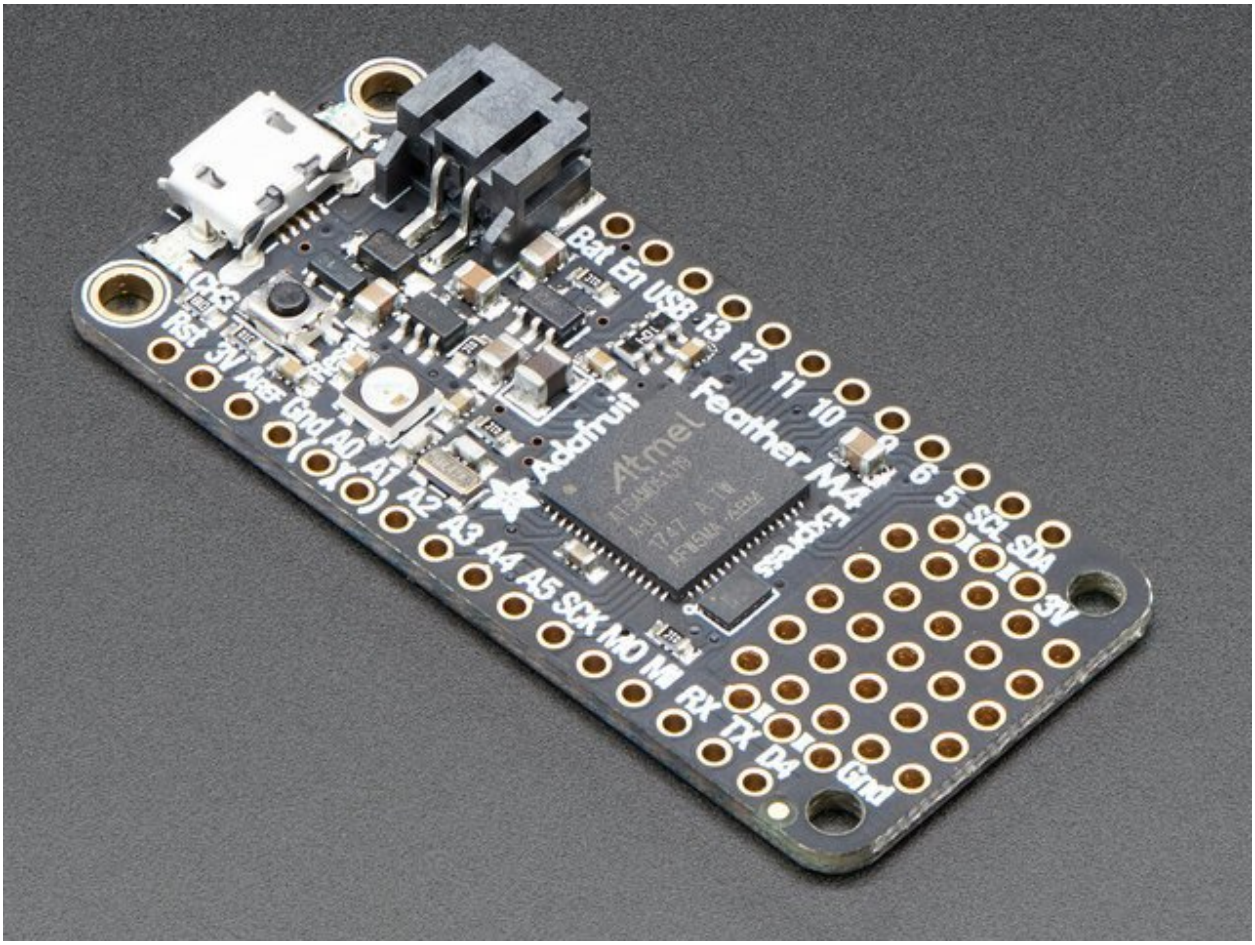


Fig. 4.2: The Adafruit Feather M4 Express (courtesy of Adafruit)

For more extensive details on how to use CircuitPython with M4 express, visit Adafruit's web page. Here you will find

a summary of information that is necessary to get up and running.

4.3.1 The Bootloader

All computer and computer-like devices require an operating system, and the software that loads this operating system into memory is called the *bootloader*. For microcontrollers such as the M4 Express, the bootloader controls what type of programming language can be used for the device. Due to the ease and accessibility, we use the UF2 bootloader, which allows the microcontroller to understand CircuitPython instructions and also makes updating the code extremely simple.

To install the latest bootloader, you must first download the most recent stable version. Next, with the M4 Express connected to your computer with a USB cable, you will double-click the resent button. At that point, a new drive (FEATHERBOOT) will appear on your computer. Drag the *.uf2* file you downloaded previously to this drive, which will case the red LED on the board to flicker and the FEATHERBOOT drive will disappear. A new drive, CIRCUITPY should appear, indicating that you have successfully updated the bootloader.

4.3.2 Programming with Mu

CircuitPython programs can be generated with any software that can generate text files; however, it is convenient to use a program designed for writing such code. Mu is a simple program that provides everything needed for this course and is available for Windows, MacOS and Linux. One of the advantages of Mu is that updating the software on the microcontroller is as simple as pressing the save button in the Mu software. No additional task is needed. Additionally, Mu (as well as other IDEs or Integrated Develop Environments) provides a tool for viewing output from the microcontroller using a *serial console*.

4.4 Circuit Python

Note: Ensure that you are using at least version 6.3.0 of Circuit Python. You can find this information in the *boot_log.txt* file located on the CIRCUITPY drive

With the latest version of CircuitPython installed on the M4 Express and Mu up and running, it is time to start programming the microcontroller. Type the following code into the Mu editor:

```
import board
import digitalio
import time

led = digitalio.DigitalInOut(board.LED)
led.direction = digitalio.Direction.OUTPUT

while True:
    led.value = True
    time.sleep(0.5)
    led.value = False
    time.sleep(0.5)
```

Some things to note about python code:

- commands are *case sensitive* which means *True* and *true* are two different things.

- spacing is important. Note that the code under the *while True:* line has been indented. The size of the indent isn't important; however the fact that all subsequent lines are indented the same amount is important.

Once the above code is saved in a file named *code.py*, you will notice that a red LED begins to blink regularly on the M4. Let's break down the code to see what was done.

```
import board
import digitalio
import time
```

These three lines import *modules* into memory. Modules contain pre-written code that is intended to be reused. In this case, *board* contains useful information like the location of the red LED. *digitalio* contains information about controlling the digital inputs and outputs and *time* contains the sleep function, which allows a programmer to insert delays in the code.

Note: Look at the remaining code and identify where these modules are used. You can tell they are used because the module names will precede a period.

```
led = digitalio.DigitalInOut(board.LED)
led.direction = digitalio.Direction.OUTPUT
```

The first line highlighted here creates a *variable* called *led* that instructs the program communicate with the output connected to the red LED using digital signaling. The second line indicates that digital signaling will be output from the microcontroller.

```
while True:
    led.value = True
    time.sleep(0.5)
    led.value = False
    time.sleep(0.5)
```

The remaining 5 lines contain the bulk of the program. First a loop is established; the command *while True:* is one way to tell python to keep performing the subsequent tasks indefinitely. The indentation identifies which commands should be performed during this loop. The next line, *led.value = True* turns on the LED. The subsequent lines tell python to do nothing for 0.5 seconds, turn off the LED, and wait another half second before repeating the loop.

Some things to try:

- Adjust the delays so that the LED is on for twice as long as it is off.
- Why are there two delays? What happens if you remove one of the delays?
- What happens if you start by turning the led off before turning it on?

Before continuing, it is useful to get into the habit of including documentation in the code. Comments can be included by prepending a line with a # symbol. Any text following that symbol, up until the end of the line, will not be viewed as an instruction. Excluding the *import* lines, add a comment before each line to describe what the code is doing.

4.4.1 Python Programming Activities

Note: Below is an activity designed to introduce some python programming constructs. In the future, additional activities will be incorporated into this section.

Morse Code Interpreter

In this suite of four projects, you will learn how to blink an LED using the digitalio interface, add timing delays to your code, assign variables, define functions, accept user input, manipulate character strings, and use other programmer's code.

Introduction

One of the challenges when programming with microcontrollers is the lack of a typical *interface*. We have grown accustomed to computers and computer-like devices having a screen of some sort that can transmit information in a textual or graphical format. A basic microcontroller does not have a display, and instead it must communicate with the user via the *serial console* - which requires access to another device that can read and display the serial console content - or other means such as light and sound. In this activity, you will use the red LED embedded on the M4 Express microcontroller to communicate with the end user.

Because it is relatively straightforward to control the state (on/off) and timing of an LED, it is possible to develop a Morse Code like communication strategy. Morse Code converts alphanumeric values into a series of dots and dashes. These dots and dashes can be converted into LED blinks of varying durations. Here are the timing rules for Morse code:

- The length of a dot is one time unit
- The length of a dash is 3 time units
- The delay between characters in the same letter is one time unit
- The delay between characters in the same word is three time units
- The delay between words is seven time units.

There is no official definition for the time unit, so that value can be whatever is *reasonable* for your application.

Assignment submission Should this activity be used as an assignment, the following submission guidelines are recommended. Code for each of the four programs should be submitted with sufficient annotation of the code. At a minimum, each function and flow control loop should be preceded with a comment indicating its operation, global variables should be described upon first use/assignment, and the program should begin with a comment section indicating the intended purpose of the program and the program's author.

In addition to annotated code, the submission should include a paragraph that comments on the following topics: describe the data domain conversions using Enke's data domain map; can the Morse code generator be considered a *signal transducer*; identify one modification to the code that could make the final program more concise without compromising readability.

Program 1

Create a sequence of LED flashes that corresponds to the letter A. Create a global variable that sets the unit of time to a value that is comprehensible to you. The loop should include a word-length delay (7 time units). This program can be completed using only the commands introduced in the example code shown above.

Optional Expansion Repeat the same process but for a chemical symbol of your choosing. A two-letter chemical symbol will require the inclusion of a letter-length delay

Program 2

Modify the first program to replace the on/delay/off sequences with functions. See Chapter 3 of *Automate the Boring Stuff* or perform a web search on *python functions* for help on the structure of functions in Python. Functions should refer to a *global* variable that contains the standard unit of time (i.e. the duration of a dot). Create functions for dot and dash that do not require arguments and a space function that takes an argument indicating a same letter, same word or between word delay.

Note: One way to create the delay function is to make the argument the length of the delay. Then global variables such as *letterdelay* or *worddelay* with the appropriate amount of time.

Program 3

Building on the functions from Program 2, allow the user to input a sequence of dots and dashes to the serial console and have that sequence displayed on the LED. Your solution should involve an *if* statement, and details can be found in Chapter 2 of *Automate the Boring Stuff* or a web search on *python conditional statements*. Your program should handle dots and dashes, treat spaces as letter delays and ignore all other characters.

Note: To accept user input from the serial console, use the *input()* function. A line such as *inp = input("what is your age? ")* will prompt the user for her age and place the result in the variable *inp*.

Program 4

Lastly, you are to incorporate another programmer's code into your own. Refer to this Morse code translator example and copy the definition for the variable *MORSE_CODE_DICT* and the function *encrypt* into your program. Use the *encrypt* function to translate user input from alphanumeric text into Morse code. Display the encrypted text in the serial console and blink the code on the LED. This exercise introduces python dictionaries, which are covered in Chapter 5 of *Automate the Boring Stuff* or a web search on *python dictionaries*.

Optional Expansion Research the string method *upper()* and incorporate it into your program so that the user does not have to type her response in all capital letters.

Future

Can you create a game that emits the Morse code for a chemical element?

Warning: See a problem? Have a suggestion? Please raise an issue and share your thoughts there.

4.5 How to solder

4.5.1 Resources

While this section is a work in progress, visitors are referred to the following useful resources:

- Video based instructions for soldering
- Text based instructions for soldering
- Safety guidelines for soldering

4.5.2 Outline

NOTE Since this course is geared towards chemists, a discussion of soldering should include chemical compositions

- Soldering is the act of joining two metals together with a fusible metal alloy called solder.
- The composition of early solders was 60/40 tin/lead, which melts at 188 C (370F)
- With the push to remove lead from consumer electronics, lead-free solder may contain many other elements such as copper, silver, bismuth, indium, zinc and antimony. The result is an alloy with a higher melting point, upwards of 200 C higher. A lead free solder commonly in use is tin-silver-copper with a melting point of 217/423
- Solders also contain a flux, which is a chemical cleaning agent designed to eliminate/remove metal oxides, often by a chemical reduction reaction. Compositions vary greatly (and are confidential intellectual property) but fall into three basic categories: rosin, organic and inorganic. Only the first two are typically found in electronics soldering and consist of various acids ranging from hydrochloric and ammonium chloride to citric and abietic.
- Practical standpoint - typically soldering components that have been designed to be easily soldered, so just do it.
- Through hole connections: insert the piece into the circuit board, prepare to solder from the *opposite* side of insertion, use soldering iron at 450F (220 C) to heat the components for ~ 2 seconds, touch solder to the heated components. Remove solder wire then remove the soldering iron. The components we use are reasonably tolerant to heat, which means you can go a bit slower to start with, but keep the heating to under 10 seconds.
- Check the solder joint for a peaked shape. Round indicates that there is too much solder and too little will result in the through hole not being filled. Also check for bridging, which is excess solder connecting pads that should not be connected.

Warning: See a problem? Have a suggestion? Please raise an issue and share your thoughts there.

4.6 Back to Circuit Python

The following examples make use of the bob173 featherwing, which you should be able to prepare now that you have worked through the soldering introduction. If you have not yet built this featherwing, visit *Building the potentiostat*. After completing this suite of exercises, you should have a better understanding of digital and analog input.

Note: In the introduction to Python programming, you learned about the general structure of a microcontroller program (importing modules, initializing variables, defining functions and the loop routine), digital output (blinking the on-board LED) and accepting user input through the serial console. You should also understand the difference between terms such as *function*, *variable*, and *argument*. Revisit *The M4 Express* if these concepts are still challenging for you.

4.6.1 Digital input

In the previous exercise, we learned that we can use digital output to turn an LED on or off. In this exercise, you will learn how to read digital input. The bob173 featherwing has a tactile switch (two actually) that is connected to one of the digital input/output pins of the microcontroller. Looking at the schematic (LINK), you can answer the following questions:

Questions

- What digital pin is the button connected to? (Hint, the name is also printed on the featherwing circuit board)
- A tactile switch connects two points of the circuit. In addition to the digital input/output noted above, what else is the switch connected to?

The basic program for initializing and blinking the built-in LED is reproduced here:

```
import board
import digitalio
import time

led = digitalio.DigitalInOut(board.LED)
led.direction = digitalio.Direction.OUTPUT

while True:
  led.value = True
  time.sleep(0.1)
  led.value = False
  time.sleep(0.1)
```

Your task is to modify this code so that the LED is turned on/off with the push of a button. The first task is to initialize the digital pin associated with the button.

```
btn = digitalio.DigitalInOut(board.D11)
btn.direction = digitalio.Direction.INPUT
```

Note that we create a new *variable* for the button (*btn*) and assign it to the appropriate digital pin. Next we set the direction of that pin to *INPUT* since the program will read the value of the pin.

How do we check if the button is pressed? Notice from the schematic that the digital pin will be connected to ground when the button is pressed. Therefore, we should be able to check for *btn.value* == *False*, which would indicate that the LED should be turned on (*led.value* = *True*).

```
while True:
  if btn.value == False:
    led.value = True
  else:
    led.value = False
```

If you try this code (and you should) you may find that the LED stays illuminated. Something isn't quite right. If we look back at the schematic, we get a hint about the issue. We know what the value of the digital pin should be when the button is pressed, *but what is the value when the button is **not* pressed?* That answer is not clear from the schematic.

Question

Based upon your observations on how the code runs, what do you think the *default* value of the digital pin is?

The issue we are facing is a common one, and is addressed internally by the microcontroller. The simplified schematic below demonstrates what the microcontroller does to each of its digital pins:

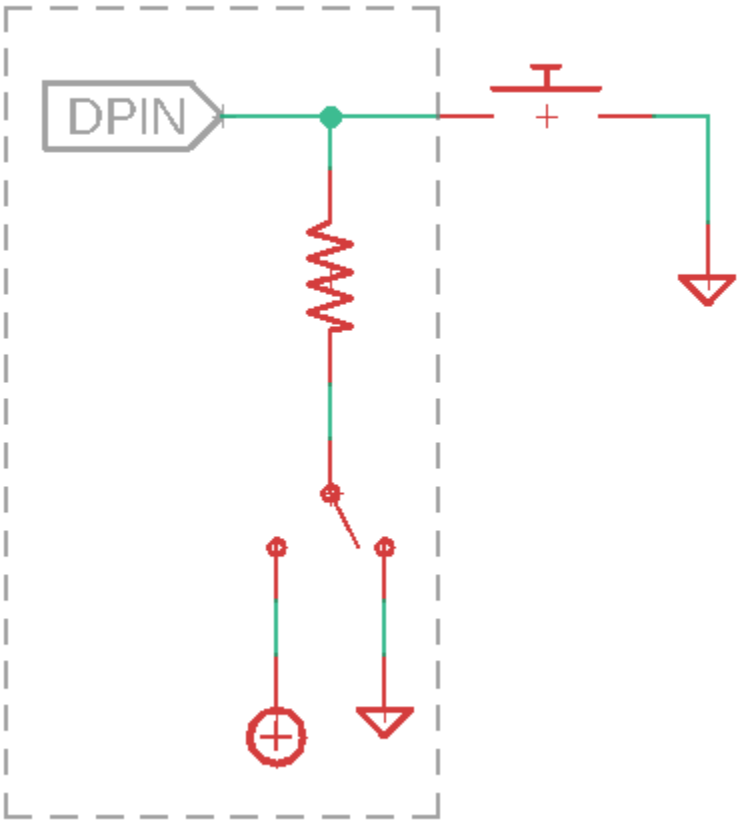


Fig. 4.3: Generalized schematic of a pullup/down resistor

Internally, the digital pin is connected to a resistor. That resistor can be fed either to ground or to the positive voltage supply. If it is tied to ground, then the resistor is referred to as a *pulldown resistor* since an unconnected pin will be drawn to ground. Conversely, a *pullup resistor* is tied to the positive supply, and will have a default high value.

To set the pullup/down value, we need to add one additional line to the setup of the button:

```
btn = digitalio.DigitalInOut(board.D11)
btn.direction = digitalio.Direction.INPUT
btn.pull = digitalio.Pull.UP
```

Now, when the button is *not* pressed, the default value fed to the digital pin will be True. The code should now work as intended.

On your own

Modify the above code so that the LED is by default *on* and only turns off when the button is pressed.

4.6.2 Analog input

With digital input/output, we can only deal with signals that are binary (on/off, high/low, true/false). In order to access a continuum of values, analog input/output is required. Many microcontrollers have pins that are capable of reading an analog value (analog inputs) but have to fake an analog output using a process called *pulse width modulation*. The M4 microcontroller, however, has two pins that provide true analog output, which is one of the reasons it is an attractive component for instrument design. In this exercise, we focus on analog input using the voltage divider labeled PVG on the bob173 featherwing.

Question

Return to the bob173 featherwing schematic and search for the name of the pin that is connected the voltage divider. It is labeled *VGND*. You are interested in the A# part of the name.

Enter the following code into your microcontroller

```
import board
import analogio
import time

adc = analogio.AnalogIn(board.A4)

while True:
    myval = adc.value
    print(myval)
    time.sleep(1)
```

At this point, you should see a number appear in the serial console that can be changed by turning the knob on the potentiometer.

Note: The term potentiometer is another name for a variable resistor. The potentiometer used in this circuit requires 25 turns of the knob to go from 0 to 22 kOhm.

If you turn the knob clockwise or counterclockwise, you will find the minimum and maximum values are approximately 300 and 65000. Nothing yet informs us what the *units* of these values are. In principle, the value should range from 0 to 3.3 V since those are the minimum and maximum voltage values of the M4 microcontroller analog input pins. In order to properly measure an analog signal, a microcontroller (or any computer, for that matter) must convert that signal into a digital format. This process is called *analog to digital conversion*, where the microcontroller divides the

range of possible analog values into a certain number of discrete value. The number of discrete values is referred to as *resolution* and in this case, there are 65536 (2^{16}) values.*[0] In order to obtain an analog voltage from the analog pin reading, we must first perform a conversion.

$$V_{reading} = V_{max} \times \frac{\text{value}}{\text{max value}}$$

We can modify the python code above to report a voltage by adding the following line:

```
myvoltage = 3.3 * myval / 65536
```

Note: If your program still returns an integer value, double check that you are printing the correct variable.

At this point, you should find that the output ranges from a value slightly above 0 V to slightly below 3.3 V. To understand why the reading does not reach the minimum and maximum values, we first need to understand how real operational amplifiers work, so we'll circle back to this issue later in the course.

The last python tip in this activity is to format the results in a slightly nicer way. We are going to replace the print statement with:

```
print(f'{myvoltage:.3f} V')
```

The above command uses what is called an *f-string* where f stands for *format*. An f-string is a string in single quotes preceded by an f. Curly braces inside an f-string are interpreted as variable placeholders, so instead of literally writing "{myvoltage:.3f}", the program will output the value of *myvoltage*. The *:.3f* part allows for formatting the number of decimal places (in this case, 3). Note that f-strings allow for the programmer to easily combine variables with text, so the incorporation of units as is done in this case, is straightforward.

You try

Change your program to output the voltage in millivolts with one decimal place. This change will require you to modify both the value for *myvoltage* and the f-string.

Tip: See a problem? Have a suggestion? Please raise an issue and share your thoughts there.

4.7 Wrapping up the Introduction

Each module will consist of some concluding remarks and self-check activities to help the learner demonstrate that material has been covered sufficiently. It is strongly encouraged that the activities presented here be completed, even if they are not officially assigned.

In order to understand how scientific instrumentation is designed, it is first necessary to learn about some strategies to discuss instrumentation. In general, the purpose of a scientific instrument is to convert a chemical/physical phenomena into a quantity that can be visualized by the scientist. We used Enke's data domain concept to describe the variety of conversions or *signal transductions* that can occur in scientific instrumentation.[Enke1971] We then coupled that concept to block diagramming of scientific instruments. As described by Rayson, virtually all instruments can be subdivided into 5 components: source, sample, detector, discriminator, and output.[Rayson2004]

[0] Technically, the M4 microcontroller has only 12 bit ($2^{12} = 4096$) resolution. The authors of Circuit Python made the decision to convert all analog readings to 16-bit to make the output of the analogio functions platform independent.

While this course may be your first introduction to microcontrollers, it is possible that you have had exposure to them in other areas of your life. Many physics and computer science programs have introduced microcontrollers into their curricula, and hobbyist microcontrollers such as the Arduino are becoming widespread in home-made light shows, sensor and actuator applications. Here, you have been introduced to the M4 Express, which has a set of features amenable to scientific instrument design. We have relied quite heavily on Adafruit's overview page to explore how to update the microcontroller, explore the pinouts and program the device using Circuit Python and the Mu editor.

You have been introduced to some basic programming in Circuit Python (CP) and should be familiar with importing libraries, setting variables, defining functions, and creating loops and conditional statements. You have also learned how to send and receive binary information from the digital pins as well as reading and interpreting an analog signal.

Lastly, you have learned how to solder and have followed instructions for building your very first instrument. In the next section, you will explore that instrument further, learning what the components are in the instrument and how they function.

4.7.1 Self Check

1. Draw the block diagram of two scientific instruments with which you are familiar (e.g. a pH meter and a visible spectrometer).
2. Identify what data domain conversions occur within or between the blocks in the instruments you diagramed in the previous question.
3. Confirm that your microcontroller is running the latest firmware and update it if necessary.
4. Write a program that will use the on-board red LED to display the Morse code for the chemical symbol of technetium when the user presses a tactile switch.

Tip: See a problem? Have a suggestion? Please raise an issue and share your thoughts there.

CHAPTER

FIVE

INSTRUMENTAL APPROACH TO VOLTAMMETRY

In this three week module, you will learn about the electrochemical method called *cyclic voltammetry* by building an instrument that performs the technique: a *potentiostat*. We are going to use an *instrumental approach* whereby the focus will be on how to create an instrument that does the experiment we want to perform.

Warning: See a problem? Have a suggestion? Please raise an issue and share your thoughts there.

5.1 The Voltage Divider

5.1.1 Introduction and prerequisite reading

A voltage divider is a simple-yet-important circuit that turns a large voltage into a smaller one. It will appear in several parts of the potentiostat, most importantly in creating something called *virtual ground*[1]. Upon completing this activity, you should have a better understanding of what a voltage divider circuit is, the mathematics behind voltage dividers, how to select components for a desired voltage, and how voltage dividers are used in a few real-world applications.

Before starting this activity, read this introductory tutorial from Sparkfun.

5.1.2 Simulated voltage dividers

The model below depicts a voltage divider with two 1 kOhm resistors and a 5 Volt power supply. The desired output voltage is obtained from the point in between the two resistors. The triangle-like shape is called *ground* and allows for all current paths to complete a loop. (Not shown is that the power supply is also connected to ground.)

Explore this simulation by doing the following:

- Change the values of the resistors and observe how they impact the output voltage. To make this exercise more manageable, limit your resistor choices so that the *sum* of the resistance remains 2k (e.g. 500 and 1.5k, 250 and 1.75k). Determine if the output potential is proportional to the top resistor or the bottom resistor.
- The simulator indicates the flow of current with yellow dots. Placing the mouse over a component will reveal the current flowing through the object and may provide other information. Mouse over the voltage source and note the power. Power is calculated using the equation $P = V \times I$ with units of Watts (W). Determine if the current and power change with changing the resistor values.
- Perform the same exploration but start with resistor values of 10k each. When adjusting the resistor values, keep the total resistance equal to 20k. Note similarities and differences in the output voltage, current and power.

[1] We will cover the concept of virtual ground later.

Create your own circuit:

You may erase the circuit in the on-line textbook (and recover it by refreshing the page) or open a new browser with the circuit simulator. Create a voltage divider with a *potentiometer* instead of two resistors. Make the input voltage 9 V and select a resistance value that results in a total power draw of less than 1 mW. Adjust the potentiometer to achieve an output voltage of 3.3 V (approximate). Some tips on using the simulator:

- The components needed for this circuit are a one-terminal voltage source, ground, potentiometer, and voltmeter. These items can be found in the Draw menu (submenus Passive Components, Inputs and Sources, and Outputs and Labels).
- Note the shortcut keys (e.g. w for wire, r for resistor) as these come in handy.
- You can save/submit your circuit as a text file from the File menu (Export as Text...).

5.1.3 Real world applications

An important use of voltage dividers is in the measurement of resistive sensors, one example being photoresistors. A photoresistor is a passive element that has a resistance that changes with the brightness of light shining on it. The simulator software has a photoresistor element that you can incorporate into a voltage divider. Create a circuit that can generate a response that is proportional to light brightness. Consider the following in your design:

- Assume that you only have access to a limited supply of resistors (100 Ohm, 1 kOhm, 10 kOhm). Which one is best suited for the photoresistor and why?
- Create a plot of output voltage vs light brightness. For the latter, estimate the percent brightness from the slider (no need to be more precise than 0, 20%, 40%, 60%, 80%, 100%).
- Explore the impact of having the photoresistor in the "upper" vs. "lower" positions of the voltage divider. Which would you choose and why?
- Identify the data domain conversions implemented in this circuit. Refer to the data domain map (Figure 6 in Enke's paper)[2]

Another real-world application is *level shifting*, which is important when one wants to communicate between two devices with different voltage levels. For example, while many new sensors utilize 3.3 V in their communication protocols, the ubiquitous Arduino uses 5 V logic, which can damage the sensor. A voltage divider (much like the one simulated in the *create your own circuit* section) can be used to facilitate uni-directional communication. You do not need to create a logic shifting circuit, since it is similar to the one above; however, if you are so inclined, it would be useful to find the combination of resistors between 1 kOhm and 10 kOhm that shifts a 5 V signal to a 3.3 V signal.

Note: Voltage dividers **cannot** be used in bi-directional communication because it can only shift a voltage *down*. A 5 V source can be shifted down to 3.3 V, but it does not make sense to shift a 3.3 V source *down* to 5 V.

5.1.4 Using the FeAtHEr-Cm voltage divider

Set up the code

Refer back to *Back to Circuit Python* for a refresher on how to read analog input. Modify that code if necessary to read the analog input from the *PVG* potentiometer, which is connected to analog pin 4 (A4). Convert the reading to a voltage and print that voltage to the serial console at a rate of 1 hz. Round the result to 3 decimal places.

[2] C.G. Enke, *Anal. Chem.*, **1971**, *43*, 69A link

Data collection

Adjust the knob of the PVG potentiometer so that the output reads in the neighborhood of 1.65 V. Collect voltage readings as a function of the number of knob turns from this location. Make note of the direction (clockwise or counterclockwise) but it is not necessary that you do both in this experiment. Aim for 6 to 10 data points.

Do some math

- Estimate the $\Delta R/turn$ of the potentiometer ($R_{total} = 22\ k\Omega$) noting that the total number of turns is 25.
- Create an expression that relates the upper resistance (R_1) to the input voltage ($V_{in} = 3.3\ V$), the output voltage (V_{out}) and the total resistance of the potentiometer.
- Repeat the above process for the lower resistance (R_2).

Plot and analyze

- Create a spreadsheet with columns for the number of turns, output voltage and calculated upper/lower resistances.
- Create three plots with output voltage, or resistance plotted as a function of number of turns.
- Find the slopes of the three plots. For the resistance plots, determine if there should be a relationship between the two slopes.
- Use the *LINEST()* function in Excel to find the error in slope of the upper resistor plot. Find the percent relative error in the slope. Compare the slope to the expected $\Delta R/turn$ value calculated earlier.

Before proceeding, adjust the potentiometer such that the output voltage is 1.65 ± 0.01 V.

Exploring the impact of load

Until now, the voltage divider has been used to produce a voltage, but the produced voltage has not been used. For example, one might want the produced voltage to turn on an LED or drive a motor. The general term for these cases is *load*. In the following activity, you will explore the impact of load on the ability of a voltage divider to supply the desired voltage.

- Identify the location of the resistor labeled RLD on the FeAtHEr-Cm potentiostat.
- Note the output voltage indicated by the Python script while no resistor is in the RLD slot. Then insert a resistor (1 kOhm, 10 kOhm, 100 kOhm) and note the change in output voltage.
- Repeat the previous step with the remaining resistors and develop a relationship between the load and the *error* in the output voltage.
- Remove the RLD resistor and adjust the voltage divider to 2.5 V. Repeat the above steps. Do this one more time with the voltage divider set to 0.75 V and determine how the magnitude of the error changes with the desired output voltage.
- Identify any trend between the magnitude of the load resistance and the total resistance of the potentiometer. Under what conditions is the output voltage error the smallest?
- Summarize the relationships between load resistance, desired output voltage and actual output voltage in several brief sentences.

Upon completion of this activity, you should have a better understanding of what a voltage divider is, some real-world applications as well as limitations. Since the potentiostat we build depends on using a voltage divider to drive a load, the next section will discuss how to overcome the limitations presented here.

Warning: See a problem? Have a suggestion? Please raise an issue and share your thoughts there.

5.2 Voltage Divider Problems

Note: In the last section, you should have noticed that the output voltage varies proportionally with the lower resistor and that the current and power are dependent on the sum of the two resistors. The equation for determining the output voltage is $V_{out} = V_{in} \frac{R_{lower}}{R_{lower}+R_{upper}}$

In addition, you finished that section by exploring how a resistive load applied to the voltage divider on the bob173 potentiostat impacts the output voltage. Here, we will use the circuit simulator to explore this phenomenon more systematically.

5.2.1 Setup

One way we could create a variable potential is by changing the resistance values. A variable resistor, or *potentiometer* provides this functionality. The circuit below is similar to the one on the previous page, with several important differences:

Note: Visit https://tinyurl.com/yfeq6ato to perform the activity described below.

- The two resistors are replaced by a potentiometer, which makes it easy to adjust the resistances above and below the test point while keeping the total resistance the same. You should see a slider on the right hand side of the simulator that will control the potentiometer position.
- A switch has been added, which can be opened and closed with a double click.
- A new resistor has been added, which is the *load* or the object that we want to apply the output voltage to.

Explore this simulation by doing the following:

- Make sure the switch is open. Adjust the potentiometer so that the resistances are roughly equal and the output voltage is about 2.5 V.
- Close the switch and note how the output voltage is affected.
- Explore how the voltage difference between open and closed states is affected by the position of the potentiometer. Where do you see the maximum difference and where is the difference negligible?
- Perform a more systematic analysis by collecting voltage readings (with switch closed) as a function of lower resistance value. You will need to collect about 11 points: 1 with the potentiometer at its middle setting and one at each of the extremes, then four above and below the middle setting.
- Plot these results along with the expected value, which you can calculate using the voltage divider equation.

5.2.2 Investigation

To understand why the load resistor is causing the output of the voltage divider to decrease, we must first recall (or learn) that resistors in a circuit can be mathematically combined, and the math depends on whether the resistors are in series or parallel. In series, the resistors add as you would expect: two 1-kOhm resistors in series results in a total resistance of 2 kOhm. If those resistors are in parallel, then the *inverses* of the resistances are added:

$$\frac{1}{R_{parallel}} = \frac{1}{R_1} + \frac{1}{R_2}$$

$$R_{parallel} = \frac{R_1 \times R_2}{R_1 + R_2}$$

Take a closer look at the voltage divider with a resistive load, and you will note that the lower resistor is in parallel with the load. Because these two resistors can be added together, we are in effect *changing the voltage divider*. The new voltage divider has the same resistance for R_{upper} but R_{lower} is now the parallel summed value of the original R_{lower} and R_{load}. Incorporating this change into the voltage divider equation, we get:

$$V_{loaded} = V_{in} \times \frac{R_{lower}R_{load}}{R_{upper}(R_{lower} + R_{load}) + R_{lower}R_{load}} \tag{5.1}$$

We can further "simplify" this equation by noting that the upper and lower resistances are related to one another, given that we are using a potentiometer. Substituting $R_{pot} = R_{upper} + R_{lower}$ into the above equation, it is possible to remove the upper resistance from the equation, yielding:

$$V_{loaded} = V_{in} \times \frac{R_{lower}R_{load}}{R_{load}R_{pot} + R_{lower}R_{pot} - R_{lower}^2}$$

You try

Use the above equation to calculate the simulated voltage of a loaded voltage divider to determine if the "experimental" data you obtained from the simulator matches the results expected from the math.

Digging deeper

It is possible to analyze the resistive load error a bit further. To do so will require a bit of calculus. If you have a good handle on derivatives, then you should be able to see how the equations presented are connected to one another. If you have no experience with calculus, then simply follow along and do not be bothered by how one equation leads to the other.

Our goal is twofold: first, we seek to determine quantitative where along the potentiometer will the greatest error due to resistive load occur; and second, we wish to explore how the magnitude of the resistive load influences the voltage divider performance.

We can define the voltage divider error as the ratio of the loaded output to the non-loaded output. The non-loaded output voltage is equal to $\frac{V_{in}R_{lower}}{R_{pot}}$ and the loaded voltage is given above. The error is then:

$$\text{error} = \frac{V_{loaded}}{V_{out}} = \frac{R_{load}R_{pot}}{R_{lower}R_{pot} + R_{load}R_{pot} - R_{lower}^2}$$

Plotting the error as a function of R_{lower} you will find that the largest amount of error occurs around the point where the potentiometer is in its central position, $R_{lower} = \frac{R_{pot}}{2}$. Substituting this equality into the error equation, we obtain an expression for the maximum amount of error that is a function of just the potentiometer and the load resistances.

$$\text{error}_{max} = \frac{R_{load}R_{pot}}{\frac{R_{pot}^2}{4} + R_{load}R_{pot}}$$

Note: If you know calculus, you should attempt to take the derivative of the error expression (remember tha chain rule) and set the result equal to zero. Upon simplifying, you should get the above answer.

With an expression for error$_{max}$, it is now possible to explore how the relationship between the load and potentiometer resistances impact the performance of the voltage divider. We will consider three conditions, when the two resistances are equal and with the load resistance is considerably larger or smaller than the potentiometer resistance.

$$R_{load} = R_{pot}$$
$$R_{load} = 10R_{pot}$$
$$R_{load} = 0.1R_{pot}$$

When the resistances are equal, error$_{max} = 0.8$, so the voltage divider output is attenuated by 20%. As the load resistance becomes much larger than the potentiometer resistance, the impact decreases. At 10 times the resistance of the potentiometer, the load attenuates the voltage divider output by 2.5 percent. The opposite effect is seen when the load is very small relative to the potentiometer, and when $R_{load} = 0.1R_{pot}$, the output has been attenuated by over 70% and the voltage divider is clearly not functioning properly.

The end result of this analysis is that a voltage divider can only work properly when the load resistance is much greater than the potentiometer resistance. This result might suggest that the solution to using voltage dividers is to use the smallest potentiometer resistance possible. However, one must also keep in mind the power consumption of the potentiometer. As the potentiometer resistance decreases, the current (and therefore the power) will increase, resulting in a poor use of the power supply.

Tip: See a problem? Have a suggestion? Please raise an issue and share your thoughts there.

5.3 An operational amplifier primer

Note: We saw in the last section that the load we are trying to work with impacts the performance of a resistor-based voltage divider. Ideally, we need to isolate the voltage divider from the load. In this section we will learn how a component called an *operational amplifier* can perform that task.

5.3.1 Just the facts

As this material is designed for chemists who need to know (or want to know) some useful electronics, we will skip a lot of the theory behind operational amplifiers and focus on the two important features that make them useful and then move on to several basic configurations that are found frequently in scientific instrumentation.

An operational amplifier is an active component that amplifies weak signals. It has two inputs (a non-inverting input designated **+** and an inverting input designated **-**) and one output. The symbol used for operational amplifiers in schematics is a triangle.

Note that in the diagram above, a square wave pulse with amplitude +/- 5 V is applied to the inverting input and the non-inverting input is tied to ground. Try the following explorations:

- Describe the output in relation to the input
- Mouse over the oscilloscope trace to find the amplitude of the output signal. Where does this value come from? (Hint: right click on the op amp and select properties.)
- Reconfigure the inputs so that the output signal is in phase with the input signal.

In this circuit, the op amp is behaving as a *comparator*. It is comparing the two inputs and responding with the following result

- If the inverting input is greater than the non-inverting input (which is tied to ground) then set the output as negative as possible.
- If the inverting input is less than the non-inverting input, then set the output as positive as possible.

Note that the comparator is also amplifying the signal. Look at the properties of the input signal and note that the voltage of the square wave is $\pm 5\ V$ while the output is $\pm 15\ V$ for an effective gain of 3. Notice what happens when the input square wave voltage is decreased further. For square waves as small as 5 mV, the comparator still provides a crisp output with the voltage limits defined by the input voltages supplied to the operational amplifier. This behavior is what is expected in an *ideal* situation, however.

The simulation software allows for using a more realistic op amp model. Swapping out the ideal op amp for a real one (under active building blocks) and note that the output no longer has a crisp switch from low to high and vice versa. Increasing the voltage of the input square wave restores the behavior to that of the ideal... almost. Mouse over the output scope and note that the output square wave is not actually cycling between $\pm 15\ V$. Many real operational amplifiers are not able to set their outputs to the supply voltages. Those that can are referred to as *rail to rail output*. Although not demonstrated in this simulation, it is difficult for some op amps to accept input voltages as the rails as well. Those that can do both (accept and deliver voltage at the power supply limits) are referred to as *rail to rail input/output* or RRIO.

A comparator does one more thing, which isn't readily apparent in the simulation: if the inputs are equal, set the output to zero.

Note: You should be able to give this a try by deleting the ground and adding a wire from the square wave source to the non-inverting input.

5.3.2 Two important points

We'll quickly see that op amps can do much more than just compare the values of their inputs. As we explore some other configurations, there are two important rules that we must keep in mind:

1. An operational amplifier does everything in its power to make the *difference* between voltages at the inverting and non-inverting inputs be zero.
2. In an ideal operational amplifier, the inputs do not draw any current.

Warning: See a problem? Have a suggestion? Please raise an issue and share your thoughts there.

5.4 Important Op Amp Circuits

Operational amplifiers have many uses in scientific instrumentation, and they play a central role in the design of a modern potentiostat. We will see shortly that the four op amps in the potentiostat we build perform one of three different roles: buffering an input signal; adding input signals together; amplifying a current while converting it to a voltage. We will take a closer look at these three op amp uses.

5.4.1 Op amps as buffers

A major problem with using a voltage divider to drive a load is that any resistive load applied to a voltage divider *becomes part of the voltage divider*. A careful inspection of the voltage divider circuit highlighted that one way to circumvent the influence of load on a voltage divider is to prevent current flowing through the divider from also flowing into the load. In other words, if it is possible to block (or buffer) the voltage divider from the rest of the circuit, it should function as intended. An operational amplifier can perform this function, and an example is shown in the simulation below.

The 22 kOhm potentiometer is set up to divide the 3.3 V input voltage in half, resulting in an output of 1.65 V. A 1 kOhm load resistance greatly attenuates the output voltage (you should be able to determine this value of ~ 253 mV); however, the op amp keeps the output voltage at the desired value.

First, let's emphasize that an op-amp buffer configuration has the output pin tied to the inverting input pin and the non-inverting pin is connected to the voltage divider. Recall that an op amp tries to keep the difference between the two input pins equal to zero, which means that it will try to set the inverting input to 1.65 V. Since this pin is also connected to the output, the output will remain at 1.65 V.

Notice in the simulation that no current is flowing into the non-inverting pin of the op amp. As far as the voltage divider is concerned, the op amp looks like a huge resistor. This characteristic of an op amp is called *input impedance* and an ideal op amp has such a high input impedance that *no current flows into the inputs*. In reality, op amps have a finite input impedance, meaning that they will draw a small amount of current.

You try

Now would be a good time to start exploring datasheets of operational amplifiers. Perform a websearch on "MCP6004 datasheet" to find the specifications of the op amp used in your potentiostat. Look through the table to find the value of *input impedance*. Use this value to estimate the amount of current flowing into the non-inverting input if the input voltage is 1.65 V.

One may ask where the current is coming from that exits the op amp. The answer is the *power supply*. Not shown in the schematic is that the op amp is connected to the ground and +3.3 V rails of the instrument. If you pay close attention to the schematic for the bob173, you'll note that one of the four op amps has two extra pins to indicate connections to the power supply. Circuit components that require power to function properly are referred to as *active components*. Those we have used earlier (resistors, photoresistors, potentiometers) are examples of *passive components* that do their job without the use of a power supply.

You try

There is one more buffer located in the bob173 schematic. Can you find it?

5.4.2 Op amps as amplifiers

As the name suggests, op amps can amplify voltage or current signals. Let's explore the circuit below:

First, notice the construction. The non-inverting input is tied to ground and the inverting input has two connections, one is a *feedback resistor* that is connected to the output source and the other is the input signal, which in this case is a voltage source passed through a resistor. The simulation indicates that current is flowing from source, through the feedback resistor and into the output pin of the op amp. The output voltage is -100 mV (with some rounding errors). How is the output voltage related to the source?

Notice that the source generates a current base upon Ohm's Law ($V = IR$) of 100 μA. This current then flows through the feedback resistor to generate a voltage (again, through Ohm's Law) of 100 mV. However, since we are tied to the inverting input of the op amp, the output voltage has a sign opposite of the input.

you try

1. Change one of the circuit elements to result in a positive output voltage.
2. Convince yourself that the current flow is not impacted by the value of the feedback resistor. Recall that you can view the current in a circuit by moving the mouse over a wire.
3. Convince yourself that the circuit elements labeled as *source* do in fact control the current. You should be able to change the current by altering either the voltage or the resistor.

In addition to viewing the above circuit as a current follower (or current to voltage converter), it is also possible to think of the circuit as a voltage scaler. To do so, we revisit the two Ohm's Law equations that describe the circuit, and notice that they can be rearranged to eliminate i from the equations:

$$
\begin{aligned}
V_{source} &= i \times R_{source} \\
V_{output} &= -i \times R_{feedback} \\
\frac{V_{source}}{V_{output}} &= -\frac{R_{source}}{R_{feedback}} \\
V_{output} &= -V_{source}\frac{R_{feedback}}{R_{source}}
\end{aligned}
$$

There is an important point to remember when using an op amp in current-to-voltage (or voltage scaler) applications: the output voltage cannot exceed the power supply rails. This limit has some hidden consequences for using op amps with microcontrollers because many instructional materials assume that the circuit employs a *bi-polar power supply*, which is not the case for the standard array of microcontrollers.

you try

To understand the problem, change the simulation to have a Minimum voltage of 0 and a maximum voltage of 3.3. With a positive source voltage, is the output voltage the same as that predicted from the equations above? Is it possible to scale a negative voltage source?

There are two approaches to addressing the lack of a bi-polar power supply (commonly referred to as single source). One is to design the instrument such that the source will only be one polarity. We will revisit this concept in the next instrument that uses the current from an illuminated LED. The second approach is to use the concept of a *virtual ground* that effectively changes what the instrument considers as zero. That is what is done in the potentiostat. Before exploring how a virtual ground is made and used, we will look at the last op amp configuration important for this course.

5.4.3 Math with op amps

With operational amplifiers, it is possible to perform many mathematical operations. The voltage scaler just described is an example of multiplying a voltage by a constant value. There are also circuits for adding signals together and performing complex operations like differentiation, integration and even logarithms. Nowadays, many of the complex operations can be performed sufficiently quickly through software (programming) that they are not done via hardware (circuit design). Adding signals has a place in designing a potentiostat, so that is the only math-related circuit we will explore.

In exploring this circuit, you should notice that it has many similarities to the voltage scaler. The non-inverting input is tied to ground and the inverting input has a feedback resistor tied to the output pin. The most significant difference is the presence of a *summing point* that ties together multiple sources. Each one of these sources can be viewed as a current (a combination of voltage and resistor), each of which is added together to generate the current that flows through the feedback resistor.

you try

1. Use the voltage/resistor values to determine the current from each of the three sources.
2. Confirm that the sum of the currents *entering* the summing point equals the current *exiting* this point.

The equation that relates the sources to the output voltage can be expressed as:

$$V_{output} = -R_{feedback} \sum i_{sources}$$

$$V_{output} = -R_{feedback} \sum_{n=1}^{\infty} \frac{V_n}{R_n}$$

Any number of sources can be added together, limited by physical and practical constraints. In cases where voltages are to be simply added, all of the resistance values (including $R_{feedback}$) are equal. It is possible to *subtract* by changing the sign of a voltage source.

you try

1. Modify the circuit to generate the average of the three sources.
2. Modify the circuit to generate a weighted average of the sources, where the 2nd source is weighted twice as much as the other two.

Tip: See a problem? Have a suggestion? Please raise an issue and share your thoughts there.

5.5 A review of Voltammetry fundamentals

Much of this section is paraphrased from David Harvey's Analytical Chemistry 2.1 text which can be accessed via Libre Text.

See also:

Be sure to check out section 11.1 of Harvey's book for a refresher on these topics.

Electrochemical techniques rely on the measurement of potential and current in order to understand chemical phenomena such as reactivity and concentration. Most, but not all, electrochemical techniques involve the study of the movement of electrons in an oxidation-reduction reaction. There are several important concepts in electrochemistry which are worth reviewing:

1. we cannot control simultaneously current and potential
2. the electrode's potential determines the analyte's form at the electrode's surface
3. the concentration of analyte at the electrode's surface may not be the same as its concentration in bulk solution
4. current is a measure of the rate of the analyte's oxidation or reduction
5. in addition to an oxidation-reduction reaction, the analyte may participate in other chemical reactions

Let's discuss how these five concepts are incorporated into voltammetry. The first item suggests that we must choose whether to control potential or current. As the name suggests, we *measure* the current (-ammetry) as a function of

potential in a voltammetric measurement. Because we control the potential, we control what is happening at the electrode surface. We can adjust the potential to force a species in solution[1] to become oxidized or reduced by adjusting the potential relative to the reduction potential of the solute. Note that we only have control of what happens *at the electrode/electrolyte interface*. The bulk of the solution is unaffected unless we have specifically designed the experiment to influence the entire solution. The fact that electrochemical reactions are happening *heterogeneously* in the solution has a significant impact on the relationships between current and potential that we will observe. The last two points are what make voltammetry interesting. Since current is a measure of the flow of electrons, more electron flow indicates that more oxidation or reduction (collectively, electrolysis) is happening. Therefore, the magnitude of the current informs us about the rate of a reaction. Lastly, we can use voltammetry to explore what - if anything - a molecule does before or after electrolysis.

Warning: See a problem? Have a suggestion? Please raise an issue and share your thoughts there.

5.6 Building the potentiostat

5.6.1 Introduction

The potentiostat board is referred to as the *bob173* and is currently in *beta* form. The design is based upon a potential control amplifier in the adder configuration which is described in full detail in Bard and Faulkner's Electrochemical Methods. [Bard2001] The main component for the potentiostat is a quad opamp (MCP6004). To complete the package, there is a 20 kOhm potentiometer for creating a voltage divider (needed for establishing a virtual ground), a three-terminal wire-to-board connector, a push button, two 1 KOhm resistors, a 10 nF capacitor and a 15 kOhm resistor. The schematic for the board is below:

The Eagle documents above can be used to create a layout that is compatible with the Featherwing format. Details for creating your own boards are forthcoming, but the project can be completed without using a featherwing (although it will not be as compact). In addition to a breadboard and some wires, you will need items listed in the table below.

Table 5.1: Bill of Materials

Part	Value	Package
CCV	10 nF	C-US075-032X103
D11	RST	TACT_PANA-EVQ
J1	Electrodes	SCREWTERMINAL-3.5MM-3
MCP6004	QUAD_OPAMPP	DIL14
MS1	FEATHERWING	FEATHERWING
PIR	20k	TRIM_US-RJ9W
PVG	20k	TRIM_US-RJ9W
RA0	1K	R-US_0204/7
RA1	1K	R-US_0204/7
RCV	10K	R-US_0204/7
RF	1K	R-US_0204/7
RIR	1K	R-US_0204/7
RLD	1K	R-US_0204/7
RST	RST	TACT_PANA-EVQ

[1] technically, or adsorbed onto the electrode surface, but we won't deal with adsorbed species here.

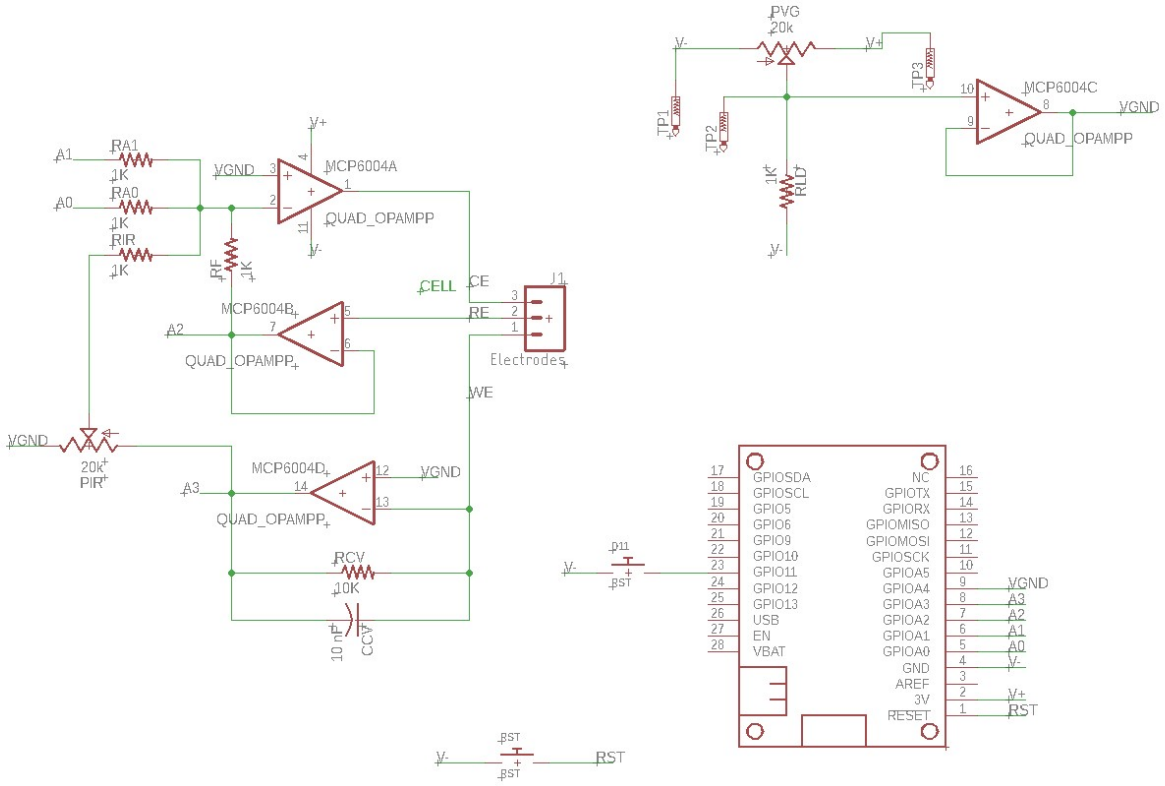


Fig. 5.1: Schematic for the bob173-gamma. You can `download the Eagle schematic and board files here`

5.6.2 Example project

The following tasks constitute a project to perform with the completed bob173 potentiostat. An example of how to control the potentiostat and how to analyze the data is found in *Example data*.

1. Install the software onto the microcontroller. Install Python on the computer and confirm that communication is established.
2. Calibrate the zero potential using a dummy cell (10 kOhm resistor). Adjust the virtual ground (pin A4) until the application of zero V results in zero current.
3. Confirm that the current axis is correct by obtaining a cyclic voltammogram of the dummy cell with the default parameters. The slope of the output should be equal to 1/R where R is the value of the resistor used.
4. Prepare a solution containing 0.1 M potassium nitrate along with approximately 5 mM potassium ferrocyanide. Obtain an electrochemical cell consisting of platinum wire for the counter electrode, a platinum disk (1-3 mm diameter) for the working electrode and an aqueous silver/silver chloride reference electrode. The working electrode should be polished with alumina slurry (or similar). The cell does not need to be degassed for this experiment.
5. Perform cyclic voltammograms in the scan rate region of 0.1 to 1 V/s. Optimize the starting/switching potentials as needed, and ensure that the best current-to-voltage resolution is obtained by swapping out the amplification resistor as needed.
6. Analyze the data to determine the diffusion coefficient of potassium ferrocyanide. Also explore the chemical and electrochemical reversibility of the reaction.

Warning: The operation notes below relate to *Mathematica* and are just notes to self at this point.

5.6.3 Operation notes

Using a terminal program or python to communicate with the FeAtHEr-Cm instrument. Mathematica can also be used via the ExternalEvaluate command.

```
(* start python session, send/receive data *)
s = StartExternalSession["Python"]
ExternalEvaluate[s, "s.write(b'cmd\\n\\r')"]
o = ExternalEvaluate[s, "
r = []
while s.in_waiting:
  r = s.readlines()
r"]
(* Convert data to useable form, removing linefeeds *)
p = FromCharactercode@DeleteCases[Normal@o,13|10,2]
(* Convert list of number strings *)
p2 = ImportString["["<>StringRiffle[Rest@p,","]<>"]","PythonExpression"]
```

Perhaps this is an easier approach

```
(* Start the session, call it s, and execute the initial commands *)
s = StartExternalSession["Python"],
ExternalEvaluate[s, "import serial, s = serial.Serial('COM3', 115200, \
timeout=1)"]
(* Create an external evaluate shortcut *)
ee[x_String] := ExternalEvaluate[s, x]
(* Check if data are in the buffer *)
ee["s.in_waiting"]
(* Send a command *)
ee@"s.write(b'get\\n\\r')"
(* Simplify command construction *)
st = StringTemplate["s.write(b'``\\n\\r')"],
ee@st["get"]
(* View the response *)
FromCharacterCode@DeleteCases[Normal@#, 13 | 10, 2] &@ee@"
r = []
while s.in_waiting:
      r = s.readlines()
r
"
(* Make that a function *)
res := FromCharacterCode@DeleteCases[Normal@#, 13 | 10, 2] &@ee@"
r = []
while s.in_waiting:
      r = s.readlines()
r
"
(* perform a sweep *)
ee@st["go"]
```

(continues on next page)

(continued from previous page)

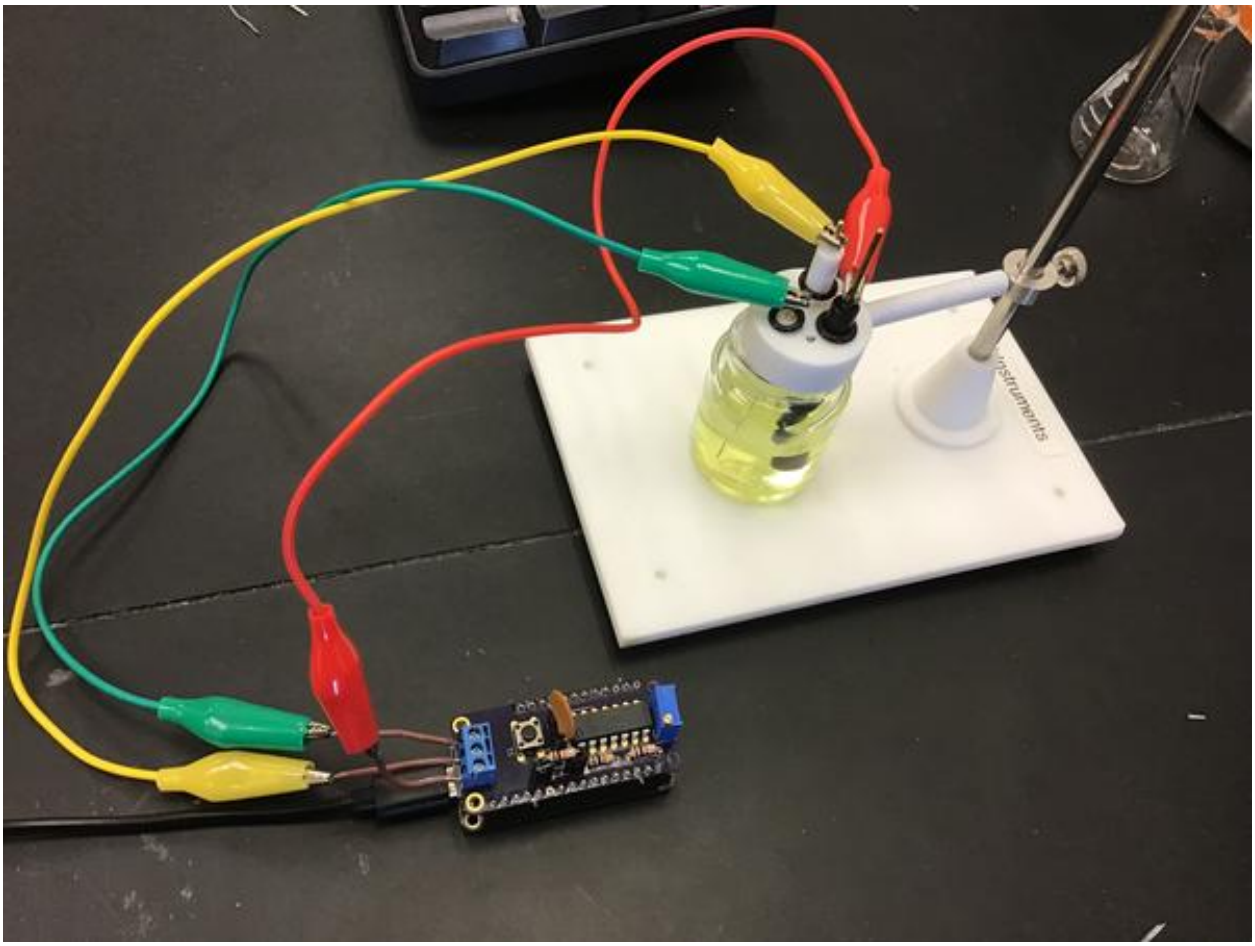
```
Pause[5]
out = res
(* A clunky way to conver the result *)
vals = ImportString["[" <> StringRiffle[Rest@out, ","] <> "]",
  "PythonExpression"]
```

Tip: See a problem? Have a suggestion? Please raise an issue and share your thoughts there.

5.7 Current operation

5.7.1 Experimental setup

Here's an overview of the potentiostat Featherwing, bob173-beta*[0], connected to an electrochemical cell.



Details on the design are provided elsewhere and this page will focus on some example usage. The setup is a 2 mm Pt disk working electrode with a Pt wire counter and Ag/AgCl reference. The solution is 4.81 mM potassium ferrocyanide

[0] The name is derived from the first potentiostat that I used as a graduate student at the University of Vermont: the PAR173 by Princeton Applied Research. While you may think that I've named the Featherwing after myself, it is actually the acronym for Brockport Original Builds.

in 0.10 M potassium chloride. The beta version of the board is pictured; however the gamma version was used for data collection.

5.7.2 Communication via python

All FeAtHEr-Cm instrumentation will communicate with the user via the serial console interface. The *easiest* way to do this is with a terminal program such as PuTTY but the main limitation with that approach is the ease with which experimental data can be transferred. Therefore, the next best approach is to interact with Python. In addition to the standard Python installation (I'm using 3.9.6), the serial communication module needs to be installed with *pip install pyserial*. Then, communication can be streamlined with the (to be documented) module *client.py*. In a python shell, one would type:

```
>>> import client
>>> f = client.fc('COM5')
>>> f.send('get')
>>> f.read()
```

The user sends commands using *f.send* and then reads the result using *f.read()*. It's a primitive strategy at present but the approach will allow for much flexibility, since many systems and programming languages are capable of communicating in this fashion. The potentiostat understands how to get and set the scan rate, potential limits and sampling frequency. It also needs to know the value of the resistor in the current to voltage converter.

```
>>> f.send('set SR 0.8')
>>> f.send('set SP -0.4')
>>> f.send('set SW 0.6')
>>> f.send('set EN -0.4')
>>> f.send('current 9080')
>>> f.send('go')
```

The last command tells the potentiostat to perform the cyclic voltammogram. The user needs to know how long the experiment will take because it is possible to read from the serial buffer before it is populated with the data. On the gamma version of the board, a red LED is lit when a CV scan is in process.

The *1* in the read command above tells the client software to strip text and format numbers for easy export (via *save*) to a CSV file. These CSV files can then be processed the same way you'd process any other data.

5.7.3 Example data

Above is an example cyclic voltammogram obtained on the bob173-gamma. The bob173 allows (rather, requires) the end user to manually select an appropriate resistor for the current to voltage conversion. In this case, the resistor is nominally 2 kOhm, which results in fairly noisy data. The peak current (~ 80 uA) corresponds to a voltage of 170 mV with this level of amplification, which is roughly 10% of the full scale (+/- 1.65 V) of the potentiostat.

As shown above, switching the resistor value in the current-to-voltage converter should have no impact on the overall shape and magnitude of the cyclic voltammogram so long as the resistance does not result in overloading the amplifier. Switching to a nominally 10 kOhm resistor results in a maximum voltage output of 730 mV. The noise in the CV with greater amplification is noticeably reduced. Note that when setting the current value, the measured resistance was used instead of the stated resistance (e.g. 9080 Ohm instead of 10 kOhm). The resistors used in this setup have a reported tolerance of 5%; therefore, it is important to use the actual resistance. Nonetheless, it can be seen that the 10 kOhm resistor results in a slightly larger current than the 2 kOhm resistor, suggesting that one (or both) of the resistor values are slightly incorrect. The percent difference between the two peak currents is 7%.

The figure above demonstrates the range of scan rates possible with the bob173-gamma. After setting the current-to-voltage resistors properly (the lowest value used is 2 kOhm and the largest is 22 kOhm), scan rates between 10 mV/s

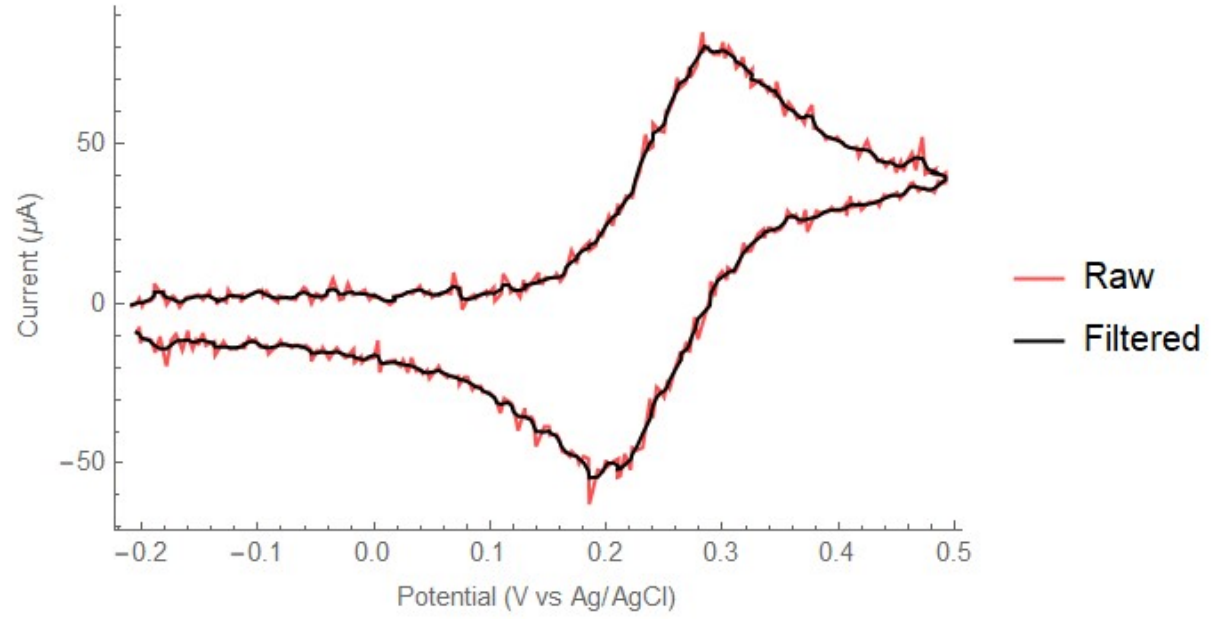


Fig. 5.2: CV of ferrocyanide, 0.5 V/s, 2 mm Pt diameter electrode. Anodic current is positive. Post processing (low-pass filter) performed in Mathematica.

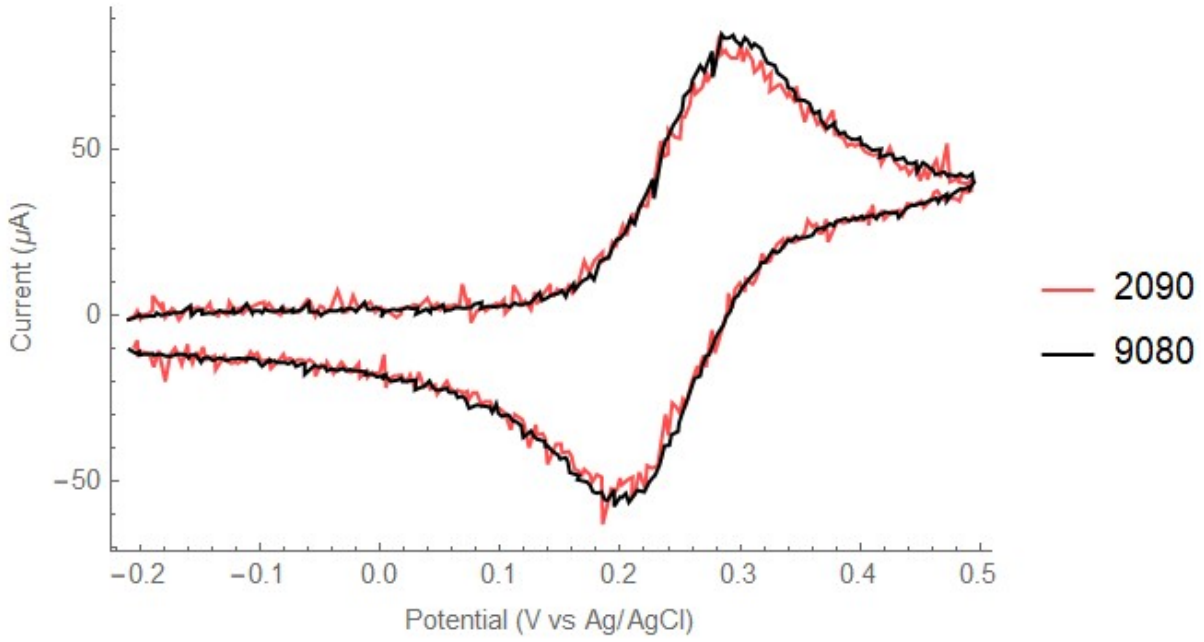


Fig. 5.3: CVs at 0.5 V/s with two separate current-to-voltage resistor values.

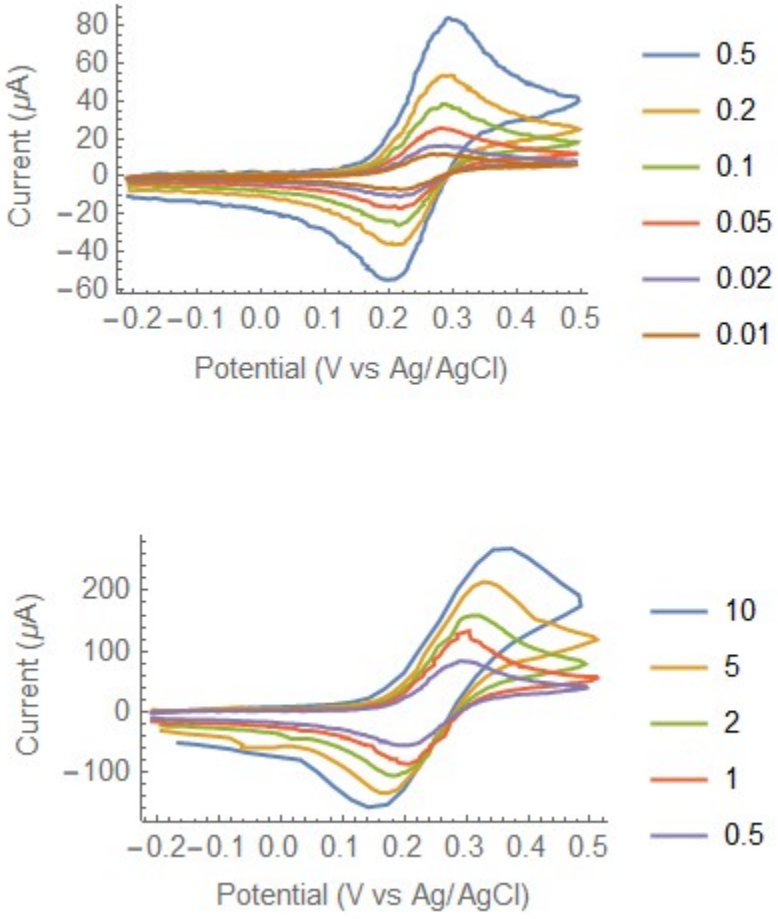


Fig. 5.4: Example of cyclic voltammograms from 10 mV/s to 10 V/s. All voltammograms have been post processed with a low pass filter.

and 10 V/s can be achieved. At the highest scan rates, sampling frequency becomes a limiting factor. The maximum sampling frequency is estimated to be 500 Hz and at 10 V/s with the potential range chosen here, the number of points in the CV is 48. At the lower end of the scan-rate range, the issue is memory allocation. Frequently, the low scan-rate scans fail with a memory allocation error. This issue is likely a bug in the code and the current workaround is to lower the total number of points collected in a single CV scan.

Note: The bob173-gamma does have an option for iR compensation; however, it was not tested in this set of data.

5.7.4 Example workup

Standard analysis of electrochemical data (workup) consists of exploring the change in peak current with scan rate, the increase in forward/reverse peak separations with scan rate, and the ratio of forward/reverse peak current as a function of scan rate. Combined, these three analyses provide a general overview of the electrochemical system and allows for preliminary assignment of electron-transfer mechanisms and rates.

The average of the forward and reverse peak potentials is the $E_{1/2}$ relative to the reference electrode used. This potential should stay constant throughout the range of scan rates employed for a redox system uncomplicated by additional chemical reactions. The average potential for these data is $0.250 \pm 0.004 V$ vs Ag/AgCl which is in reasonable agreement with other reports.†[0]

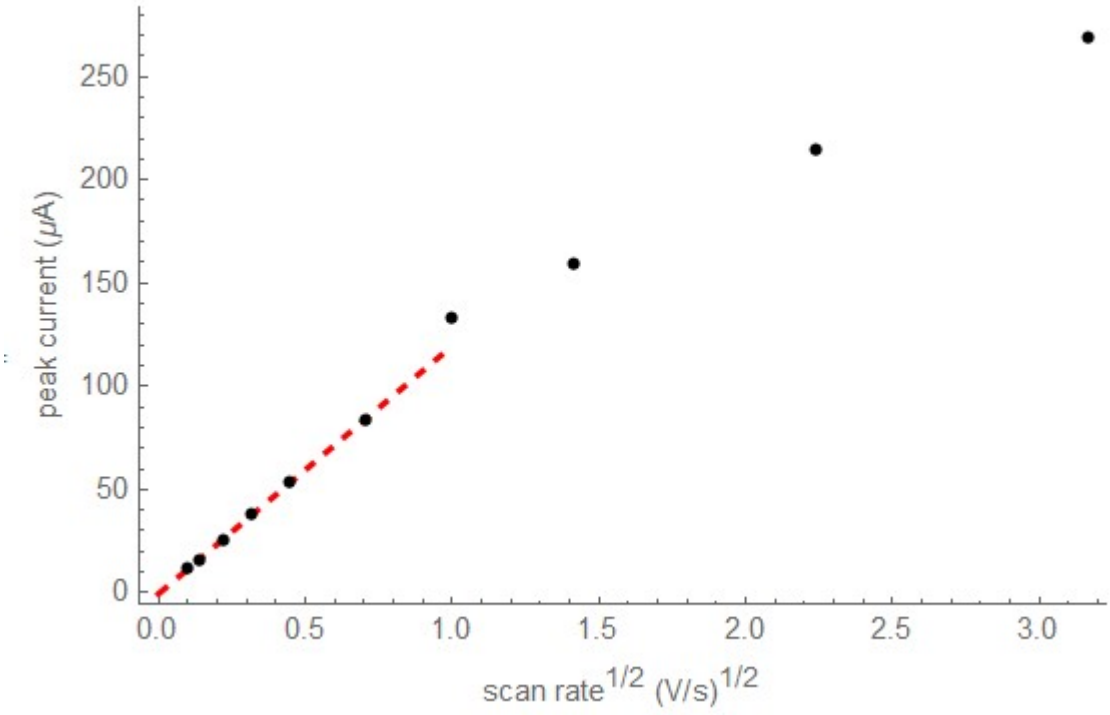


Fig. 5.5: Peak current as a function of the square root of scan rate. Red line is the best fit line through the data collected at scan rates below 1 V/s.

Above is a plot of the forward (anodic) peak current as a function of scan rate. At scan rates below 1 V/s, the plot is linear, as expected from theory. The deviation from linearity at higher scan rates could be due to ohmic drop, but it

[0] Get this citation.

cannot yet be ruled out that instrumental factors such as sampling frequency and incorrect current-to-voltage settings are contributing factors. Nonetheless, the lower scan rate data can be fit to the Randles-Sevcik equation in order to extract an estimate of the diffusion coefficient.

$$i_p = 2.69 \times 10^5 n^{\frac{3}{2}} AC\sqrt{D\nu}$$

In the equation above, i_p is the peak current, n is the number of electrons in the redox process, A is the electrode area in square cm, C is the concentration in mol/mL, D is the diffusion coefficient in cm/s and ν is the scan rate in V/s. For the ferrocyanide data collected here, the best fit line through the low scan rate data has a slope of 120 $\mu A s^{1/2} V^{-1/2}$ which corresponds to a diffusion coefficient of 8.7×10^{-6} cm/s. The reported value for the diffusion coefficient of ferrocyanide in potassium chloride is 6.7×10^{-6} cm/s.‡[0] The error is reasonable given that the electrode diameter in this experiment is only known to the nearest mm. The experimental error (30%) is much larger than what can be attributed to the estimated errors associated with the current-to-voltage conversion (7%) or the lack of temperature control (< 1 %).

The oxidation of potassium ferrocyanide is generally considered fast, and therefore the separation between the forward and reverse peaks should be approximately 59 mV and remain independent of scan rate.[159] In the figure below, ΔE_p is plotted as a function of scan rate up to 1 V/s. At very low scan rates, the peak separation is independent of scan rate and approximately 72 mV. As the scan rate increases, so does the ΔE_p in a non-linear fashion. It is likely that ohmic losses are playing a role in the peak separation increase.

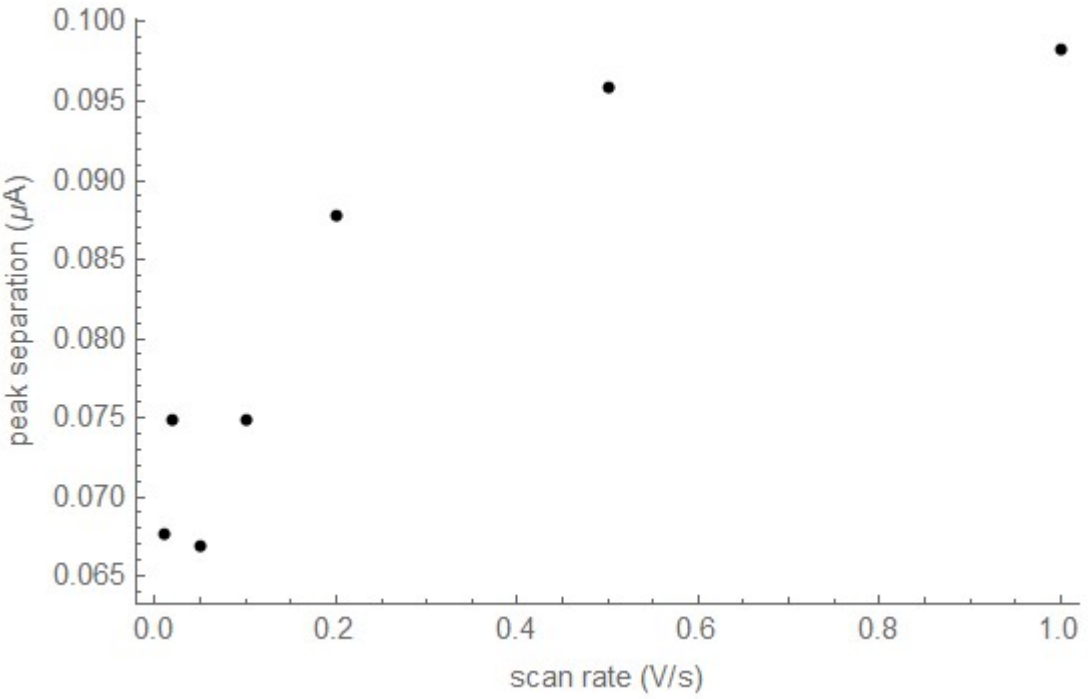


Fig. 5.6: Forward/reverse peak separation (ΔE_p) as a function of scan rate.

Lastly, the ratio of the reverse and forward peak currents provides insight into the chemical reversibility of the electron-transfer process. Both ferro- and ferricyanide are chemically stable under the conditions used in this experiment, so it is expected that the peak current ratio would be close to 1 and independent of scan rate. Using Nicholson's empirical

[0] See for example, https://www.asdlib.org/onlineArticles/elabware/kuwanaEC_lab/PDF-19-Experiment1.pdf
[159] I do not have a good citation for the heterogeneous electron transfer rate of ferrocyanide.

method for assessing chemical reversibility requires the current at the switching potential.

$$\frac{i_{rev}}{i_{fwd}} = \frac{i_{rev}}{i_{fwd}} + 0.48 \times \frac{i_{switch}}{i_{fwd}} + 0.086$$

The peak ratio for the entire scan rate regime investigated here is 0.972 ± 0.02, which is expected. Note that due to the way data are acquired by the bob173, the sampling frequency may "miss" the switching potential. upon close inspection of the raw data, the recorded data point closest to the switching potential was on average 3 mV less than the target value.

5.7.5 Discussion

The results presented here demonstrate the ability to acquire electrochemical data on an ideal redox system with sufficient quality to perform standard analyses such as estimates of the diffusion coefficient and assessment of the chemical and electrochemical reversibility of the redox system. Post processing of the data is required due to limitations in the analog to digital (ADC) circuitry of the microcontroller used (the M4 Express) as well as the absence of any hardware based noise reduction strategies. While it would be possibile to eliminate these issues through additional hardware, it would defeat the pedagogical and price-point objectives of the project. Presently, the bob173-gamma can be created for under $30. The only additional components not considered in that price are a USB cable and a computer with a USB connection capable of running Python. A Raspberry Pi computer can be purchased for under $100 (including power and SD card storage but *not* considering keyboard/mouse/monitor peripherals) which is capable of controlling the bob173 via Python as well as perform the post-processing/visualization using Mathematica. Mathematica is not required for the rudimentary processing and visualization steps, which could also be completed in Python by expanding upon the client.py package.

Additionally, the bob-173 provides some key learning opportunities. The three main points of understanding are: the design of a voltage divider and the need for buffering; adding signals together using an operational amplifier; and amplifying and converting current from a physical measurement into a voltage that can be read by a computer. Completion of this module will include lesson plans that explore simulated (via the Falstad online circuit simulator) and hands-on (via the bob173) activities.

5.7.6 client.py

Here's the module run on the client computer to interface with a FeAtHEr-Cm instrument.

```
import serial
import csv
from time import sleep

class fc():
    def __init__(self, *argv):
        self.s = serial.Serial(argv[0],11500,timeout=1)

    def send(self, cmd):
        self.s.write(f'{cmd}\n\r'.encode('utf-8'))

    def read(self, *argv):
        r = []
        while self.s.in_waiting:
              r = self.s.readlines()
        if len(argv) > 0:
            r = clean(r)
        return r
```

(continues on next page)

(continued from previous page)

```
    def dia(self, cmd, duration = 1):
        self.send(cmd)
        sleep(duration)
        return self.read()

def clean(response):
    out = []
    for i in response:
        try:
            out += [ list(map(float,i.strip(b'][\r\n').split(b', '))) ]
        # Silently ignore non-numbers
        except ValueError:
            pass
    return out

def save(filename, data):
    with open(filename, mode='w', newline='') as f:
        fw = csv.writer(f, delimiter=',')
        for i in data:
            fw.writerow(i)
```

Tip: See a problem? Have a suggestion? Please raise an issue and share your thoughts there.

CHAPTER

SIX

SPECTROSCOPY

Designing turbidimetry/nephelometry instrument

Many instrument design labs explore colorimetry or spectrophotometry, and the intention here is not to reinvent the wheel but to add to it. We will explore some aspects of instrument design such as signal amplification, non-chemical deviations from Beer's Law, signal transduction and task-specific design.

Outcomes

- A spectroscopic instrument that uses an LED as a source and detector
- Circuit design for signal amplification, passive and active filtering
- 3D printing of a sample holder
- Comparison of home-built vs commercial instrument performance
- Exploring a relevant project idea.

Project timeline

1. Intro to turbidimetry/nephelometry - building an instrument that can measure the clarity of water
    - Watch intro videos and read paper
2. Setting goals for instrument design and introduction to CAD with OpenSCAD
    - Become familiar with OpenSCAD; develop idea for instrument
    - Download from their website
    - Which filter will be used (on board active does additional gain)
    - Source color (EPA uses White, we'll use red or green)
    - Sample holder design
    - Phone wire for easy wiring
3. New circuit components: LEDs, transistors and filters
    - Do integrated simulation activities
4. Building the instrument
    - Solder instrument
5. Creating 3D printed prototypes of the instrument
    - 3D printing
    - Create a device that can hold two LEDs at a fixed angle and incorporate the sample cell.
6. Modifying the prototype

    - 3D printing
7. Measurements 1: Instrument setup
8. Measurements 2: Initial data acquisition
9. Measurements 3: Fixing mistakes, testing reproducibility and robustness
10. Presentation

6.1 Turbidimetry Backround

- Turbidity background
  - Video from Cleveland water department
  - Video from Endress-Hauser
- Paper on measurement challenges
  - Direct link to paper (requires subscription)
  - Abstract (includes citation information)

Tip: See a problem? Have a suggestion? Please raise an issue and share your thoughts there.

6.2 Turbidimetry instrument design

In designing our instrument, we will build upon several aspects of electronics introduced in the electrochemistry section. You should be familiar with operational amplifiers and passive components. Recalling the types of operational amplifier circuits may be helpful, but is not necessary.

Discuss a block diagram of the spectrometer and specify that we need to discuss LEDs, transistors and filters. We'll then conclude with the schematic.

Chapter sections

6.2.1 Essentials of transistor switching

Better outline?

1. Define the problem: we want to be able to turn our source on and off from software, and in principle we can do that with a GPIO pin. However, we also want a large amount of current so that our source is bright, and that presents a problem because the GPIO pins can only source a limited amount of current (for the M4 express, it is 10 mA max and 7 mA recommended). How can we make a switch and avoid damaging the microcontroller
2. An electronic component called a transistor can be used as a switch. A good resource can be found here, but the short story is that applying a small amount of current to the base of a transistor allows a larger flow of current to move from its collector pin to its emitter pin.
3. What is "a small amount" of current from the transistor's standpoint?
4. What do I need to know as a practical builder?

- Using a transistor switch

  - When using microcontrollers, we have to be careful with how much current is drawn from a pin (https://tinyurl.com/ye4uay78) use of transistor helps us turn a device on and off which little current drawn from the switch itself.

..note:: Please visit this link to complete this activity

- Before looking at the transistor switch, let's focus on the LED circuit. Mouse over the source and note the max voltage. There is one resistor in the circuit, so the current should be determined using Ohm's Law. Turns out this isn't quite right. Mouse over the resistor and note that the voltage drop is only 1.55. Therefore, the LED is also acting like a resistor of sorts. If the total voltage is 3.3 V and there is 1.55 dropped across the resistor, how much is dropped across the LED? This amount is not fixed, and it's impractical to find the value mathematically, so we use a ballpark figure of an LED dropping about 1.8 V. More importantly is that we want to ensure that there is no more than 20 mA of current flowing through the LED, or else it will be damaged. So finding an optimal resistance requires
  - Finding remaining resistance (3.3 V - 1.8 V = 1.5 V)
  - Finding a resistor that gives us between 16-18 mA with that voltage (1.5/0.016 or 1.5/0.018 = 94 to 83 Ohm). Rounding to the next highest resistor value (100 typically) gives us a good starting point.
- The problem we note here is that the microcontroller can only deliver 8 mA from its digital IO pins but we want to deliver twice that amount to light the LED. The transistor will help with this problem.
- One of the uses of a transistor is as a digital switch. A transistor contains three connections, a collector, base, and emitter. The base can be thought of as the toggle which turns on/off the flow of current from the collector to the emitter.
- To turn on a transistor switch requires a certain amount of current flowing into the base. The actual amount of current depends on the type of transistor, but it will roughly be the amount of current we want to flow through our device (16 mA for the LED) divided by 100.
- Since the digital IO pins deliver 3.3 V and we want 160 uA to flow, we need a resistor with a value of about 20 K to apply the correct current to the transistor base.
- Note that the numbers don't quite add up, and that is because the transistor experiences a voltage drop as well, and we haven't taken that into consideration. Again, this value varies based on the type of resistor and the amount of current flowing, but we can use a ballpark figure of 0.6 - 0.7 V. So revisiting the entire switch circuit:
  - The signal we are switching is Vt - Vr1 - Vled - Vtrans = 0
  - The switch is Vt - Vr2 - Vtrans = 0
- If we want approximately 16 mA to flow through the LED, we need a resistor that solves 3.3 - (0.016) R - 1.8 - 0.65 = 0
- Here is a good resource for understanding how a transistor can be used as a digital switch.

Tip: See a problem? Have a suggestion? Please raise an issue and share your thoughts there.

6.2.2 Exploring active/passive filters

Ideally, one should design an instrument such that data are collected in the absence of noise. Unfortunately, this situation can rarely - if ever - be realized. Therefore, it is necessary to filter noise from an instrument's signal. The theory, design and application of filters can become very complex, and our treatment here is design to introduce the concept and focus on one aspect of filtering that is relevant to the turbidimetry project.

The most rudimentary filter can be formed with a capacitor and resistor in series. Since these components do not require power in order to operate, an RC filter is referred to as passive.

Note: Visit https://tinyurl.com/yh478el3 to view this interactive image

---

In this figure, The voltage source consists of an AC component with an amplitude of 0.5 V and a DC component with a magnitude of 0.5 V. The signal is passed through a resistor then capacitor. The output signal is measured at the point *in between* the two components. It is important that the components are in the designated order.

At a frequency of 10 hz, the output signal nearly mimics the input signal. The peak voltage should be around 1 V. Increase the frequency to 100 hz and you'll notice that the output no longer looks like the input. The peak voltage is around 579 mV. Increasing the frequency further to 1000 hz and the output loses nearly all of the AC component and what remains is the 0.5 V DC signal. The high frequency component has effectively been removed while the DC component remains; hence, the name of this circuit is a *passive low pass filter*.

A more sophisticated, yet accessible filter involves the use of an operational amplifier, and is therefore referred to as an active filter. The operational amplifier looks to be in a buffer-like configuration with the inverting input tied to the output. A capacitor is in a feedback loop with the non-inverting input. The two pairs of resistors and capacitors share the same values and the resulting output has a slightly better performance than the passive filter. As we will explore more fully, the added complexity of the active filter results in overall better performance.

Note: Visit https://tinyurl.com/yfnc3kgh to view this interactive image

---

In the simulation below, you will find both of these filters, along with an operational amplifier in an adder configuration that combines three sources: a DC signal; an AC signal; and white noise. Perform the tasks noted below the simulation. By the conclusion of this activity, you should be able to compare and contrast the two types of filters and understand their performance limitations. You will also learn some terminology (dB, transition band) that are used in evaluating filters.

Note: Visit https://tinyurl.com/yfbh53l2 to perform the activity described below.

- [Do] Identify the three parts of this circuit: the source adder; the passive filter; and the active filter. The three components are separated by a switch, which can be clicked to direct the source to either of the filters.
- [DO] Note the resistor and capacitor values in the *filter* parts of the circuit. Are they the same or different?
- [Do] Determine how to change the input by using the sliders for noise and DC signal. The frequency is set by right clicking on that source component and selecting edit.
- [Do] When finished with introducing yourself to the simulation, refresh the page to reset the simulation to its default values.
- [DO] Note the different in maximum voltage between the passive filter output and the source.
- [PREDICT] Based on what you have learned earlier in this section, what do you expect will happen to these values when the frequency is increased to 400 hz?

- [Explore] Investigate qualitatively how the maximum voltage of the output compares to the maximum voltage of the input as the frequency is changed from 40 to 400 to 4000 hz. Compare the responses of both filters.
- [REPORT] What is the ratio of output voltage to input voltage for each filter at each of the three frequencies? Which filter does a better job at eliminating the AC signal?
- [REPORT] Was the prediction you made correct? If not, identify what was incorrect about it and what misconceptions you may have had.
- [EXCEL] Perform a more quantitative exploration of the output voltage by creating a spreadsheet with the maximum output voltage of each filter at the following frequencies: 1, 2, 5, 7, 10, 12, 15, 17, 20, 50, 70 and 100 Hz. Insert a column that calculates the log of the frequency
- [PLOT] Create a single plot with the Voltage as a function of the log of frequency. Include both markers and connected lines in this plot.

The plot you have created describes the performance of the filter as a function of frequency. It has three regions: one where the signal is essentially unaffected; one where the signal is completely attenuated and a transition region referred to as the *transition band*. Many applications require that the transition band be as narrow as possible in order to avoid some undesired noise from leaking through the filter.

Filter performance is typically reported using the unit decibels (dB), which is calculated using :math`dB=20 log(frac{V_{out}}{V_{in}})`. We will use this value to define the transition band of each filter and define this region (somewhat arbitrarily) as the region between -3 dB and -20 dB.

- [CALCULATE] Determine the output voltage at -3 dB and -20 dB.
- [CALCULATE] the *percent* of the AC signal that is filtered out at -3 dB and -20 dB.
- [DO] Find the frequency at which the output is -3 dB and -20 dB for each of the filters. The difference between these values is the width of the transition band.
- [REPORT] Report the width of the transition bands for each of the filters and comment on which filter performs better.

One other characteristic of a filter is its *cutoff frequency* which is equal to $\frac{1}{2\pi RC}$.

- [CALCULATE] Calculate the cutoff frequency $\frac{1}{2\pi RC}$ for the filter. Note that since the same values of R and C are used, this characteristic is the same for both the active and passive filters.
- [DO] Find the maximum voltage at the cutoff frequency for each of the filters.
- [CALCULATE] Determine the filter magnitude (in dB) of each filter at the cutoff frequency.

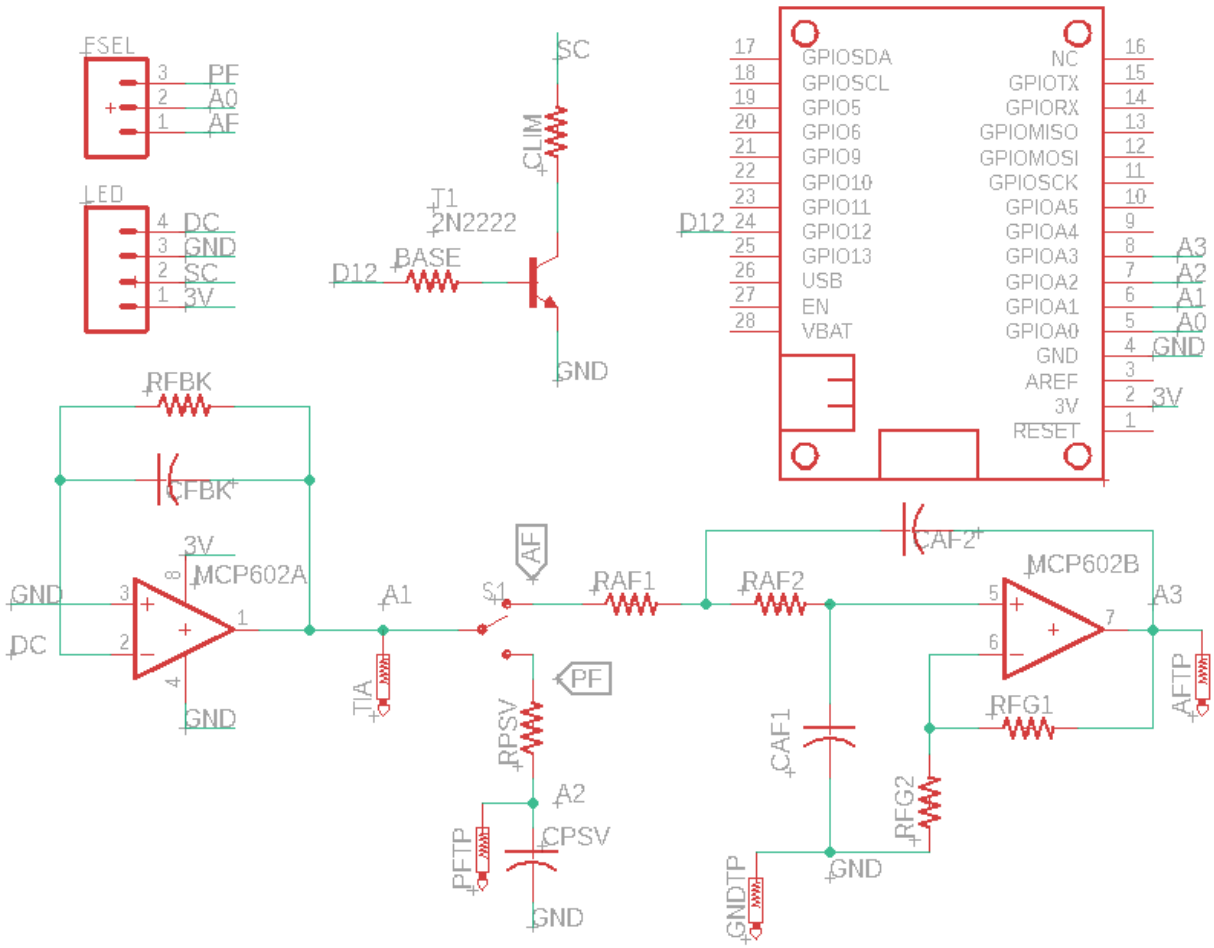
Tip: See a problem? Have a suggestion? Please raise an issue and share your thoughts there.

6.2.3 Fundamentals of light emitting diodes (LEDs)

Videos worth watching: This playlist just the following videos in this order: 3, 1, 2, 5 and 8. Questions to ponder before, during and after videos are coming soon.

Tip: See a problem? Have a suggestion? Please raise an issue and share your thoughts there.

6.2.4 Instrument schematic



Schematic description

The btm-100 consists of a dual operational amplifier (MCP602) that serves the role of current to voltage converter (Op amp A) and an active 2-pole filter (Op amp B) and a transistor to control power to an LED source. In addition, the board contains a passive filter which can be accessed via a tactile switch. The passive components (capacitors and resistors) that set the signal amplification and filter cutoff frequencies are designed to be adjusted by the end user.

The default configuration has the following passive components

- RFBK: current to voltage converter feedback resistor, 10 MOhm
- CFBK: current to voltage converter feedback capacitor, 10 pF
- BASE: current limiting resistor to transistor base, 10 kOhm (should not be changed)
- CLIM: Current limiting resistor for source LED, 330 Ohm
- RAF1 and RAF2: Resistors in active filter, 33 kOhm (should be identical)
- CAF1 and CAF2: Capacitors in active filter, 1 uF (should be identical)

- RFG1 and RFG2: Gain resistors in active filter, 1 kOhm (Can leave RFG2 open and short RFG1 for an active filter with no amplification)
- RPSV: Resistor in passive filter, empty
- CPSV: Capacitor in passive filter, empty

Important relationships

In the section on exploring active and passive filters, we observed the relationship between output attenuation and input frequency. The relationships can be described mathematically with the equations below. It is not important to memorize these equations or know their derivations.

For the passive filter, the output is related to frequency through the following equation:

$$\frac{V_{out}}{V_{in}} = \frac{1}{\sqrt{1 + (2\pi f RC)^2}}$$

The active filter relationship is similar, but note that the denominator is not quite the same:

$$\frac{V_{out}}{V_{in}} = \frac{1}{1 + (2\pi f RC)^2}$$

Lastly, since the amplitude of an AC frequency is an energy (in volts), the definition of decibels is:

$$dB = 10 \log \frac{V_{out}}{V_{in}}$$

Exploring the schematic

1. Estimate the current flowing through the source.
2. How much current is applied to the base of the transistor in order to turn the digital switch on?
3. Calculate the cutoff frequency for the passive and active filters with the default component values.
4. The current to voltage converter behaves as a filter with the same performance as the passive filter. What is the cutoff frequency (-3 dB) for the current to voltage converter with the default component values?
5. When an LED is exposed to light, current flows in the *opposite* direction (cathode to anode). Notice which end of the LED is connected to the op amp and predict whether the op amp will perceive this current as positive (flowing towards the input) or negative (flowing away from the input).
6. Given the default components and the maximum voltage allowed for the M4 Express, predict the maximum current that can be detected from the LED without saturating the amplifier.
7. Recall that the M4 Express has 12-bit resolution (2^{12}). What is the smallest current that could be detected by the amplifier, assuming no noise.

Tip: See a problem? Have a suggestion? Please raise an issue and share your thoughts there.

Tip: See a problem? Have a suggestion? Please raise an issue and share your thoughts there.

6.3 Designing an instrument

- brief introduction to CAD with openscad
- create a simple sample holder with source/detector parts

Tip: See a problem? Have a suggestion? Please raise an issue and share your thoughts there.

6.4 Projects

- Settling rate of particles
- Water analysis

Tip: See a problem? Have a suggestion? Please raise an issue and share your thoughts there.

CHAPTER

SEVEN

BIBLIOGRAPHY

Use the browser’s back button to return to your last location.

CHAPTER

EIGHT

BLANK DOCUMENT

Nothing to see below me.

CHAPTER

NINE

THERMAL CONDUCTIVITY DETECTOR

A thermal conductivity detector (TCD) finds use in analysis of gases. Additionally, it uses an important, fundamental circuit design called a Whetstone bridge. Therefore, it is an interesting component to add to our toolbox of instrumental approaches to chemical analysis.

CHAPTER

TEN

GAS CHROMATOGRAPH

A gas chromatograph is one of the more common instruments in chemistry laboratories. Commercial instruments use high pressure gases, autosamplers, and expensive columns. The instrument designed here will be fairly limited in scope, but will allow for an exploration of the concepts of chromatograph design and serve as a platform for some types of interesting chemical analyses.

CHAPTER

ELEVEN

HOW TO CONTRIBUTE TO FEATHER-CM

This project is designed to be open and accessible. Furthermore, it is anticipated that consumers of the content will contribute suggestions and corrections so that the materials can continually be improved. Here are the initial steps for participating in the FeAtHEr-Cm community.

- Create a Github account at https://github.com/.
- Navigate to the repository for the course at https://github.com/bobthechemist/feathercm.
- Use the *issues* tab to view open issues, raise concerns and propose suggestions.

Fig. 11.1: Go to the issues tab to view and raise issues.

- If you are familiar with how Github works, feel free to fork FeAtHEr-Cm so that you can create your own pull requests. (And if you have no idea what that sentence means, don't worry about it.)

To do Extracting relevant components of https://learn.adafruit.com/contribute-to-circuitpython-with-git-and-github/overview

CHAPTER

TWELVE

CIRCUIT SIMULATOR

This circuit simulator can be found at www.falstad.com

CHAPTER

THIRTEEN

FEATHERCM PACKAGE

13.1 Submodules

13.2 feathercm.base module

feathercm.base.**analogRead**(*pin*, *n=8*)
Reads an analog pin and adjusts the value according to a linear calibration. Allows rapid signal averaging and returns the value as 12-bit. Currently, errors seem to be on the order of +/- 5 mV.

feathercm.base.**analogWrite**(*pin*, *value*)
Writes to an analog pin after correcting for calibration curve adjustments and switching value from 12 bit to 16 bit.

feathercm.base.**analogWriteValue**(*x*)
Returns the 'correct' 12-bit integer value that generates the expected voltage for the input value. To perform this calibration, measure the actual voltage (with DVM) at A0 with respect to A0 setting. Best results are obtained for voltages under 2.0. Find the absolute error and convert that to an A0 adjustment value. Find the best fit line between A0 (x axis) and A0 adjustment value (y axis).

Note: the M4 cannot handle rail-to-rail and output is limited to 0.1 to 3.2

feathercm.base.**analogWriteVoltage**(*pin*, *value*, *vground=True*)
Writes to an analog pin the requested voltage after correcting for calibration curve adjustments. Will correct for virtual ground if requested.

feathercm.base.**commandValidQ**()
Checks if the latest command is valid

feathercm.base.**init**(*featherwing*)
Starts feather-cm

feathercm.base.**initFunc**(**argv*)
Default initialization function

Featherwing module should override this function, although it doesn't have to.

feathercm.base.**listen**()
Starts listening to the serial line

If a command is present, read the line, check if it is a valid command, execute the command and return the response.

feathercm.base.**rebootFunc**(**argv*)
Reboots the instrument

feathercm.base.**respond**(*response*)
Posts a response

feathercm.base.**sba**()
Checks if a command is available on the serial line

feathercm.base.**toVoltage**(*reading*, *vground=True*)
Convert a 12-bit reading into a voltage. Setting vground = False will return 0 to 3.3 V, a true value will return -1.65 to +1.65 V

13.3 feathercm.data module

13.3.1 feathercm.data

data class for FeAtHEr-Cm

class feathercm.data.**data**(**argv*)
Bases: object

Need to determine if these functions belong here or in base. Currently in both places

append(*item*)
Appends item onto val

fromRaw(*which*, *value*)
Converts a reading into a value

processData()
Converts reading data into values

toCurrent(*value*)
Convert raw value to a current

toInvertedVoltage(*value*)
Convert raw value to a voltage

toReading(*voltage*)

toSelf(*value*)
Convert raw value into a time, possibly useful for prefix conversion

toVoltage(*value*)
Convert raw value to a voltage

13.4 feathercm.echem module

13.4.1 feathercm.echem

This module contains functions and classes used in electrochemical measurements

class feathercm.echem.**base**(*control*, *voltage*, *current*)
Bases: object

Base class for electrochemical measurements

controlOff()

controlOn()

scanrateCheck(*scanrate*)

voltageCheck(*voltage*)

feathercm.echem.**getFunc**(**argv*)

feathercm.echem.**goFunc**(**argv*)

feathercm.echem.**initFunc**(**argv*)

feathercm.echem.**setFunc**(**argv*)

class feathercm.echem.**sweep**(*control*, *voltage*, *current*)
Bases: *feathercm.echem.base*

doSweep()
Next gen CV sweep

getParameter(*param*)

setParameter(*param*, *value*)

setSamplingFrequency()
Calculates the sampling frequency that results in 1024 points for the CV

feathercm.echem.**testFunc**(**argv*)

13.5 feathercm.settings module

feathercm.settings.**makeCommandDict**(*commands*, *functions*)
Create a dictionary from arguments

feathercm.settings.**mergeDict**(*newDict*)
Merges the passed dictionary into the global dictionary. There is no checking of key overlap at present. Currently using this 'feature' to overload the base initialization command. Not sure if this is necessary.

feathercm.settings.**setCurrentMultFunc**(**argv*)

feathercm.settings.**setDataSizeFunc**(**argv*)

13.6 feathercm.spec module

feathercm.spec.**getFunc**(**argv*)
Get current value for a parameter. Do not include a parameter for a list of possibilities.

feathercm.spec.**initFunc**(**argv*)

feathercm.spec.**readFunc**(**argv*)

feathercm.spec.**setFunc**(**argv*)
description

feathercm.spec.**sourceFunc**(**argv*)

feathercm.spec.**timeSeriesFunc**(**argv*)
Collects a time series of measurements.

feathercm.spec.**toVoltage**(*value*)

13.7 Module contents

CHAPTER

FOURTEEN

EXAMPLES

These are some design examples - or how various RST constructs are rendered.

14.1 Installation/Usage:

I'll get there eventually

14.2 Some other example

$\alpha > \beta$

$$n_{\text{offset}} = \sum_{k=0}^{N-1} s_k n_k$$

$$H2O- > H2 + O2$$

See also:

this is a simple **seealso** note.

Note: This is a **note** box.

Warning: this is a warning to you.

Sidebar Title

Optional Sidebar Subtitle

Subsequent indented lines comprise the body of the sidebar, and are interpreted as body elements.

14.3 Really cool embed

Wouldn't it be nice if you could embed code for dynamic web-based textbooks?

CHAPTER

FIFTEEN

INDICES AND TABLES

- genindex
- modindex
- search

BIBLIOGRAPHY

[Bard2001] Bard, Allen J., and Larry R. Faulkner. Electrochemical methods : fundamentals and applications. Hoboken, NJ: John Wiley & Sons, Inc, 2001. Print.

[Enke1971] Enke, C. G. *Anal. Chem.* **1971**, 43, 69A. Link

[Rayson2004] Rayson, G. D. *J. Chem Ed.* **2004**, 81, 1767. Link

PYTHON MODULE INDEX

f

INDEX

V