Baylor University

Department of

Electrical and Computer Engineering

BME/ELC 4372

Bioinstrumentation Laboratories

Keith Schubert
Professor
Department of Electrical and Computer Engineering
Baylor University

Contents

0 Raspberry Pi **3**
- 0.1 Network . . . 3
- 0.2 GIT . . . 3
  - 0.2.1 Some commands for reference . . . 4
- 0.3 Das Blinken LED . . . 4
- 0.4 Das Blinken LED II . . . 6
- 0.5 Das Blinken LED III . . . 7

1 Electromyography (1A) **9**
- 1.1 Biopotentials . . . 9
- 1.2 Setup . . . 9
  - 1.2.1 Olimex EKG/EMG Shield . . . 10
- 1.3 General Advice . . . 10
- 1.4 Test . . . 10

2 Electro-Kardio-Gram (1B) **13**
- 2.1 Test . . . 13

3 Op Amps **15**
- 3.1 MCP 3008 . . . 15
- 3.2 Follower . . . 16
- 3.3 Non-Inverting Amplifier . . . 17
- 3.4 Inverting Amplifier . . . 17

4 Build Your own EMG (2A) **19**
- 4.1 Wiring the Circuit . . . 19
- 4.2 Graphing . . . 21
- 4.3 Optional Reference: Understanding the AD8221 Breakout . . . 21

5 Pulse Oximeter (2B) **23**
- 5.1 Background . . . 23
- 5.2 First Method . . . 25

6 One Wire Thermometer **27**
- 6.1 Assemble Circuit . . . 27
- 6.2 Add One Wire Support . . . 27
- 6.3 Test It . . . 28
- 6.4 Code It . . . 28

7 Electro-Encephlogram **31**
7.1 Installing in MatLab 31

Lab 0

Raspberry Pi

Over the course of this semester we will be building a variety of medical and biological sensors. We will be using the Raspberry Pi as our microcomputer to control them, because of its ease of use, large number of IO pins, and tons of example code to build on. In this lab we will be introducing the Raspberry Pi and how to use it. I am going to try to do most things in Python, due to its simplicity and extensibility, but I can't guarantee we won't have to do a little programming in another language.

0.1 Network

Your Pi either has a usb WiFi connector or built in WiFi. In both cases you need to log into the network using the network gui. Select it from the top right of the Pi's desktop, select the network then enter the login information.

0.2 GIT

One thing I really want to bring to your attention is the use of **git**. Git is a version control system (VCS), that was designed by Linus Torvalds to handle the development of Linux. I will be maintaining a git repo at github, which means you will be able to clone it and do a pull any time you want to update it. You do not have to use git, it just saves time and is a good skill to know for industry. Open up a terminal (upper left of the desktop icon is black and blue with a white `>_`). Type the following:

```
ls
```

If you don't see a directory named `code` then you will need to make it by typing:

```
mkdir code
```

Now you need to change to the code directory and get the git repo with starter code.

```
cd code
```

```
git clone https://github.com/BaylorBioMedicalEngineering/BME-4372-BioInstrumentation-labs.
git1
```

```
cd labs
```

```
ls
```

0.2.1 Some commands for reference

To update from the main repository, just do a pull

```
git pull
```

More sample code can be found by:

```
cd  /code

git clone http://github.com/adafruit/Adafruit-Raspberry-Pi-Python-Code.git

cd Adafruit-Raspberry-Pi-Python-Code
```

0.3 Das Blinken LED

- Raspberry Pi (with keyboard, mouse, monitor, cobbler, and breadboard)
- 220Ω - 330Ω Resistor
- LED

First you need to assemble the circuit. Hook up a wire from a gpio pin to the anode (long leg) of the diode, then connect the resistor to ground. The resistor limits the current flow. The Pi will output 3.3V and the diode will cause about a .7V drop resulting in a 2.6V drop left over. The remaining 2.6V flowing through around 220Ω to 330Ω will result in around 10mV, which is enough to light the LED but not cause problems sinking or sourcing the current for the Pi. Generally be careful with more than 20mV - 30mV for an IC.

Boot the Pi by plugging it in. When the desktop appears, launch a terminal either from the menu or the terminal button. Type the following line to edit the code to turn on and off with a timer.

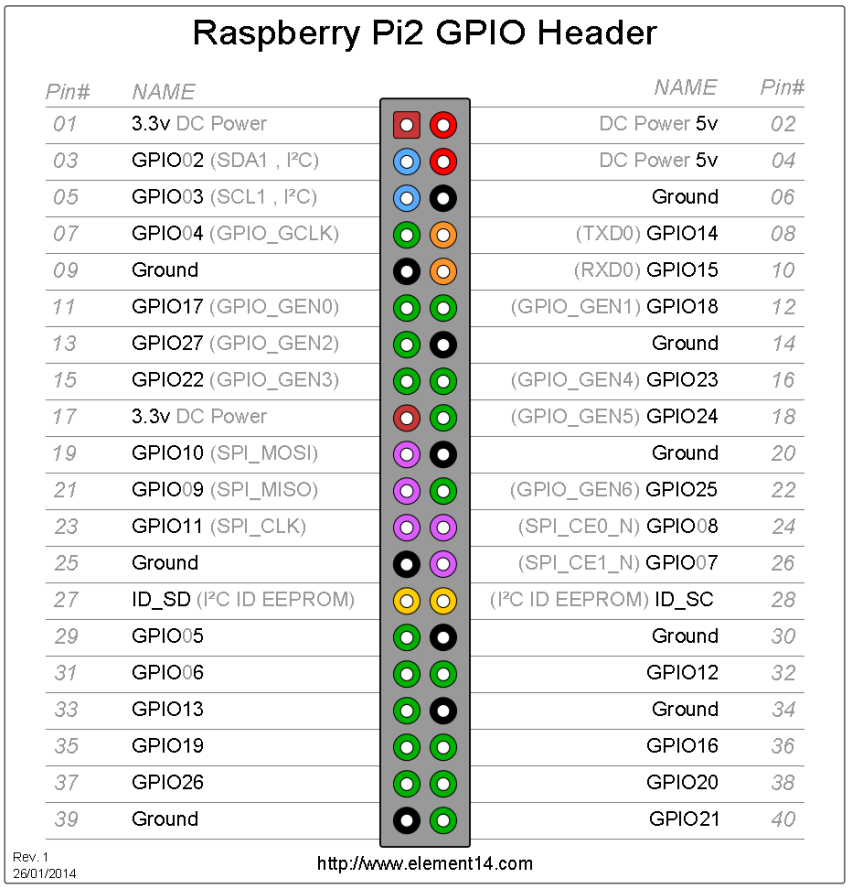
```
sudo nano das_blinken_light.py
```

Note that nano is a small editor, hence the cute name. You are welcome to use any editor you like. You should see something that looks like Code 1. The only thing you have to edit is the gpio pin number to match the one you used, see Figure 1.

Listing 1: Blink the Light.

```
# das_blinken_light.py

import RPi.GPIO as GPIO
import time

ledPin = 23

GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
GPIO.setup(ledPin, GPIO.OUT)
GPIO.output(ledPin, GPIO.LOW)
```

Figure 1: Raspberry Pi 2 General Purpose Input Output (GPIO) pinout.



```
try:
  while True:
    GPIO.output(ledPin,GPIO.HIGH)
    time.sleep(1)
    GPIO.output(ledPin,GPIO.LOW)
    time.sleep(1)
except KeyboardInterrupt:
  GPIO.cleanup()
```

To run the code you need to type the following in the terminal window.

```
sudo python das_blinken_light.py
```

You need sudo to give enough permission to access the gpio pins. The LED should blink on for a sec and off for a sec.

0.4 Das Blinken LED II

- all the previous parts
- switch

Hook up with switch or button to ground. The Broadcom chip that runs the gpio for the Pi has the ability to connect a pullup or pulldown resistor to an input. We will thus use a pullup resistor, and the button will pull it down to ground when pressed.

We will now write code to read input and blink led if the button isn't pressed. Type in:

```
sudo nano das_blinken_light_II.py
```

You can also refer to the GitHub repository, if need be.

Listing 2: Blink the Light while the button is not pressed.

```
# das_blinken_light_II.py

import RPi.GPIO as GPIO
import time

ledPin = 4
butPin = 22

GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
GPIO.setup(ledPin, GPIO.OUT)
GPIO.setup(butPin,GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.output(ledPin, GPIO.LOW)

try:
  while True:
    GPIO.output(ledPin,GPIO.input(butPin))
    time.sleep(.1)
    GPIO.output(ledPin,GPIO.LOW)
```

```
    time.sleep(.1)
except KeyboardInterrupt:
  GPIO.cleanup()
```

```
sudo python das_blinken_light_II.py
```

0.5 Das Blinken LED III

One last thing for today is to dim the led. The Raspberry Pi does not have an A/D converter, so we will just deal with two levels based on the button. We also only have one pulse width modulated output on Broadcom (BCM) pin 18, which is Board pin 12. Lack of A/D converters is one of several reasons it is not a replacement for a micro-controller, the main one being it doesn't have a real time operating system (RTOS) and lacks the necessary timers, like watchdog timers, to build one. If you ever need a micro-controller then use an Ardino, MSP430, or similar. The Pi has a bunch of advantages too in standard tools, quad core, and so on. We care more about the later for labs and testing. If you ever build patient used tools, get a micro-controller as you need the RTOS.

Getting back to the point, we want to have the LED on all the time now, and we will dim it when the button is pushed. To handle the diming, we will use a pulse width modulator. We will set the frequency to 50Hz, which is a reasonable refresh rate, and will use the duty cycle (percent of the wave that is high) to control the brightness. This is a simple and standard way to handle this. Type in:

```
sudo nano das_blinken_light_III.py
```

You can also refer to the GitHub repository, if need be.

Listing 3: Dim the Light while the button is pressed.

```
# das_blinken_light_III.py

import RPi.GPIO as GPIO
import time

ledPin = 18
butPin = 22

GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
GPIO.setup(ledPin, GPIO.OUT)
GPIO.setup(butPin,GPIO.IN, pull_up_down=GPIO.PUD_UP)
pwm = GPIO.PWM(ledPin, 50)
pwm.start(100)

try:
  while True:
    if GPIO.input(butPin):
      pwm.ChangeDutyCycle(100)
    else:
      pwm.ChangeDutyCycle(25)
except KeyboardInterrupt:
  pwm.stop()
  GPIO.cleanup()
```

```
sudo python das_blinken_light_III.py
```

Lab 1

Electromyography (1A)

1.1 Biopotentials

Biopotentials are formed by ion concentration differences inside and outside a cell. Membranes and specialized pumps in the membrane regulate and adjust the concentrations in response to external and internal stimulation, permitting the generation and propagation of biosignals. Measuring the electrical potential in muscles is called electromyography or EMG.

Generally, doctors use needle electrodes, so the skin only needs to be wiped with alcohol to prevent infection, but we will not be using needle electrodes for safety and legal reasons (Texas Court of Appeals, Third District, at Austin, Cause No. 03-10-673-CV. April 5, 2012)++.

1.2 Setup

First, we will get the Arduino IDE. You we need to make sure we have the latest version of our software. Open a terminal window and enter the following commands:

```
sudo apt-get update

sudo apt-get upgrade

sudo apt-get install arduino
```

The ArduBerry is a shield that allows Raspbery Pi's to use Arduino shields. Type the following commands:

```
cd code

git clone https://github.com/DexterInd/ArduBerry.git
```

Open a web browser and go to `http://www.dexterindustries.com/Ardubecrry/getting-started/` and follow the instructions. You will download and install the drivers and run a simple test script that verifies everything is working. The brief summary is below (the website has pretty pictures to accompany this):

1. Stack ArduBerry on Raspberry Pi.
2. Go to the scripts directory in the ArduBerry repo : `cd ArduBerry/script`

3. Make the install script executable: `sudo chmod +x install.sh` then run it as root: `sudo ./install.sh`

   and follow prompts, pressing `enter` and `y` as needed.

4. The system should automatically reboot.
5. Open the Arduino IDE from the system menu.
6. From the **Tools** menu, select the **Programmer** sub-menu, then select **RaspberryPi GPIO**.
7. Load **blink** or another sample sketch, and press **CTRL+Shift+U** to upload (or select **Upload using Programmer** from the File menu)
8. Verify that the LED blinks (if running blink)

1.2.1 Olimex EKG/EMG Shield

You should not have to perform any installs for this. The shield is static sensitive so be careful, and the pins are often bent so be more careful. Disconnect power. Insert onto the Arduberry. Connect power. From Pi launch Ardino editor, and set the target device to an uno and the programmer to GPIO. **Load arduberry emg sketch**, then **program**.

Now open a command window and cd to your EMG directory. Run EMG.py. This is a simple test program, that should give you 8 numbers (1-4 interspersed with something around 300). If you get this all is well.

Now connect a volunteer to the ekg leads. L goes on the left arm, R goes on the right arm, and D goes on the right leg. They need to be symmetrically placed, i.e. all at wrist/ankle or elbow/knee, etc.

Run either the fixed (takes 2k samples then plots and holds) or plot (for dynamic plots).

1.3 General Advice

Gel electrodes provide a good measure but a few basic precautions should be followed to get the best signals.

- Remove oils from your skin with soap and water to improve the signal.
- Don't apply lotions or creams.
- Make sure you are hydrated (dry skin doesn't conduct as well).
- Try to use smooth skin with minimal hair (callouses and hair make it more difficult to get a good signal).
- Body fat reduces the signal, so try to find areas where the muscle is as close to the surface as possible.

1.4 Test

You will first need to setup the arduino on the shield to respond to our code. From the application menu, pick the first menu group and the arduino ide should be the first pick. In it, open the arduino sketch in our lab_02 directory. From the file menu select upload via programmer. You are set.

Now open a terminal window and type

```
cd c*/B*/l*/lab_02
```

```
sudo python EMG_plot.py
```

After a second or two the plot window will open and start displaying the plot of the difference of R and L with D used as ground. You can stop the plot with ctrl-c, though this will exit. If you want to take a fixed data length and have the plot stay up then use

```
sudo python EMG_fixed.py
```

I suggest using plot to try most of the experiments below.

Start with the metal plate electrodes. Place ground on bicep. Place the other two on the muscle the forearm that moves the wrist - one in the middle and one near the end. What happens if you use the right leg? Try metal plates and gel electrodes (just do arm)?

Place other two on the same forearm, one in the middle of a muscle and one near the end of the same muscle. Locat muscles for two different fingers on the same hand by extending and contracting individually and noticing muscle flexure (i.e. I want you to do this twice). What happens if you use the opposite arm ?

Record signals for motion and identify finger flexed by chart only. Explain.

How does the amount of exerted force relate to the signal?

Lab 2

Electro-Kardio-Gram (1B)

2.1 Test

Use of the EKG is essentially the same as for EMG. I have copied the basics here for convenience.

You will first need to setup the arduino on the shield to respond to our code. From the application menu, pick the first menu group and the arduino ide should be the first pick. In it, open the arduino sketch in our lab_02 directory. From the file menu select **upload via programmer**. You are set.

Now open a terminal window and type

```
cd c*/B*/l*/lab_02
```

```
sudo python EMG_plot.py
```

After a second or two the plot window will open and start displaying the plot of the difference of R and L with D used as ground. You can stop the plot with ctrl-c, though this will exit. If you want to take a fixed data length and have the plot stay up then use

```
sudo python EMG_fixed.py
```

I suggest using plot to try most of the experiments below.

You may use whatever electrodes you want. We want to measure the three main leads:

1. Left Arm - Right Arm
2. Left Leg - Right Arm
3. Left Leg - Left Arm

The key information to look for is the direction of the heartbeat. You can do this one of two ways:

1. Look for the two leads that are most in phase with the ideal signal drawn in class, and the actual heatbeat is between them.
2. Look for the lead that is most out of phase with the ideal signal and the actual heatbeat is orthogonal to this.

Find the quadrant of the heartbeat.

Lab 3

Op Amps

3.1 MCP 3008

The MCP 3008 is an 8 channel, 10 bit A/D converter that is bus addressable. Communication is through a SPI bus. Our Raspberry Pi has hardware support[1]. There are four code sequences I have provided:

1. mcp3008bitbang.py - slow software SPI bus. Don't use, this is only for reference if you don't have a system with hardware support.
2. mcp3008hw.py - fast hardware SPI bus. outputs values on command line to 3 decimal places. Good for getting precise values, but bad for lots of values.
3. mcp3008plot.py - hardware SPI bus that plots the results on a graph. Tends to be slow because it has to sample and plot.
4. mcp3008plot-thread.py - multi-threaded hardware SPI bus that plots the result on a graph. Yes I was having fun... This has three threads, one that reads, one that outputs a square wave, and one that plots. Syntax for the plot commands follows MatLab standards. Fast and oh so much fun.

These, of course, require library support so we have written an install shell script. You will need to navigate to where it is located, change its permissions so it is executable, then run it. From a terminal window type:

```
sudo raspi-config
```

Select interface setup, then enable SPI, I2C, and 1-wire by selecting them and then yes. Note you have to do this three click process for each. Once done you will be prompted to reboot, which needs to be done at this point since you changed OS options. Once the system has rebooted open a terminal window and type:

```
cd  /code/labs/labs

chmod 755 install_python_libs.sh

./install_python_libs.sh
```

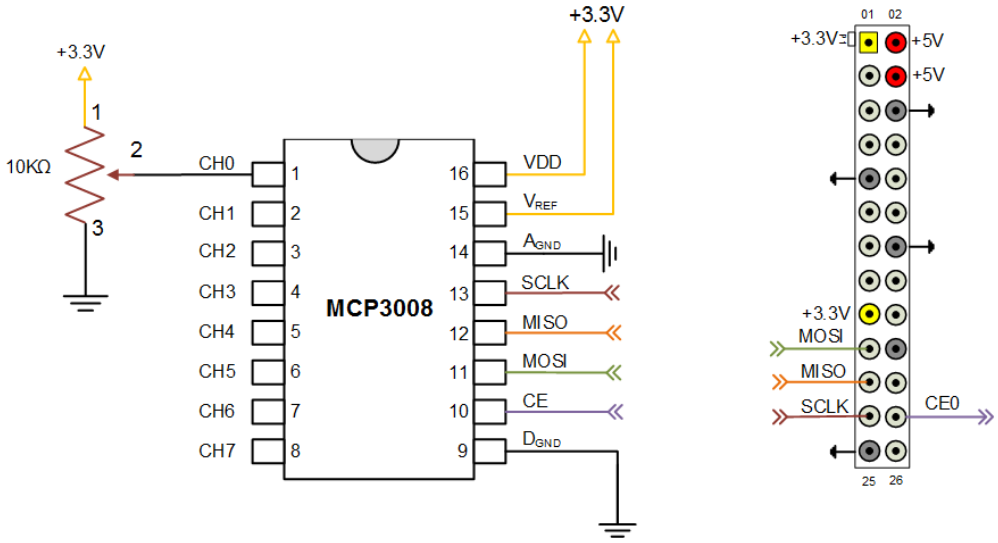
The mcp3008 has an analog power reference(Vref) and ground and a digital power (Vdd) and ground. Vdd must always be connected to 3v3 (3.3V) and its ground (pin 9) to ground. Vref and analog ground define the voltage range to compare. Often this is the same, but it doesn't have to be. For instance, connecting to

[1] If you had a controller that didn't have hardware support you would have to implement it in software, which is very slow and called bit banging. I have included a bit banging code for your reference in the code part of lab 3.

5v gives a bigger swing. Remember that the chip only has 10 bits of precision (just over 1000 divisions) and thus the precision is:

$$\frac{V_{ref} - A_{gnd}}{2^{10}}$$

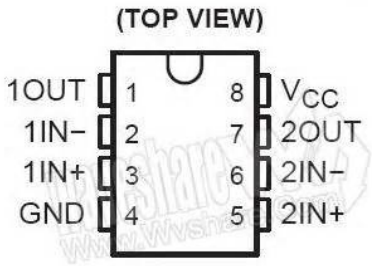
Thus the larger the reference swing, the less precise the measurement. If you want to measure something small, you should have a small reference! The basic hookup is thus



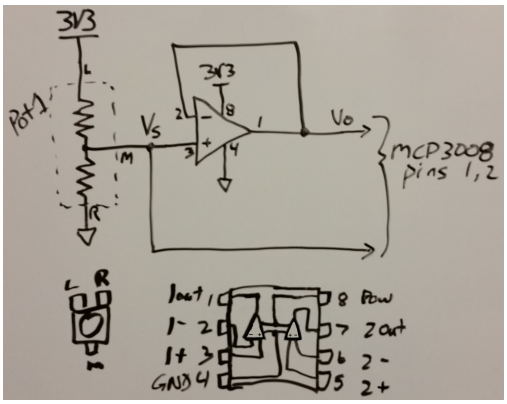
Note the voltage divider on the left is only for testing and does not need to be done each time. You will build the divider the first time to test your code and setup. The 8 pins on the left are now available for measuring your circuit.

3.2 Follower

We will be using a general purpose LM358 Op Amp. We will connect the positive rail to 3v3 and the negative rail to ground. Hook the output to the negative input and the signal to follow goes on the positive input. The pinout of the 358 is:

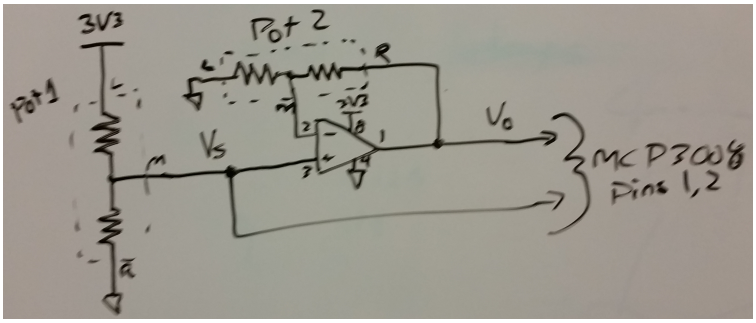


To make a signal you can easily vary, use a potentiometer (pot) to make a voltage divider. Pick a high value, which will make the current low and the signal easy to harm, thus justifying the follower circuit. The Now hook the output of the opamp to the inverting input and the middle sweep of the potentiometer to the non-inverting input. The resulting circuit is:



3.3 Non-Inverting Amplifier

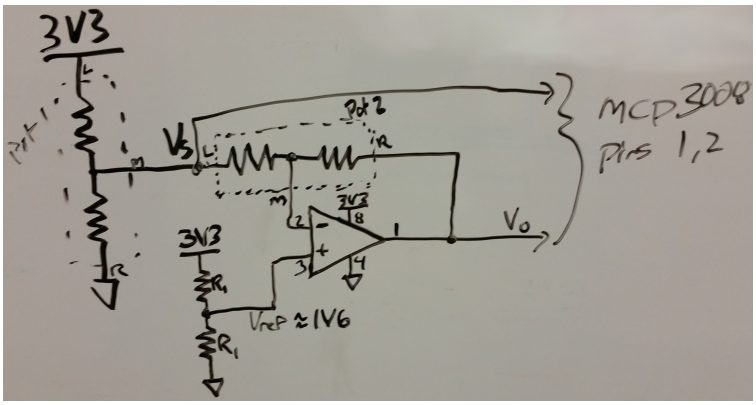
The signal stays on the non-inverting input, but now we will feedback the output across a voltage divider to ground. The middle sweep of the voltage divider will be hooked to the inverting input of the opamp. This second op-amp controls the gain of the system.



3.4 Inverting Amplifier

The signal now replaces the ground on the feedback circuit (gain potentiometer), so that now the output of the opamp and the input signal are on oposite sides of the second potentiometer, and the middle sweep of the second potentiometer still goes to the inverting input. We would normally put the ground on the positive input of the opamp, but we can't handle negative voltages, so we have to make a fake ground half way between 0V and 3V3. We can make a new reference ground by a voltage divider between 3v3 and ground across two 7.5k ohm resistors[2], so the center will be around 1v6. Hook this to the positive input, and the rails should be connected to 3v3 and ground still. The Vref on the MCP3008 is also still connected to 3v3.

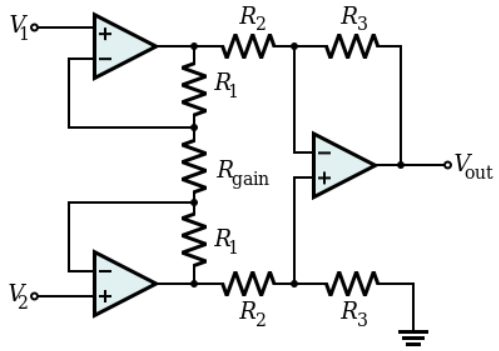
[2]hook the resistorsin series and put 3v3 to one outer pin and ground to the other, the center is the reference


3V3
Pot 1
Vs
Pot 2
MCP3008
pins 1,2
3V3
8
Vo
3V3
R1
R1
VREF ≈ 1V6

Lab 4

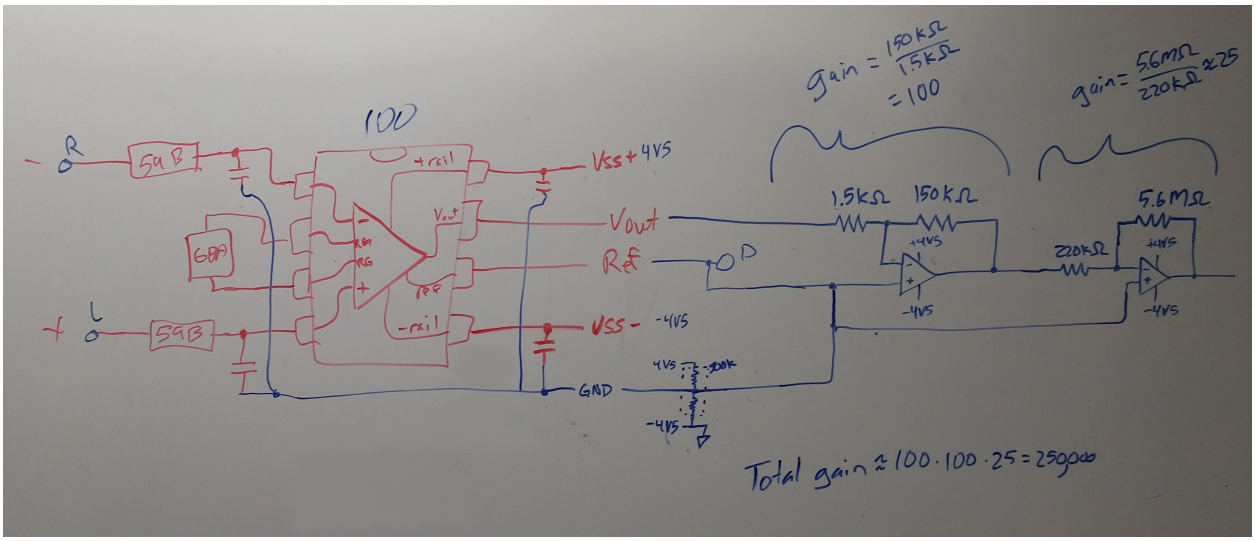
Build Your own EMG (2A)

Now that we know how to build an opamp circuit to amplify a signal, we can build our own Electromyography circuit. You probably noticed how sensitive the opamp circuits were to the resistor values, so we will use a specialized device called an instrumentation amplifier, which is designed to amplify weak signals by combining non-inverting amplifiers with precision matched resistances referenced to each other through a gain resistor (which controls the gain of both) that feeds into a differential amplifier setup (like an inverting amplifier of one signal with respect to a voltage divided version of the second).



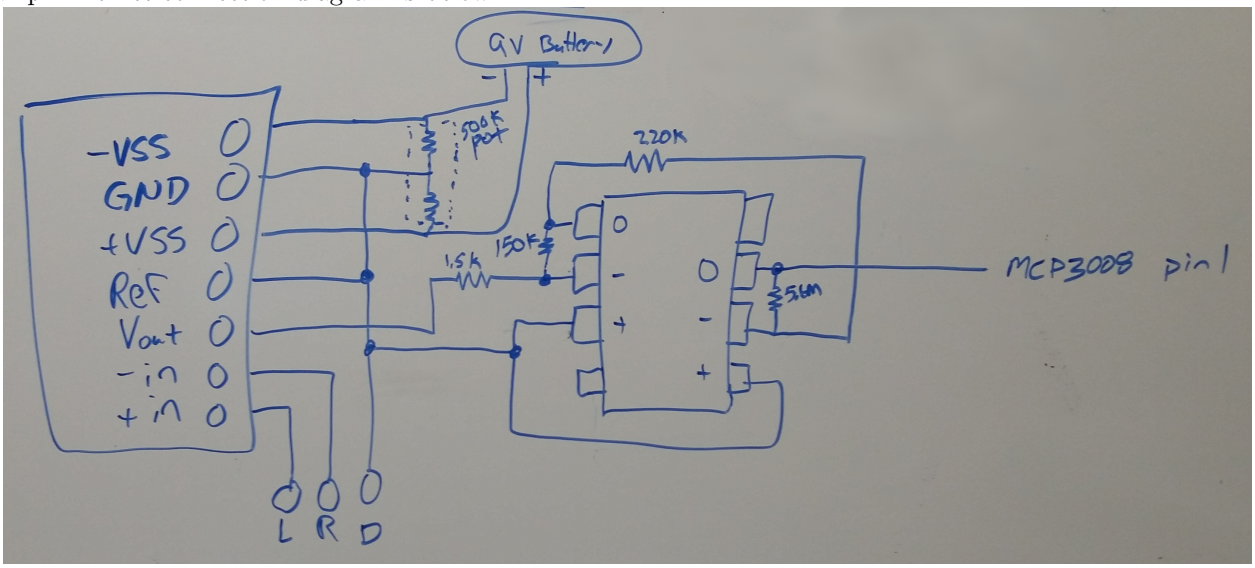
4.1 Wiring the Circuit

We are using an Analog Devices 8221 (AD8221) instrumentation amplifier, that is on a breakout board. The breakout board contains resistors and capacitors needed to make the circuit run well. For instance the positive and negative rails have a capacitor to ground that is used to clean up the power. Additionally, the positive and negative inputs have 4kΩ resistors inline and a capacitor to ground - which acts like a low pass filter. Finally, the gain resistor is set at 499Ω to make the gain about 100. The breakout board greatly simplifies our lives by providing a nicely set up system. The input impedance is a bit low for us, but it will work. If you add more resistance on the inputs, it will reduce the corner frequency of the low pass filter and cause other issues, so we will just leave it. The signal will need extra amplification so we will run the output into two inverting amplifier stages (each side of our current LM358 opamp) and then to the 3008 and then to our PI. The first stage will get gain of 100, which makes the total gain 10000 times, but since the input is loaded and weak, we will need a second stage with a gain of 25, then we should get a nice signal. The overall circuit we will build is below. Note the red lines and components are the instrumentation amplifier breakout board, so you don't need to add each component, just the board.



To get the gain of 100, we need to set the resistor between the output of the first stage (pin 1 of our opamp) and the negative input (pin 2 of the opamp) to be 100 times the resistance of the one between the negative input and signal from the instrumentation amplifiers output. We will set the impedance to be a moderate range using 150kΩ and 1.5kΩ respectively.

The second stage needs around a gain of 25, and we will pick the impedance to be around a mega ohm, so that we can easily set up a filter with a corner frequency around 100 Hz if we need to. Thus we will pick the resistor between the output (pin 6 of the opamp) and the negative input (pin 5 of the opamp) to be 5.6MΩ and the resistor from the negative input and the output of the first stage (pin 1 of the opamp) to be 220kΩ. Note: I tested a number of the instrumentation amps and some needed 33okΩ resistor instead of 220kΩ, because the gain was too high for the dc component. If you get clipping you can reduce the gain by swapping the 220kΩ resistor for a 330kΩ resistor. The output of the entire circuit is from pin 6 of the opamp. The net connection diagram is below.



The positive and negative rail will be powered by a 9v battery. This will do two things:

1. you will be isolated from wall power, greatly increasing your safety.
2. the circuit will have a bigger voltage swing, which will make the amplifier's job much easier.

We will connect the negative battery lead to the Raspberry Pi's ground[1] We will make a false ground for the instrumentation amp using the 500k potentiometer and the 9v battery, setting the false ground around 1.4 volts.

4.2 Graphing

This circuit is much more power hungry than the simple circuits we used, and so you will notice a difference in the plotting efficency, since the Raspberry Pi will run slower. To counter this we need to go to a more efficient piece of code. Our current code uses one thread that takes in data and plots. It is more efficient to make two threads - one that takes in data and puts it in the array, and one that just plots data. This is implemented in the MCP3008thread[1-3].py code. The first two are more generic, while the third is special of our case - it plots one line and has our values preset, so use it.

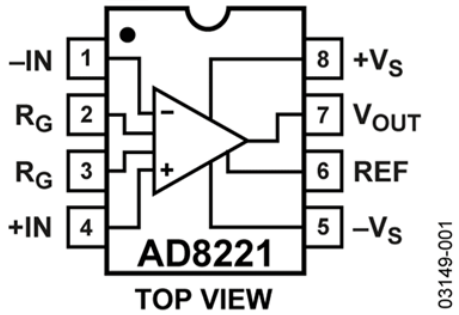
```
cd c*/B*/p*/*3

sudo python MCP3008thread3.py
```

Plot the data for the same muscle group you did on the EMG shield and compare how easy it is to make out the muscle contractions. Compare the quality of the two circuits.

4.3 Optional Reference: Understanding the AD8221 Breakout

Bottom Top

The AD8221 instrumentation amplifier pinout is



[1]In a commercial circuit we would put this through an optocoupler or similar to provide further safety. The setup we are using should only be used for EMG on the same arm with people who don't have a pacemaker. In extreme circumstances such as a lightning strike to ground current could come back from the ground. The amount of current to cause problems is lower if the heart is between any of the probes (hence only use for one arm) and even more with arrythmia that requires a pacemaker or similar.

Surface mount resistors have three potential codes: 3 numbers, 4 numbers, and EIA-96. The board uses E96 resistors (1% tolerance) and thus uses the EIA-96 Codes. The first two numbers are a code you look up (see first chart) to get the 3 digit value of the resistor, and the Letter at the end is the multiplier (see second chart).

Code	Value	Code	Value	Code	Value	Code	Value	Code	Value	Code	Value
01	100	17	147	33	215	49	316	65	464	81	681
02	102	18	150	34	221	50	324	66	475	82	698
03	105	19	154	35	226	51	332	67	487	83	715
04	107	20	158	36	232	52	340	68	499	84	732
05	110	21	162	37	237	53	348	69	511	85	750
06	113	22	165	38	243	54	357	70	523	86	768
07	115	23	169	39	249	55	365	71	536	87	787
08	118	24	174	40	255	56	374	72	549	88	806
09	121	25	178	41	261	57	383	73	562	89	825
10	124	26	182	42	267	58	392	74	576	90	845
11	127	27	187	43	274	59	402	75	590	91	866
12	130	28	191	44	280	60	412	76	604	92	887
13	133	29	196	45	287	61	422	77	619	93	909
14	137	30	200	46	294	62	432	78	634	94	931
15	140	31	205	47	301	63	442	79	649	95	953
16	143	32	210	48	309	64	453	80	665	96	976

Code	Multiply factor
Z	0.001
Y/R	0.01
X/S	0.1
A	1
B/H	10
C	100
D	1,000
E	10,000
F	10,0000

Lab 5

Pulse Oximeter (2B)

5.1 Background

Hemoglobin is a protein in red blood cells that reacts to oxygen forming oxyhemoglobin (HbO_2). When not oxygenated hemoglobin is referred to as deoxyhemoglobin (Hb). Pulse oximetry is a non-invasive way to measure the amount of oxygen dissolved in the blood, which is called the oxygen saturation (SpO_2). Oxygen saturation is measured by detecting Hb and HbO_2, using their absorption sectra at two different frequencies (typically red around 660nm and infrared around 840nm to 940nm). These values were selected because Hb has higher absorption of red light and HbO_2 has higher absorption of infrared, see figure 5.1.

We are going to use two LEDs, one red one infrared, a photodiode, and a simple filter/amplifier to measure the absorbed light at two frequencies. The data that produced the graph came from the tabulated molar extinction coefficient, e, for hemoglobin in water compiled by Scott Prahl using data from

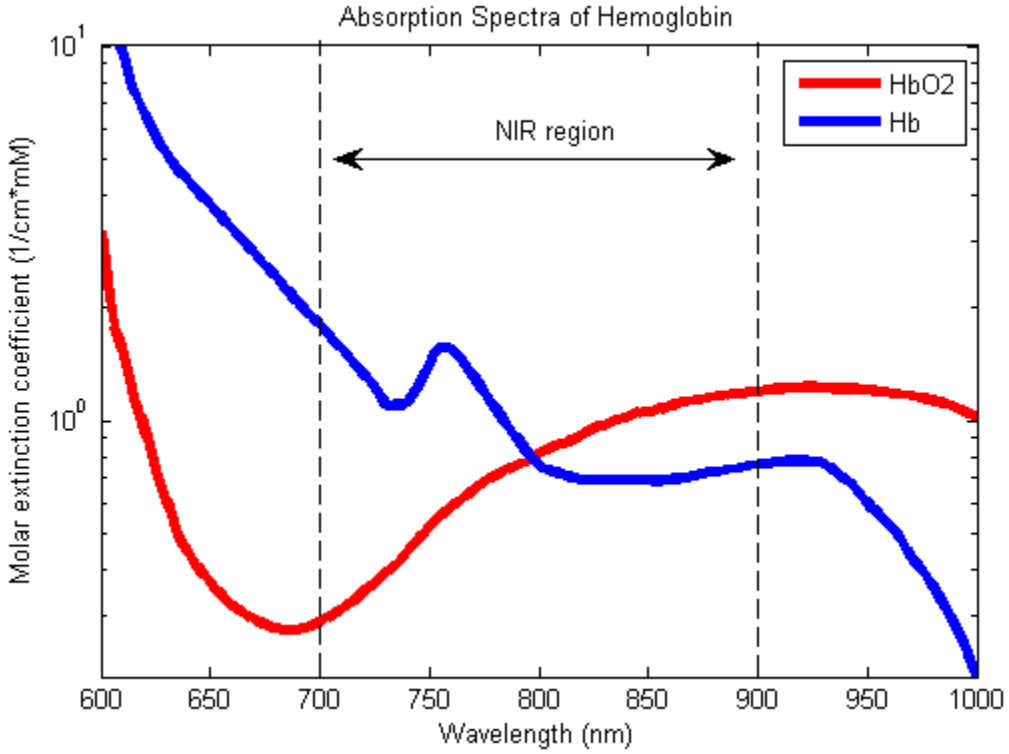
- W. B. Gratzer, Med. Res. Council Labs, Holly Hill, London
- N. Kollias, Wellman Laboratories, Harvard Medical School, Boston

$\lambda[nm]$	$HbO_2[cm^{-1}/M]$	$Hb[cm^{-1}/M]$
660	319.6	3226.56
830	974	693.04
840	1022	692.36
930	1222	763.84
940	1214	693.44

To get absorption, you multiply the molar extinction coefficient times the molar concentration times the pathlength and divided by the molecular weight of hemoglobin. Since we are comparing two absorption measurements at different frequencies, the molecular weight of hemoglobin cancels and can be ignored. Similarly, if we put both light sources (red and infrared) equally distant through your body to the photodiode, then the pathlength will also cancel and can be ignored.

Light is absorbed by tissue, venous blood, non-pulsatile arterial blood, and pulsatile arterial blood. The first three are constant and will be measured as the DC component of the measurements. The final one, pulsatile arterial blood, will be the AC component, and will also allow us to get heart rate.

Figure 5.1: Absorption spectra of oxyhemoglobin and deoxyhemoglobin. *Image by Adrian Curtin used under Creative Commons Attribution-Share Alike 3.0 Unported License.*



5.2 First Method

By taking the ratio of oxygenated arterial blood in the pulsatile (AC) over the other (DC) portion of the signal at two frequencies we can calculate the absorption ratio (AR)[1]

$$AR = \frac{\frac{AC_{red}}{DC_{red}}}{\frac{AC_{infrared}}{DC_{infrared}}} \tag{5.1}$$

We can then compare this number to previously measured values that were tested in a different manner. This is only approximate because of differences in blood volume, perfusion, etc. as well as movement and misplacement effect measurements. Even so we can make a simple table of values and interpolate to get other values:

AR	SpO_2
0.4	100%
1.0	85%
3.4	0%

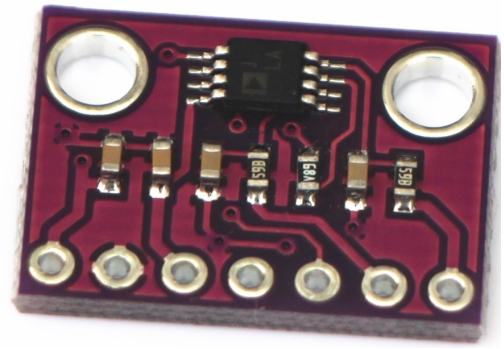
[1] SpO_2 can also be calculated by ratio of the logarithms of the AC components at the two frequencies. Multiply by 100 to get percent.

Lab 6

One Wire Thermometer

6.1 Assemble Circuit

The basic one wire temperature sensor from Dallas Semiconductor, uses a TO-92 package, just like a transistor. Originally they only operated in parasitic mode, but the latest versions have had a powered version that can be run in parasitic mode. The pinout is as follows:



where pin 1 is GND, pin 2 is the data line, and pin 3 is NC on old systems and 3.3v or GND on new ones (Ground for parasitic). The data line needs a pull-up resistor[1] of around 3kΩ to 6kΩ, with the sugested value at 4.7kΩ. The one wire thermometer is very easy to connect to a Raspberry pi. Connect pin 1 to ground, pin 2 to GPIO 4 (4th pin on the left of the Raspberry Pi's GPIO), and pin 3 to 3.3 volts. Now connect a resistor around 4.7kΩ between pins 2 and 3 of the one wire. You are ready to set up your Pi!

6.2 Add One Wire Support

First you need to edit the boot configuration file to add one wire support.

```
sudo nano /boot/config.txt
```

Scroll to the bottom (use the down arrow), then type **dtoverlay=w1-gpio**. Save by typing **ctrl-o** then exit with **ctrl-x**.

Now reboot to make your changes active.

```
sudo reboot
```

[1] A resistor that supplies power to a bus, connected between the data and 3.3v in our case.

6.3 Test It

```
sudo modprobe w1-gpio
```

```
sudo modprobe w1-therm
```

```
cd /sys/bus/w1/devices
```

```
ls
```

Note that the next directory has a really long name, and incorporates the device number so it is not constant on all systems. This allows multiple to be connected.

```
cd 28*
```

```
cat w1_slave
```

6.4 Code It

Listing 6.1: Read from a One Wire Thermometer.

```
import os
import glob
import time

os.system('modprobe w1-gpio')
os.system('modprobe w1-therm')

device_folder=glob.glob('/sys/bus/w1/devices/28*')[0]
device_file=device_folder+'/w1_slave'

def read_w1_file():
  f=open(device_file,'r')
  lines=f.readlines()
  f.close()
  return lines

def read_temp():
  lines=read_w1_file()
  while lines[0].strip()[-3:] != 'YES':
    time.sleep(0.2)
    lines=read_w1_file()
  temp_loc=lines[1].find('t=')
  if temp_loc != -1:
    temp_string=lines[1][temp_loc+2:]
    temp_c=float(temp_string)/1000.0
    temp_f=temp_c*9.0/5.0+32.0
    return temp_c, temp_f
```

```
while True:
  print(read_temp())
  time.sleep(1)
```

Lab 7

Electro-Encephlogram

7.1 Installing in MatLab

We will be on the desktop machines today. You need to setup EEGLAB in your directory inside MatLab.

1. Unzip the EEGLAB zip file in the folder of your choice
2. Start Matlab
3. Change the Matlab path to the EEGLAB folder you have just uncompressed
4. Type “eeglab” and press enter on the Matlab prompt