Lab 3 – ECG System (2-week lab)

GOALS

1) Build and test the signal conditioning electronics for an ECG system.
2) Develop the Arduino and MATLAB data acquisition system.
3) Record an ECG waveform from yourself.

GENERAL GUIDELINES

You should know the guidelines by now …

REQUIRED MATERIALS

- Lab kit and BLUE WIRE (for -9V connections).
- Oscilloscope, probe kit, function generator, benchtop power supply
- Integrated Circuits (ICs): AD620 inst amp (one); TL081 op amp (one); CNY17 optocoupler (one)
- Resistors: 150 ohm (brown/green/brown) 5% resistor (one)
  220 ohm (red/red/brown) 5% resistor (one)
  1.0 kohm (brown/black/red) 5% resistor (two)
  2.61 kohm (red/blue/brown/brown) 1% resistor (one)
  10.0 kohm (brown/black/black/red) 1% resistor (two)
  100 kohm (brown/black/yellow) 5% resistor (two)
  1 Mohm (brown/black/green) 5% resistor (one)
- Capacitors: 0.1 μF capacitor (two)
- Diodes: 1N4148 rectifiers (four); Red LEDs (three)
- For the final lab demo, Buma will provide you with two 9V batteries, battery leads (two), shielded cables (three), and electrodes (three)

LAB ACTIVITIES

1) Construct and test input protection circuitry, instrumentation amplifier, and high pass filter
2) Construct and test amplifier + low pass filter
3) Construct and test the optocoupler
4) Develop Arduino data acquisition program
5) Develop MATLAB data display program
6) Make a live ECG measurement using your AWESOME circuit!

INTRODUCTION

The entire ECG amplifier has three main parts: (1) Instrumentation amplifier with high-pass filter (2) High gain amplifier with low-pass filter (3) Optocoupler. You will use the Arduino and MATLAB to record and display data. This is the most time consuming lab of the course. However, seeing your own ECG signal with a circuit you built is a rewarding experience!

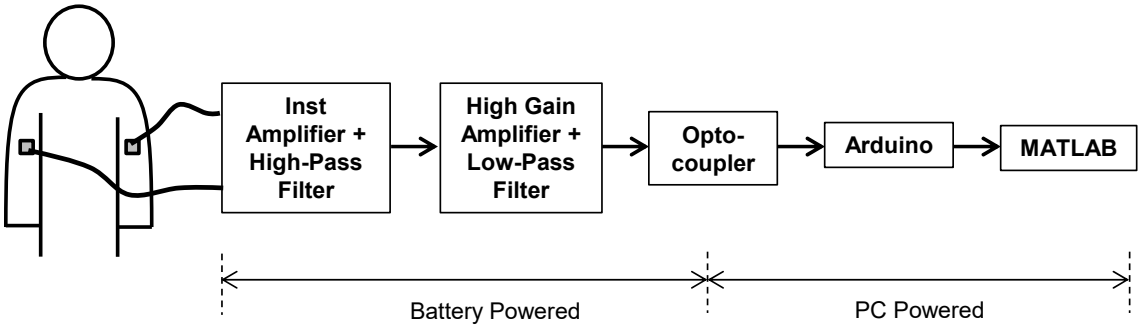


Fig. 1: Simplified diagram of ECG system.

PART 1: INST AMPLIFIER AND HIGH-PASS FILTER

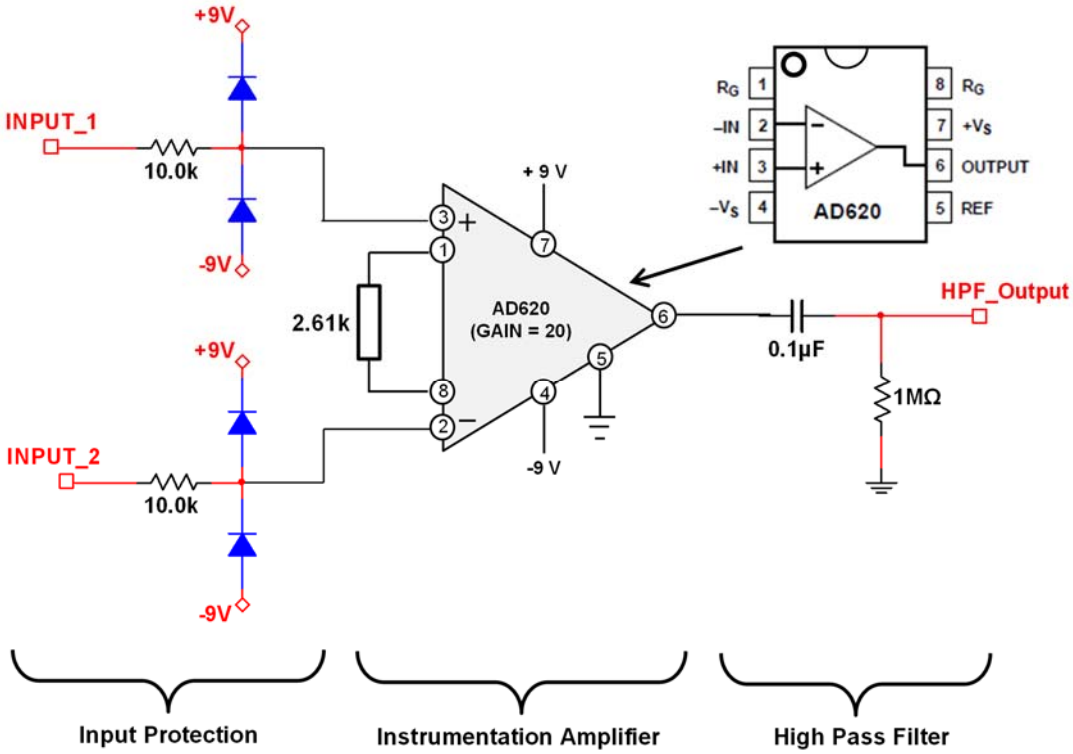


Fig. 2: First part of ECG amplifier circuit. Color-code your wiring and be NEAT! During initial testing, +/-9V and GND are provided by the benchtop supply. Once everything works, the benchtop supply is replaced with two 9V batteries.

- **Step 1a**: Build the preamplifier and high-pass filter circuit. See comments below:

1) Power connections.
   - Reserve the LEFT THIRD of your breadboard for +/-9V power.
   - Fig. 2b shows the designated columns of holes for +9V, GND, and -9V.
   - The "left pair" and "right pair" of columns will eventually come from 9V batteries (end of lab).
   - You can see Buma's board as a reference.

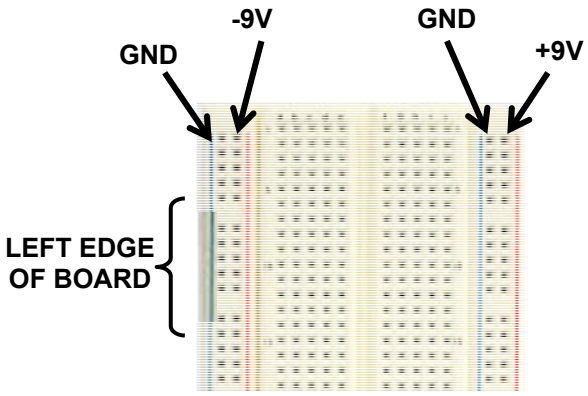


Fig. 2b: Power connections on breadboard. These will eventually be connected to 9V batteries.

2) Protection circuitry.
   - The diodes are the orange components with a black stripe.
   - The black stripe is the "dash" side of the diode (see Fig. 2c).
   - For now, you can leave "Input 1" and "Input 2" unconnected (see Fig. 2). Don't worry -- we'll connect them soon!
   - Remember to properly color-code your wiring and be neat!
     - RED wire +9V connections
     - BLUE wire for -9V connections
     - BLACK wire for GND connections
     - YELLOW wire for all other connections

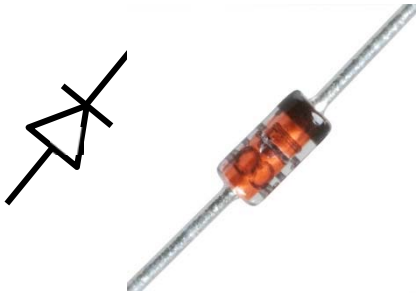


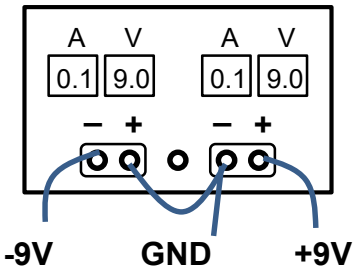
Fig. 2c: Diode symbol and picture.

3) Instrumentation amplifier.
   - The 2.61 kohm resistor sets the gain to 20.

4) High pass filter.
   - Theoretically, the RC high pass filter has a cut off frequency of f_C = 1.6 Hz.
   - However, this cut-off is not abrupt. Frequencies below about 0.16 Hz are strongly blocked by this filter.

5) Check your circuit with Buma. He needs to confirm that your diodes are correct and wiring is NEAT.
   - The protection diode connections are usually the trickiest part.
   - Buma will brutally punish breadboards that have improperly color-coded and/or messy wiring.

- **Step 1b**: Configure the circuit inputs. Your circuit will initially be tested with a "cardiac" waveform produced by the Agilent function generator.
    - Build a 100:1 voltage divider (see Fig. 3) fairly close to the AD620 chip (e.g. within 2 inches or so).
        - The purpose of the divider is to reduce the amplitude of the cardiac test signal to 1 mVpp
        - The resistor color codes are on Page 1 of this handout.
        - Use the coax cable with alligator clips to connect the function generator to "INPUT 1" of the circuit (see Fig. 3).
            - The RED clip goes to the top of the 100k resistor.
            - The BLACK clip goes to GND.
        - For circuit testing, connect "INPUT 2" to ground (see Fig. 3).
        - If necessary, feel free to take a peek at Buma's board.
    - For now, we'll use the benchtop supply to provide +/- 9V and GND to your circuit board (see figure to the right).
        - Ask Buma or the lab assistant if you are unsure about the power supply connections.
        - Later in the lab (after testing is complete), you will replace the benchtop supply with two 9V batteries.



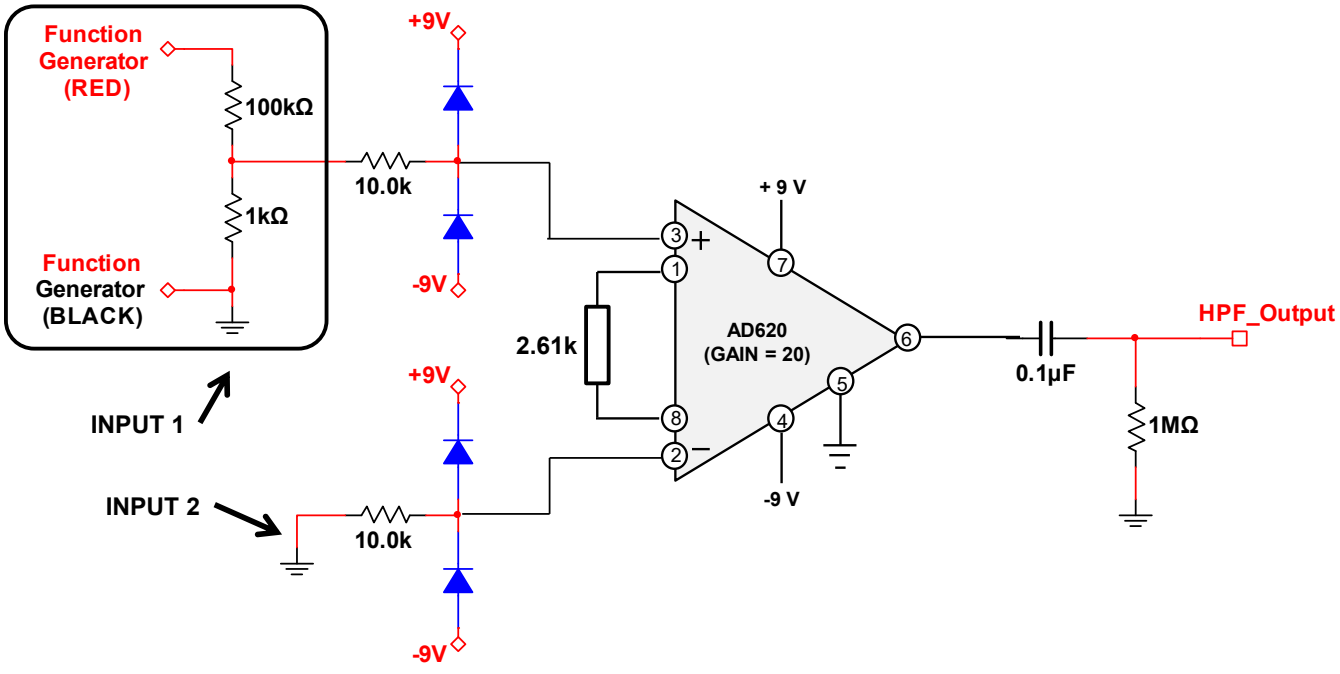


Fig. 3: For initial testing, the function generator is connected to the circuit using a 100:1 voltage divider attached to INPUT 1. Meanwhile, INPUT 2 is connected to ground. For now, the +/- 9V power comes from the benchtop supply.

- **Step 1c**: Configure the function generator. Your circuit will initially be tested with a "cardiac" waveform produced by the Agilent function generator.

  ***If you have the older Agilent function generator (Model 33120A)*:**

  1) Set the Agilent generator to "High Z" output.
     - Press "Shift" and "Enter".
     - Press ">" three times until "D:SYS MENU" appears.
     - Press "∨" once to obtain "1: OUT TERM".
     - Press "∨" again to obtain "50 OHM"
     - Press ">" to make "High Z" appear and press "Enter" to finish.
  2) Set the waveform to "Cardiac".
     - Press "Shift" and "Arb".
     - Press ">" four times to get "CARDIAC" and press "Enter" to finish.
  3) Set the waveform properties to: **Amplitude = 100 mVpp; Frequency = 1 Hz; DC offset = 100 mV**.

  ***If you have the newer Agilent function generator (Model 33220A)*:**

  1) Set the generator to "High Z" output.
     - Press "Utility", then select "Output Setup", then "High Z", and finally select "Done".
  2) Set the waveform to "Cardiac".
     - Press "Arb", then select "Select Wfm", then "Built In", then "Cardiac", and finally select "Done".
  3) Set the waveform properties to: **Amplitude = 100 mVpp; Frequency = 1 Hz; DC offset = 100 mV**.

- **Step 1d**: Measure your circuit output with the scope.
  - Set up the scope in the following manner:
    - Press the "Force" button if the red "Rmt" appears in the top right corner of the scope display.
    - Reset the scope by pressing "Default" on the scope.
    - Make sure both the scope and probe are set to 1X.
    - Set the scope to 10 mV/div (vertical) and 200 ms/div (horizontal).
  - Place the scope probe on the output of the amplifier chip (Pin 6).
  - You should see a PQRST waveform with: **Amplitude ≈ 20 mV (peak-to-peak) and Offset ≈ 20 mV**.
    - It will probably appear pretty fuzzy – that's OK since the low-pass filter will clean things up!
  - Now measure the output of the high-pass filter. The PQRST waveform should no longer have an offset!
  - Demo your circuit to Buma.

(End of Part 1)

PART 2: AMPLIFIER WITH LOW-PASS FILTER

- **Step 2a**: Build the amplifier with low-pass filter. This uses a TL081 op amp configured as a non-inverting amplifier (see Fig. 4).
    - The TL081 pin diagram is shown in Fig. 4.
    - Pins 2 and 3 are the op amp inputs.
    - Pin 6 is the op amp output.
    - You do not need to connect anything to Pins 1, 5, or 8.
    - **Use color-coded and neat wiring!!!!**

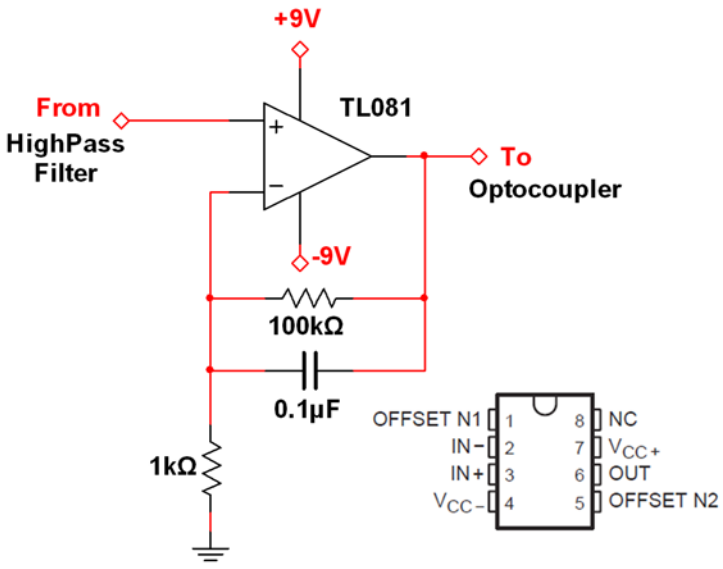


Fig. 4: Amplifier with low-pass filter using a TL081 op amp powered by +/- 9V. The pin diagram is shown on the lower right.

- **Step 2b**: Test the circuit. The input for this circuit comes from the high pass filter.
    - Place the scope probe on the output (Pin 6) of the op amp.
    - Change the scope vertical setting to about 1 V/div.
    - You should see a much larger PQRST waveform with about a **2V peak-to-peak amplitude**.
        - The amplified signal should be MUCH CLEANER than what you observed in Step 1d.
            - ➢ This shows the importance of low-pass filtering!
    - Demo your circuit to Buma.

(End of Part 2)

PART 3: OPTOCOUPLER

As mentioned previously, the optocoupler transmits and receives light with an LED and phototransistor, respectively. Our optocoupler is the CNY17 chip.

- **Step 3a**: Hook up the "left side" of the optocoupler. This is the "battery side" of the optocoupler (although we are currently using the benchtop supply to provide +/- 9V for testing).
    - The left side of the CNY17 chip has a slight bevel (see Fig. 5).
    - The three red LEDs outside of the chip are handy indicators of the ECG waveform – it pulsates with your heart rate!
        - The longer LED pin is the "top" side of the LED.
        - The bias current in the optocoupler LED is roughly 10 mA, which is determined by the red LEDs and 220 ohm resistor.
    - **Use color-coded and neat wiring!!!!**

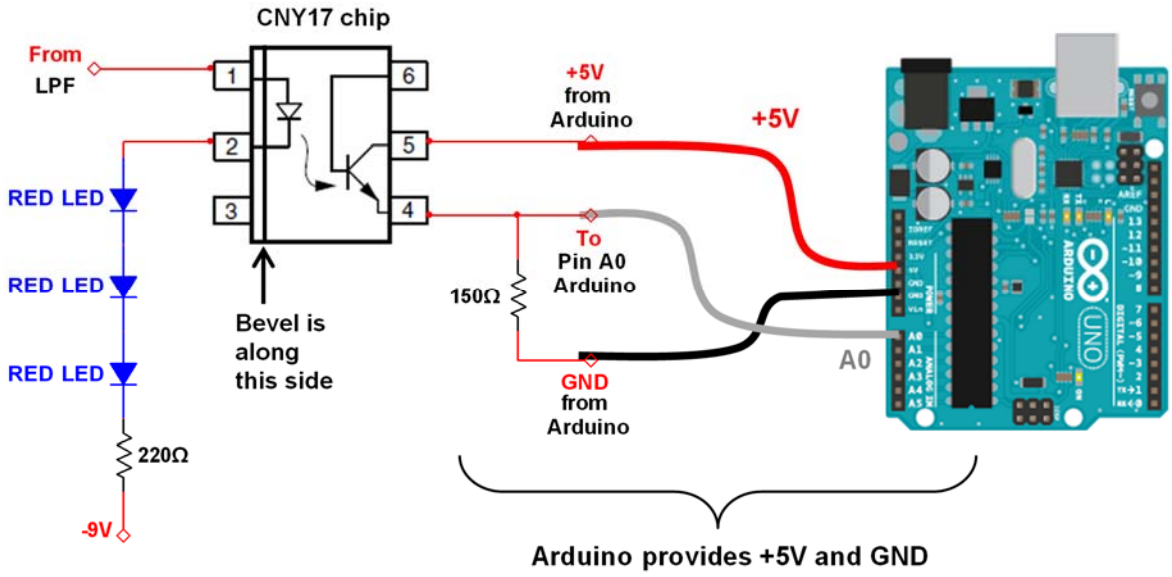


Fig. 5: Optocoupler circuit, where the left side is powered by +/- 9V. The right side is SEPARATELY powered by the Arduino.

- **Step 3b**: Hook up the "right side" of the optocoupler. This is the "PC side" of the optocoupler.
    - Connect Pin 5 of the optocoupler to the +5V pin of the Arduino.
    - Connect Pin 4 of the optocoupler to a 150 ohm resistor that leads to GND from the Arduino.
        - Do NOT connect the Arduino GND to the rest of the breadboard.
        - Remember that we want separate "battery ground" and "PC ground"!
    - **Use color-coded and neat wiring!!!!**

- **Step 3c**: Test the circuit.
    - Make sure the Arduino is plugged into the computer (via USB cable).
    - **The red LEDs should have a periodic (and faint) "flickering" at a 1 Hz rate**.
        - This is because the LED current is modulated by the ECG signal!
        - The flickering will be fairly subtle, so don't expect dramatic flashing from the LEDs.
    - Attach a scope probe to the optocoupler output (Pin 4).
    - The probe ground should be the same as the Arduino.
    - You should see the ECG waveform with ≈ 1 volt amplitude and a ≈ 2V offset.
    - Demo your circuit to Buma.

OK, you're done with your circuit! Now for Arduino and MATLAB programming …

(End of Part 3)

PART 4: ARDUINO

In the previous lab, each data sample required a separate request from the PC to the Arduino. This back-and-forth communication is fine for slow sampling rates, but is problematic for applications requiring high sampling rate.

ECG typically requires a sampling rate of 100 Hz or higher. This is much too fast for our "old" method of data acquisition. **We will instead have the Arduino send data to the PC in "bursts". An outline for the new code is shown below**:

(a) **Setup( )** routine
    - Initialize the serial port
    - Send message to PC

(b) **Loop( )** routine
    - Wait for message from PC
    - If PC requests data,
        - Repeat the following for N = 300 times:
            - Record ADC output
            - Compute Vmeas
            - Send Vmeas to computer
            - Wait 10 ms

- **STEP 4a**: Download and unzip the Arduino template.
  - Go to the course website and download "Lab3_files.zip" to your desktop.
  - Double-click on the zipped folder.
  - Select "Extract all files" (near top of the window) to unzip the contents of the folder.
  - Double-click the "Lab3_Arduino" folder and open the template.
  - You can consult the Lab1+2 handouts or Google for help and/or examples using Arduino functions.
- **STEP 4b**: Write the **setup( )** routine.
  - Initialize the serial port using the **Serial.begin** command
    - Replace the "**???**" with appropriate code.
    - Use the **Serial.begin** command and set the data rate to 57600 bits per second.
    - Don't forget the semicolon at the end of each line!
    - Send message to PC (use the **Serial.println** command to send the character "**a**").
- **STEP 4c**: Declare variables for the **loop( )** routine.
  - The loop routine uses five variables. Declare these before the loop( ) routine. As shown in Fig. 6, Buma wrote the first one for you (he is so kind ….). Replace any other **???** with appropriate code.

Data Type	Name	Value	Purpose
char	**PCstatus**		Stores one byte of text data from the PC
int	**A0_PIN**	**0**	Name of Analog Pin 0
int	**ADCoutput**		Stores the ADC output
float	**Vmeas**		Measured voltage
int	**T**	**10**	Time between readings (ms)
int	**N**	**300**	Number of samples per trace

    - Don't forget semicolons at the end of each line!

```
// loop() is where we do stuff over and over again.

char PCstatus;      // message from PC
???                 // assign name to Analog Pin 0
???                 // stores the ADC output
???                 // measured voltage
???                 // time between readings (ms)
???                 // number of data points per trace

void loop()
```

Fig. 6: Declare some variables before the loop( ) routine.

- **STEP 4d**: Write the **loop( )** routine.
  - Replace all the ??? so that your code looks like Fig. 7. Comments about the code.
    - The **Serial.available** command checks if the serial port has received any data.
      - The **while** loop repeats this process and stops when data has been received.
      - The **Serial.read** command reads one byte of received data from the serial port.
    - The **Serial.read** command reads one byte of received data and stores the value in **PCstatus**.
    - If **PCstatus** is the character "**y**", then we proceed to record/send an entire data trace (N = 300 samples).
      - We use a **for** loop to repeatedly acquire each sample until a full trace is finished.
        - The variable **n** keeps track of the loop iteration.
      - Use the **analogRead** command to read the data from **A0_PIN**.
      - Convert **ADCoutput** to **Vmeas** (see Lab1 handout if you forgot how to do this).
      - **Serial.println(Vmeas, 3)** sends the **Vmeas** value (3 decimal places)
      - Use the **delay** command to wait **T** seconds before the next sample acquisition.

```
void loop()
{
  while (Serial.available() == 0)    // wait until data from PC is available
  {
  }

  PCstatus=Serial.read();     // read one byte of received data from PC

  if (PCstatus == 'y')
  {
   for (int n=0; n < N; n++)      // loop to acquire and send N data values
   {
     ADCoutput = ???;             // read ADC output
     Vmeas=???;                   // compute voltage
     Serial.println(???);         // send Vmeas (3 decimal places)
     ???;                         // wait time T
   }
  }

}
```

Fig. 7: The loop( ) routine checks for a message from the PC and then makes a voltage measurement.

- **STEP 4e**: Upload your code and observe the Arduino output using the **Serial Monitor** on the computer.
    - Make sure the bottom right of the window is set to "57600 baud".
    - You should notice the letter "**a**" on the first line.
    - Type in the letter "**y**" in the command line and press the **Send** button.
        - You should see a rapid burst of voltage values appear (see Fig. 8).
        - Every time you send 'y', a new burst of data should appear.
    - If your code works, CLOSE the Arduino software.

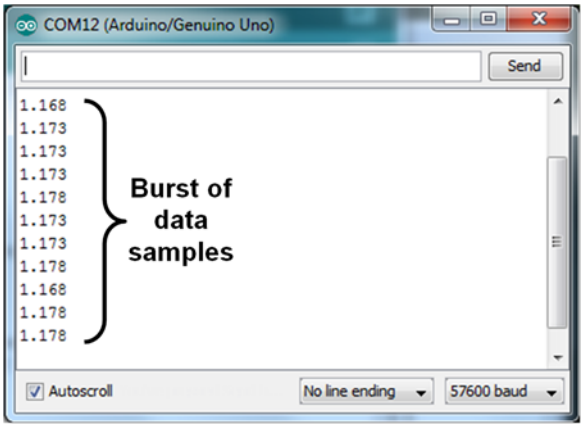


Fig. 8: Sending 'y' to the Arduino should result in a rapid burst of data.

(End of Part 4)

PART 5: MATLAB

OK, now we need to write a MATLAB program to receive data from the Arduino and plot the ECG signal.

- In the "Lab3_files" folder you recently downloaded from the course website, double-click on the "Lab3_MATLAB.m" file.

- **STEP 5.1a**: Replace any "???" in the template so that your code looks like Fig. 9. Comments about the code are shown below:
    - Creating the serial port connection (use the same COM port number as the Arduino code!):
        - We are using the **set** command to configure the **serObj** data rate for **57600** bits per second.
        - We are recording **N = 300** samples for each trace.
        - The burst of data from the Arduino is stored in a memory buffer.
            - ➢ We are setting the input buffer size of **serObj** to be **8*N** bytes.
    - Waiting for the Arduino message.
        - This is the same as before.

```
% Create serial port "object" ================================

serObj = serial('COM12');       % create serial port object
set(serObj,'BaudRate',57600);   % set baud rate to 57600
N=300;                          % number of data points per trace
serObj.InputBufferSize=8*N;     % set input buffer size
fopen(serObj);                  % open the connection
disp('Serial port connected ...'); % display a message

% Wait for message from Arduino ================

ArduinoReady = 'n';     % initially assume message is the letter 'n'
while (ArduinoReady ~= 'a') % check if ArduinoReady is NOT equal to 'a'
    ArduinoReady = fscanf(serObj,'%s'); % read text string from Arduino
end
disp('Arduino is ready ...');   % display a message
```

Fig. 9: Your MATLABcode should look like this. Do NOT forget the semicolons!

- **STEP 5.1b**: Replace any "???" in the template so that your code looks like Fig. 10. Comments about the code are shown below:
    - Initialize the plot window
        - We first need to create a 1-D array of time values for plotting.
            - The time vector **t** has a length of **N = 300** samples.
            - The time increment is **dt = 0.01** seconds.
            - The time vector is therefore [0, dt, 2dt, 3dt, . . . , 299dt] = [0, 1, 2, 3, . . . , N-1]dt
        - We next create an array of voltage values.
            - The voltage vector **Vdata** will initially be defined as a 1-D array of ones.
        - The **figure** command creates a window, where we assign **h** as the figure "name".
        - The **plot** command plots **t** on the x-axis and **Vdata** on the y-axis.
        - **drawnow** displays the plot immediately (this is necessary while the program is running).
        - We want the program to STOP when the user clicks on the figure window. We can do this using the **ButtonDownFcn** in MATLAB.
            - We use the **set** command to configure the **ButtonDownFcn** to do whatever we want.
            - We define a variable **q** that is initially equal to **0**.
            - When the use clicks on figure **h**, we want **ButtonDownFcn** to make **q = 1**.
            - Later in the code we'll see how q = 1 causes a while loop to exit.

```
% Initialize plot window ==============================

dt=0.010;              % time between samples (sec)
t=[0:N-1]*dt;        % row vector of time samples
Vdata=ones(1,N);     % data vector (initialize to bunch of ones)

h=figure;            % create figure window
plot(t,Vdata);       % plot Vmeas vs time
drawnow;             % display the plot immediately
q = 0;                              % initially make q = 0
set(h,'ButtonDownFcn','q=1;');  % mouse click makes q = 1
```

Fig. 10: This part of the code creates 1-D arrays for time and voltage, displays a plot, and configures the "ButtonDownFcn" feature.

```
% Start data acquisition stream ========================

disp('Data acquisition starting ...');

while (q == 0)                % loop continues as long as q is zero
    fprintf(serObj,'%c','y');     % send a single character 'y' to Arduino

    while (serObj.BytesAvailable<(7*N))  % wait until 7*N bytes received
    end

    for m=1:N
        Vdata(m)=fscanf(serObj,'%f');     % read the N samples of data
    end

    plot(t,Vdata);                    % plot the data
    ylim([1,3]);                    % y-axis limits
    xlabel('Time (sec)');             % x-axis label
    ylabel('Amplitude (volts)');      % y-axis label
    title('My awesome ECG signal'); % title
    drawnow;                          % display plot immediately
end

disp('Data acquisition is finished!');

fclose(serObj);
```

Fig. 11: The data acquisition loop does two things: (1) receives data trace from the Arduino (2) displays an updated plot.

- **STEP 5.1c**: Now we want the Arduino to send a complete data trace to the PC! Replace all "???" with appropriate code shown in Fig. 11. Comments about the code are shown below:
    - The while loop keeps running as long as q is equal to 0.
    - Request the data samples from the Arduino by sending the letter '**y**'.
    - Now we need to WAIT for all the data to be acquired by the Arduino and sent to the PC.
        - The while loop runs until there is enough data (**7*N** bytes) in the **serObj** memory buffer.
            - This is done with **serObj.BytesAvailable < 7*N**.
    - We now run a **for** loop to read the received data.
        - Each array element in Vdata is updated with a received floating point data value.
    - Plotting the updated data is the same as before.
    - The last section of the program ("Finish up the process") is the same as before.

- **STEP 5.1d**: Save your program, then run it by clicking on the "Run" button (with green triangle) at the top of the MATLAB window.
    - An initial plot should appear showing a flat line with amplitude = 1.
    - After a few seconds, a new plot should appear showing your amplified ECG signal.
    - **If you get MATLAB errors, remember to type "fclose(serObj)" in the MATLAB command line BEFORE debugging your code!**

(End of Part 5 … go to next page for final ECG measurement!)

PART 6: ECG MEASUREMENT

- **STEP 6.1**: Disconnect the benchtop supply from your breadboard.
    - Insert two battery connectors to the breadboard (see Fig. 12).
        - For +9V power:
            - +9V = RED wire
            - Gnd = BLK wire
        - For the -9V power:
            - -9V = BLK wire
            - Gnd = RED wire
    - Attach a 9V battery to each connector (ask Buma for batteries).
    - Run your MATLAB code to make sure everything works.

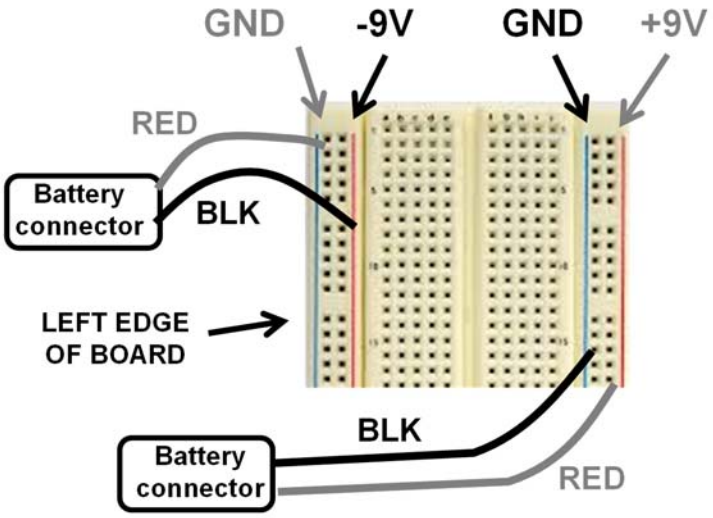


Fig. 12: An example of how to attach the battery connectors to your breadboard. This schematic assumes -9V and GND are on the left vertical "strip" while +9V and GND are on the right vertical "strip".

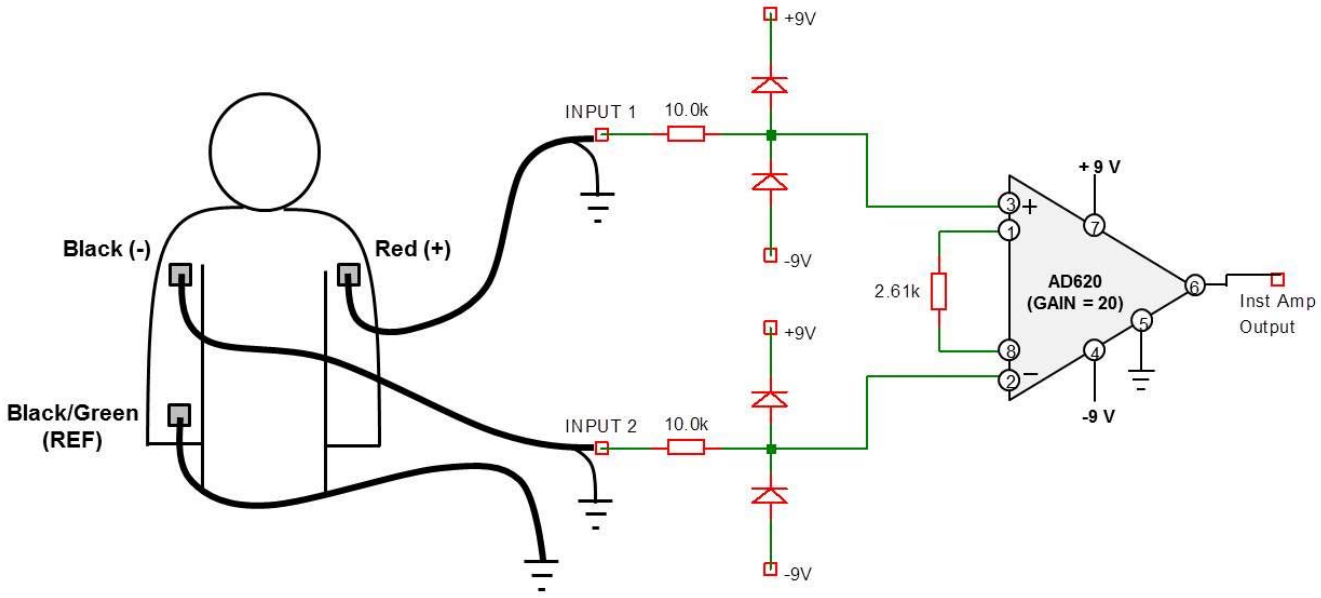


Fig. 13: ECG measurement using shielded cables and electrodes (only first part of ECG circuit is shown).

STEP 6.2: OK! We are now ready to do a live ECG recording!

- Remove the 100:1 voltage divider from your board.
- Unground "INPUT 2" of your board.

- Buma will give you three shielded cables and electrodes. Attach them according to Fig. 13.
  - Firmly press each electrode to your skin.
    - Poor electrode-skin contact will degrade the ECG measurement!
  - Red clip goes to upper left arm. The cable's yellow wire goes to "INPUT 1" of your board.
  - Black clip goes to upper right arm. The cable's yellow wire goes to "INPUT 2" of your board.
  - Black/Green clip goes to lower right arm. The cable's yellow wire goes to BATTERY GND of your board.
  - The green wires of all three cables are connected to **BATTERY GROUND**!
- Be very still and relaxed during the ECG recording -- hopefully your PQRST waveform is on the computer!
- Record a snapshot of your ECG waveform for your lab report.
  - If you want to adjust the y-limit values of your plot to better display the waveform, you should STOP the data acquisition before editing the code.
- Take a photo (e.g. with camera phone) of your working circuit for your lab report.
- Demo your system to Buma.

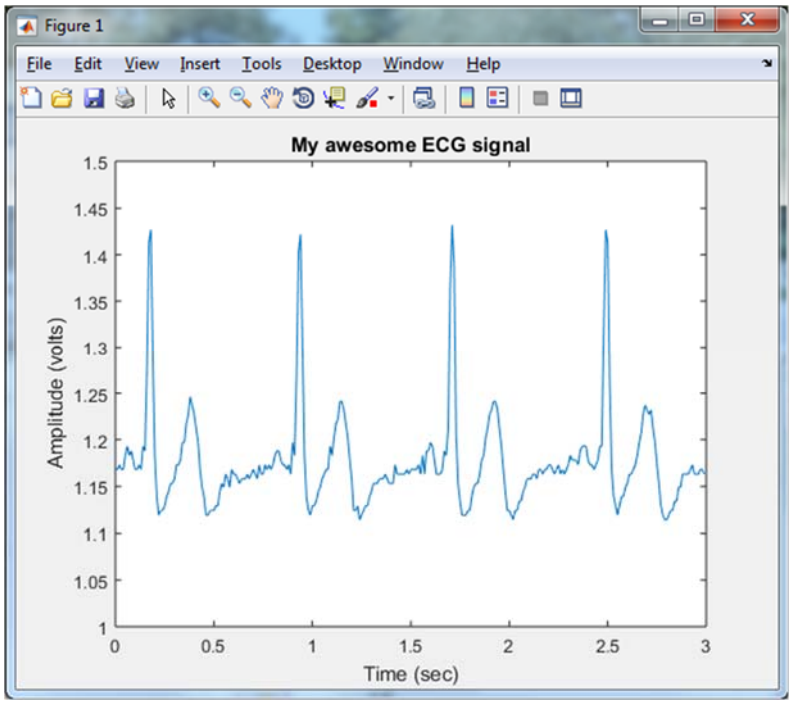


Fig. 14: Buma's ECG recording. If you want to adjust the y-limit values of your plot, you should STOP the data acquisition before editing the code. Be VERY STILL AND RELAXED to get a nice clean waveform!

(End of Lab3)