Capstone Preliminary Report:

PPG part

PPG (photoplethysmogram) Sensor:

Background & Theories:

As a crucial part of patient monitoring system, PPG sensor, known as pulse oximeter is used to measure the SpO2 level. “SpO2 stands for peripheral capillary oxygen saturation, an estimate of arterial oxygen saturation, or SaO2, which refers to the amount of oxygenated haemoglobin in the blood.” [1].

The blood oxygen saturation reading of a normal person should be between 95% and 100%. [1]. A respiratory or cardiovascular problem may be present if the oxygen saturation drops to 90% ~ 95%. The patient will highly likely experience hypoxic if the oxygen saturation falls under 90%. [2]

Working principle of the infrared PPG sensor:

Photoplethysmogram Sensor (PPG sensor) takes advantage of the different absorption level of Oxyhemoglobin (HbO2) and deoxyhemoglobin (Hb) **for the light beams with different wavelength.** By placing a pair of infrared and red LED on one side of the finger and a receiver on the other side, the variation of light intensity can be measured. The light transmitting through fingertip will be absorbed by pulsatile arterial blood, non pulsatile arterial blood, venous blood and tissue. [5] As we know the pulsatile arterial blood is varying according to the heart pulse, hence the light absorption is also varying according to the heart pulse. Such variation is used as the waveform of PPG signal.

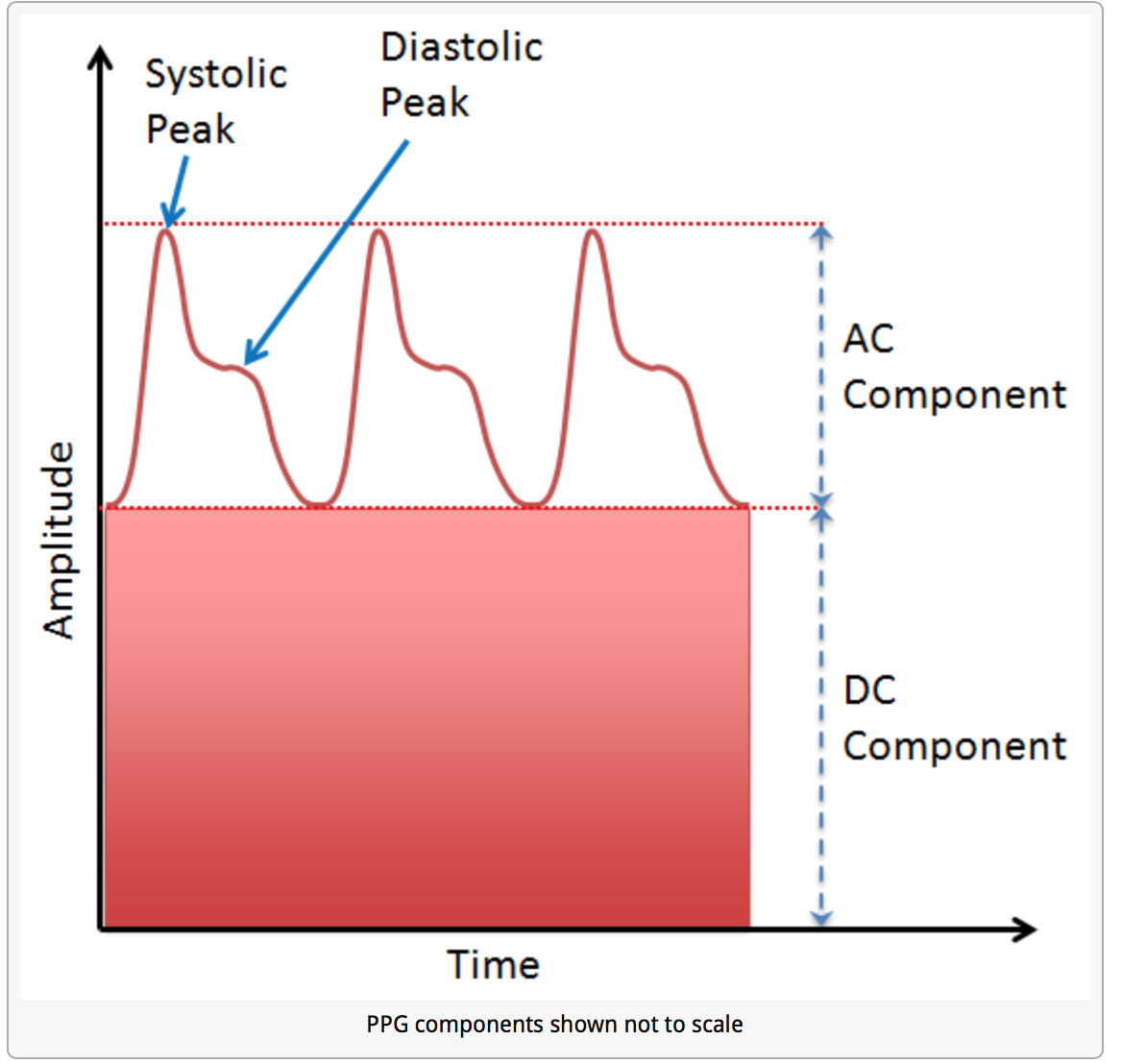


Figure XXX, [6]

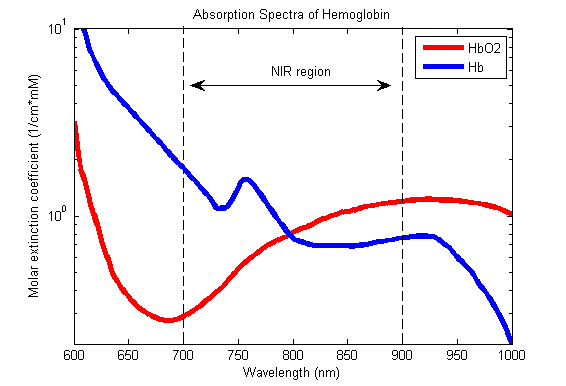


Figure XXX , [3]

The Oxygen saturation level can be calculated by the following equation:

* **Oxyhemoglobin.**
* **deoxyhemoglobin.**

The pulse oximeter uses **Beer–Lambert law** [4][5] to define relationship between the light attenuation (light absorption) and the material it travels through. [7]

A normalized ratio of the Red to IR **then can be calculated**:

* **AC component of Red light waveform.**
* **DC component of Red light waveform.**
* **AC component of Infrared light waveform.**
* **DC component of Infrared light waveform.**

**The accurate SpO2 is proportional to this ratio. Proper calibration can be done to get the accurate SpO2 reading. [10]**

**Sensor Design & measurement:**

**When we tried to obtain the PPG waveform** by directly connecting a pair of infrared transmitter and receiver with the Arduino, we were not able to visualize any valid signals. This could be due to two main reasons:

1. The photoplethysmogram signal is very small in amplitude compare with the DC offset component in the carrying signal.
2. The pair of sensors is not working properly.

To eliminate the second possibility, we did some testing with the sensors along. The intensity of the infrared beams received at the receiver is proportional to the separation between the transmitter and the receiver. By moving the transmitter towards and away from the receiver, we can observe instant change of the readings.

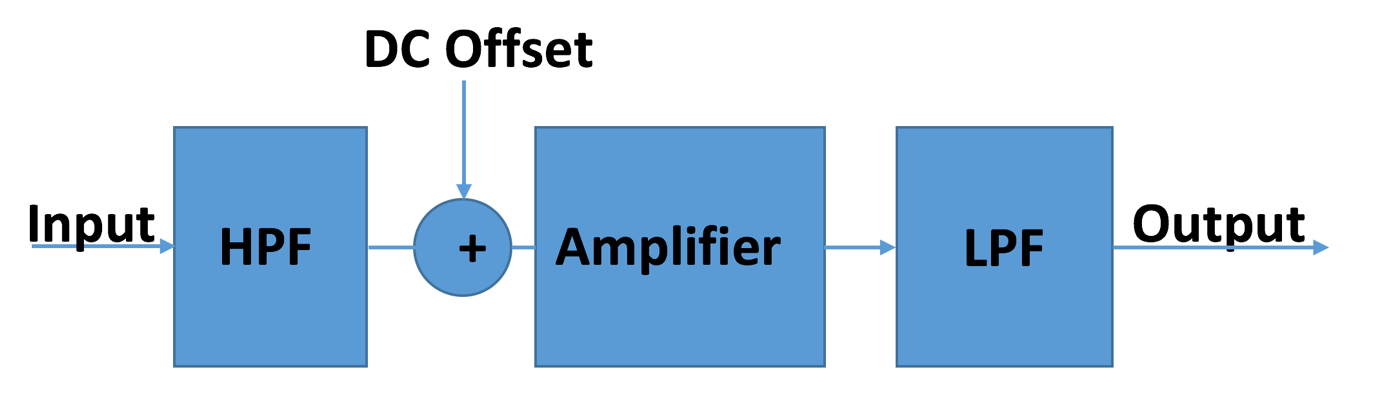
**Hence, we come to the conclusion that the signal from our PPG sensor is very small in amplitude compare with the DC offset and the high frequency noise. Therefore, low-pass filter, high-pass filter and op-amp are used to extract the useful information from the “raw” signal from PPG sensor.**

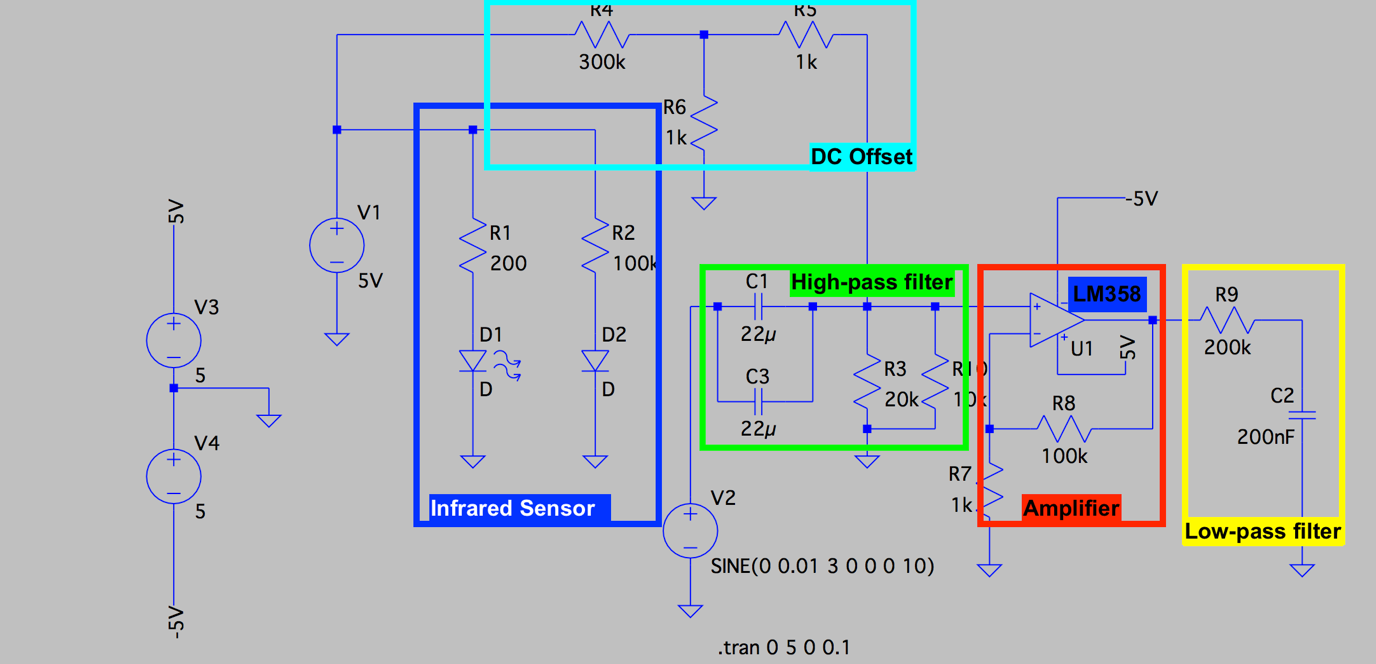
For the Arduino we are using, it has 10-bit analog to digital convertor which maps the analog input between 0 to 5V into 1024 discrete levels between 0 and 1023. This gives the resolution around 5mV per unit. **Hence, in order to obtain a clear visualization of the PPG signal, the AC component needs to be amplified to few volts in amplitude (less than supply voltage 5V).**

Bandwidth selection:

The frequency of the normal resting heart rate is between 1Hz to 2Hz. [8] Hence, for our Pulse Oximeter, we allow the signal between 0.5Hz to 4Hz to pass through. **Extra margin is designed for special cases.**

**Circuit design:**

****



High pass filter:

A RC high pass filter is designed to meet the lower bound of the bandwidth requirement.

In our design, we choose R = 6666 ohms (20k and 10k in parallel) and C = 44 uF which gives us a cut-off frequency of 0.543 Hz.

Low pass filter:

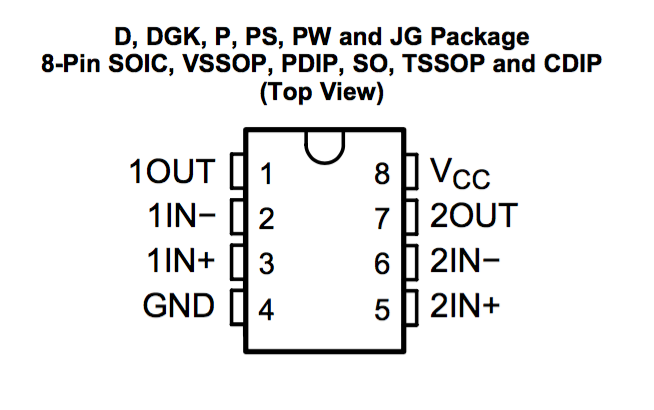
A RC low pass filter is designed to meet the upper bound of the bandwidth requirement.

In our design, we choose R = 200k ohms and C = 200nF which gives us a cut-off frequency of 3.98 Hz which is close to 4 Hz.

**Amplifier:**

**Since we are not able to observe any waveform from the original signal, the amplitude of the original AC component must have magnitude of few mV because the resolution of the Arduino is 5mV. If the oscillation of the original signal has few hundred mV in amplitude, the waveform would be clearly observed without any signal processing. In order to amplify the signal to the magnitude of volts, we design an amplifier with gain of 100 which gives the output in Volts.**

**Another limitation is the power supply from the Arduino UNO. A normal operational amplifier requires positive and negative voltage for its supply rails. But Arduino UNO only provides positive 5V and GND. LM358 op-amp, which is used in our prototype, requires only positive voltage supply which is perfectly compatible with Arduino.**

****

**LM 358, [12]**

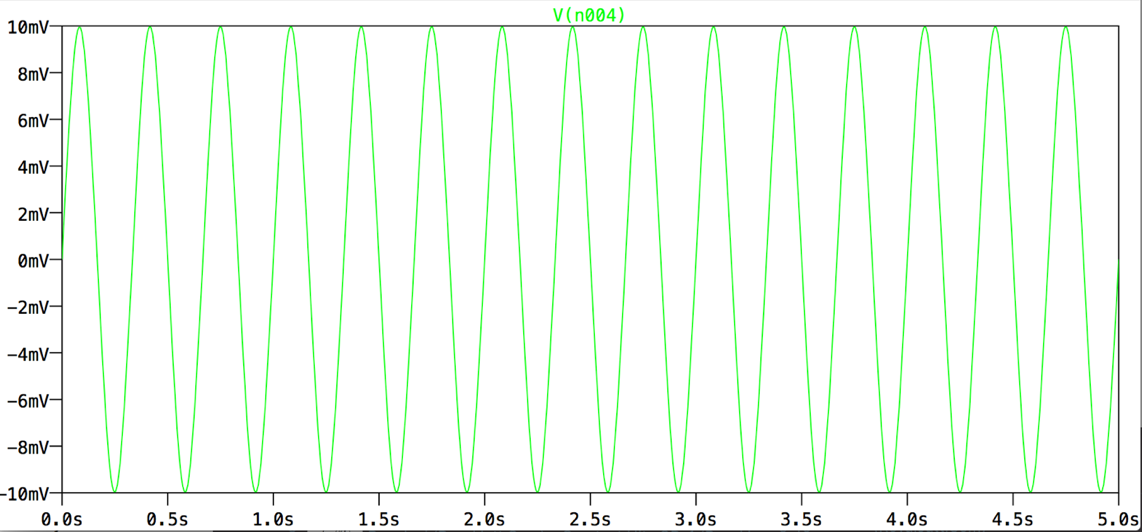
Offset voltage:

**To avoid signal being saturated at 0, an offset voltage is supplied to the input signal before the amplifier. Since the Arduino UNO can detect values between 0 ~ 5V, it is reasonable to have an offset voltage of 1.5V.**

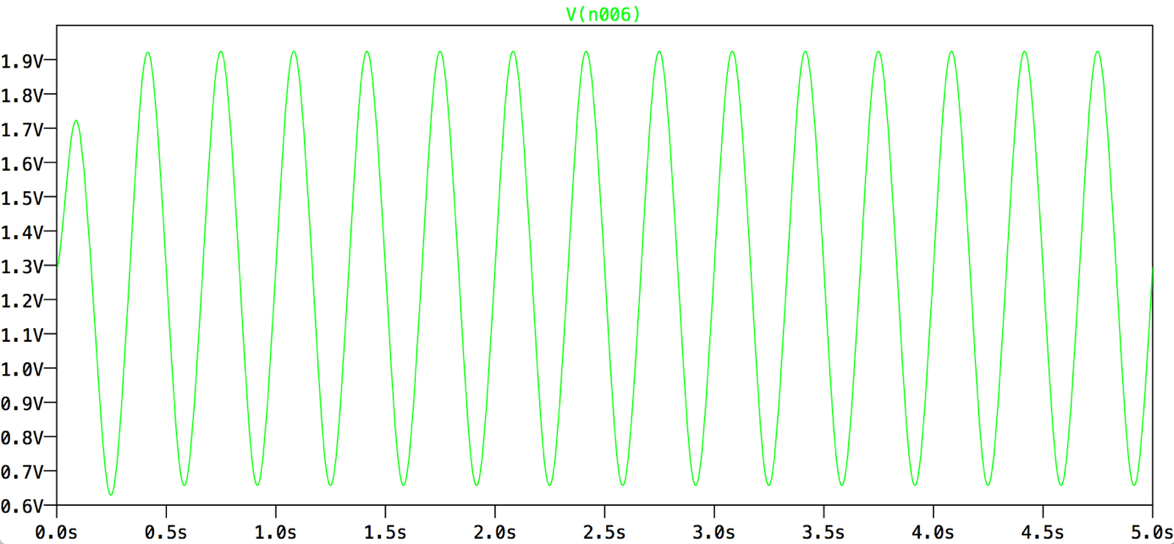
In order to achieve 1.5V offset at the output, the pre-amplified offset is set to be 1.5V/100 = 15mV, where 100 is the gain of the amplifier. Voltage divider is used to get 15 mV from a 5V source.

LTSpice simulation:

A sinusoidal input with frequency 3 Hz with peak-to-peak voltage 20 mV is used for simulation:



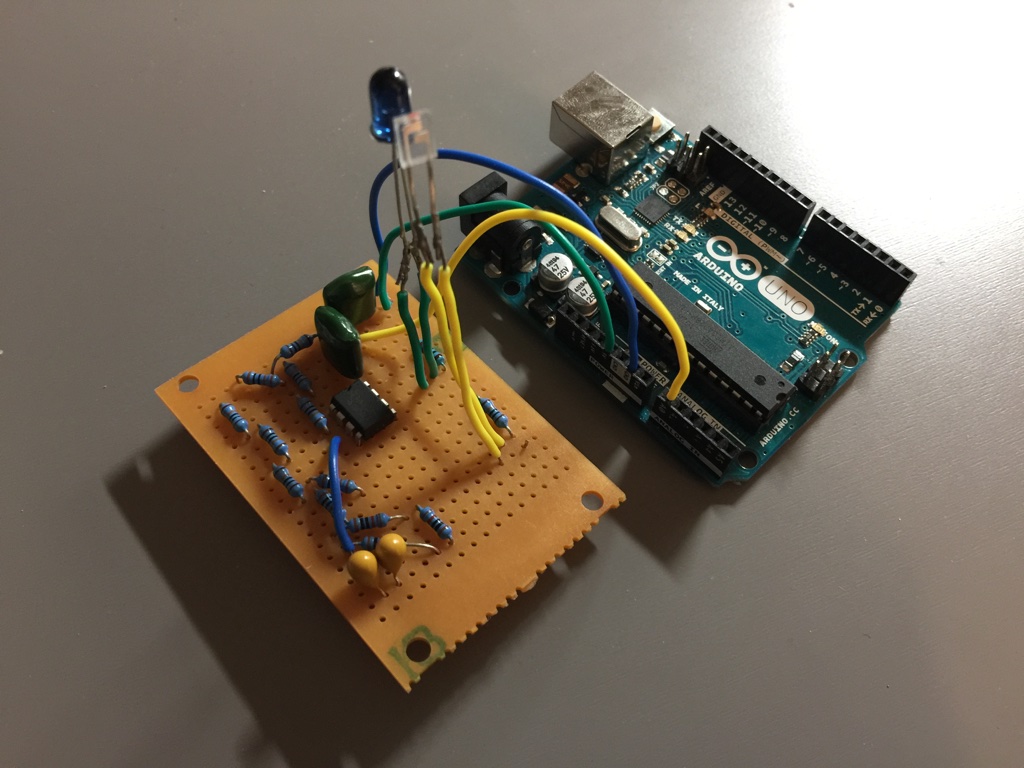
The output is:



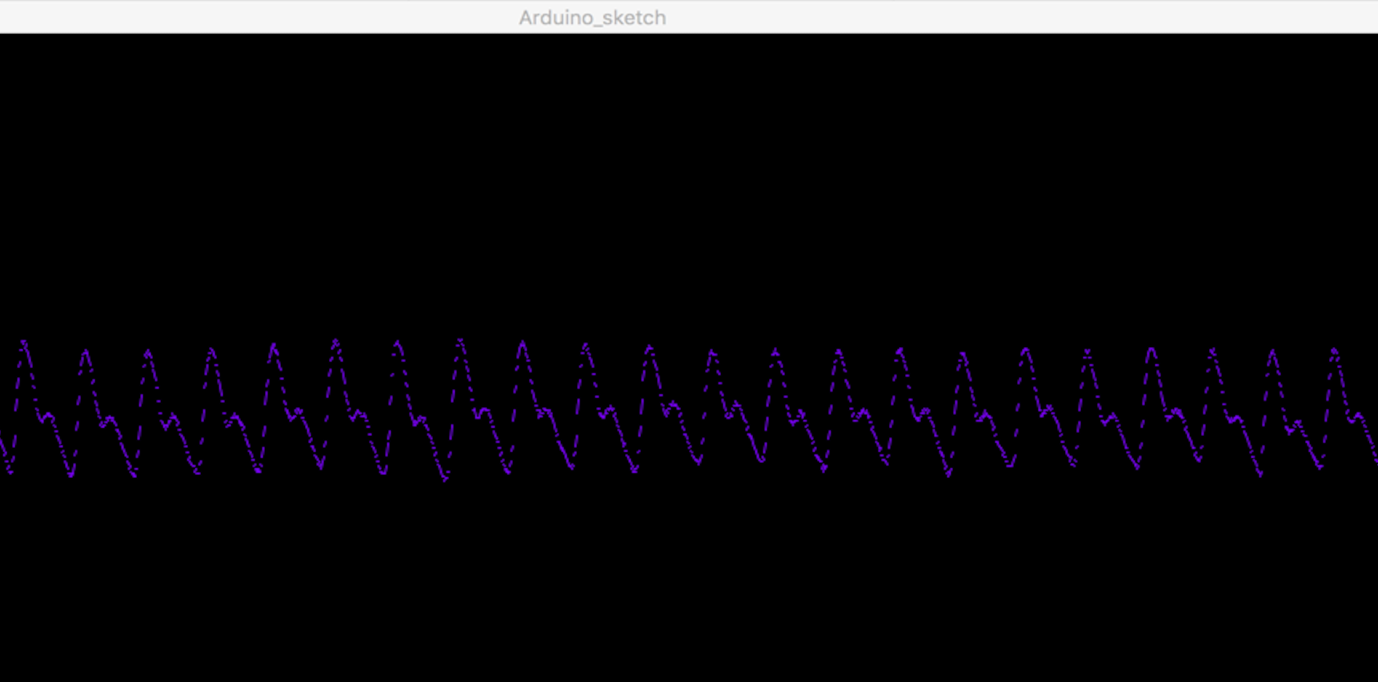
From the output plot, we are able to see the input sinusoidal signal has been amplified to roughly 1.4V peak-to-peak with 1.3V DC-offset.

Results:

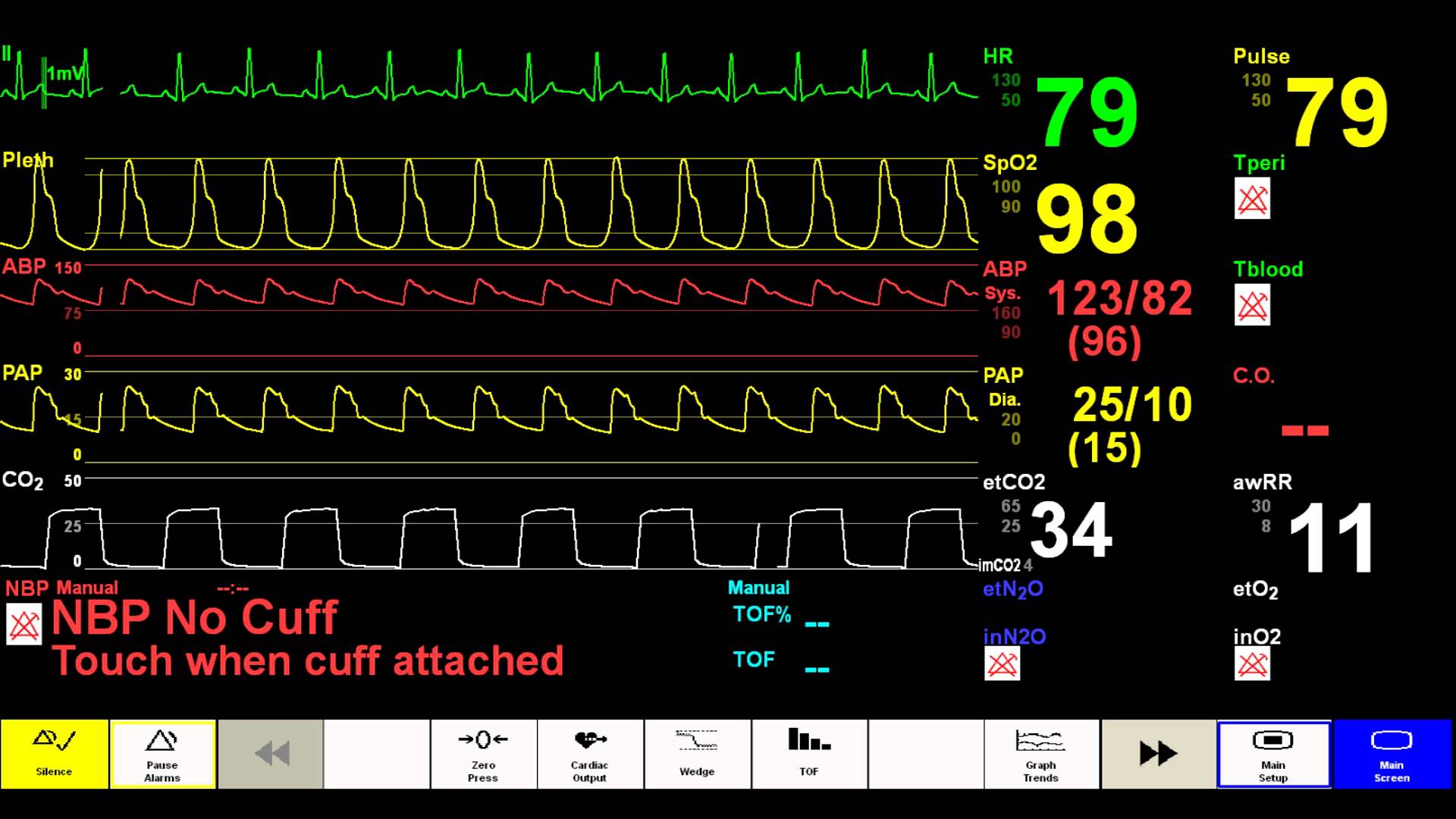
Prototype sensor with Arduino UNO:



Processing 2+ is used for waveform visualization:

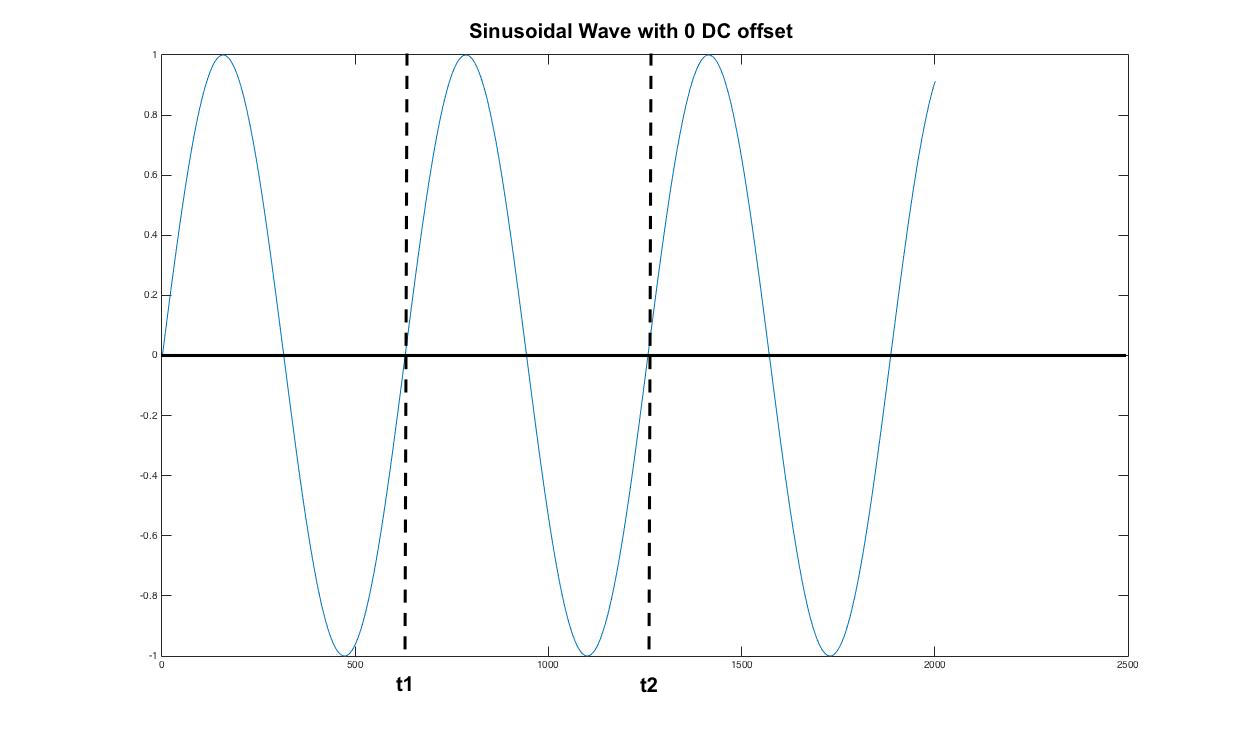


Comparison with the PPG wave from medical standard monitor display:



**Heart rate measurement:**

**The basic idea of calculating heart rate with the data collected by pulse oximeter is measuring the period of each periodic pulse and calculate the frequency. For normal sinusoidal waves, this is very easy to be achieved by finding the DC offset of the waveform and measure the time difference between every time the signal goes above that DC offset. Then the frequency can be calculated from the time difference.**

****

**However, the pulse oximeter is very sensitive to any physical movement. The amplitude of the waveform is relatively stable if you keep your finger at the same position. But the movement of the pulse oximeter itself can cause the change in DC offset of the signal.**

**In order to mitigate the fluctuation of DC offset, we implement a moving average filter with window size of 400 samples. For normal human resting heart rate, each periodic pulse takes approximately 200 samples. With a window size of two periods, the moving average can be used as the DC offset for calculating the heart rate.**

* **Moving average DC offset (Matlab plot):**

****

**From the plot above, we can observe the moving average line behaves as a trend line for the PPG waveform.**

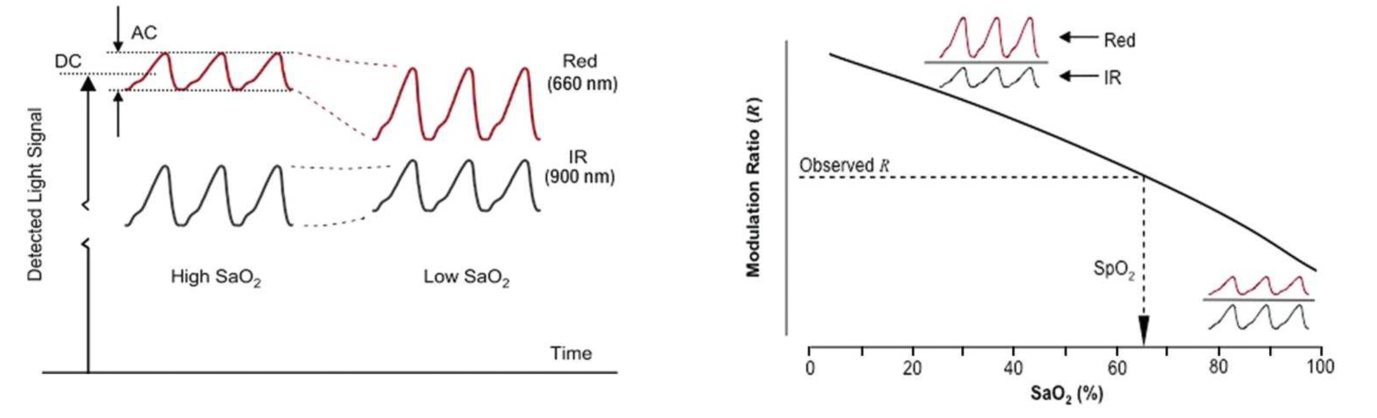
**Abstract AC/DC values:**

**As explained in the previous, the modulation ratio R is the key parameter for calculating Oxygen Saturation level SpO2.**

**For DC component value for Red/IR signals, the moving average can be used here as the.**

**The**

**…….. (might need one more page later)**

****

**Figure XXX, [10]**

Citations:

[1] <https://withings.zendesk.com/hc/en-us/articles/201494667-What-does-SpO2-mean-What-is-a-normal-SpO2-level->

[2] <http://www.ruralareavet.org/PDF/Anesthesia-Patient_Monitoring.pdf>

[3] <https://commons.wikimedia.org/wiki/File:Oxy_and_Deoxy_Hemoglobin_Near-Infrared_absorption_spectra.png> (picture)

[4]<https://www.researchgate.net/publication/220043789_A_Real_Time_Analysis_of_PPG_Signal_for_Measurement_of_SpO2_and_Pulse_Rate>

[5] <http://www.nxp.com/files/32bit/doc/app_note/AN4327.pdf?tid=AMdlDR>

[6] <http://embedded-lab.com/blog/easy-pulse-version-1-1-sensor-overview-part-1/> (picture)

[7] <https://en.wikipedia.org/wiki/Beer–Lambert_law>

[8] <http://www.mayoclinic.org/healthy-lifestyle/fitness/expert-answers/heart-rate/faq-20057979>

[9] “Heart rate measurement through PPG, Heartbeat measurement in a wireless headset” – J.Guyomard & Stortelder

[10] <http://www.ti.com/lit/an/slaa655/slaa655.pdf>

[11] <http://playground.arduino.cc/Main/RunningAverage>

[12] <http://www.ti.com/product/LM358>

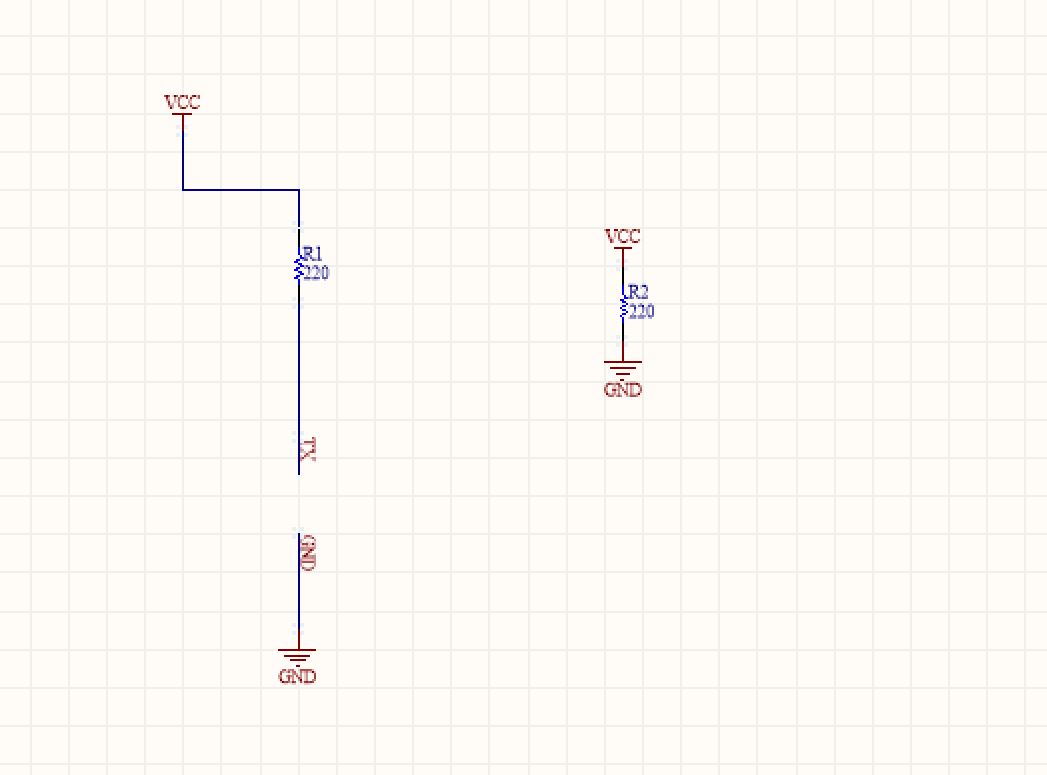
[13] <https://i.ytimg.com/vi/jcXidSeirMU/maxresdefault.jpg>

[14]

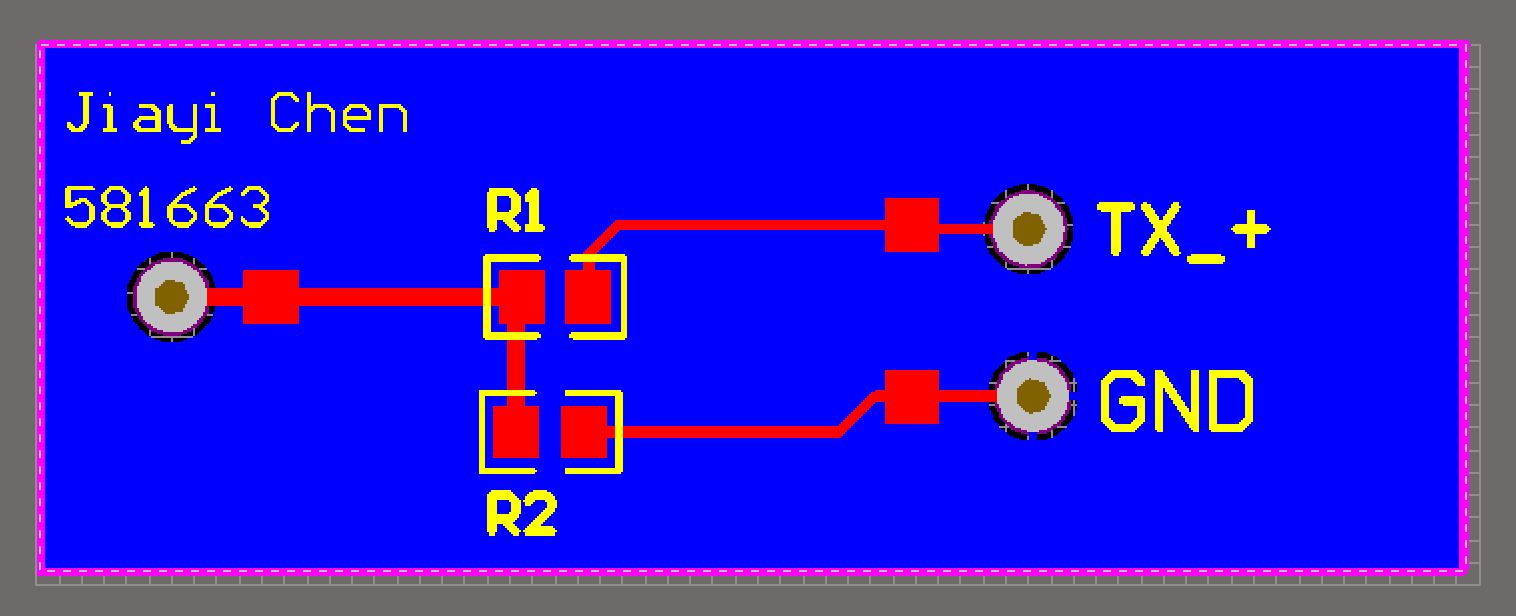
**Appendix:**

PCB design:

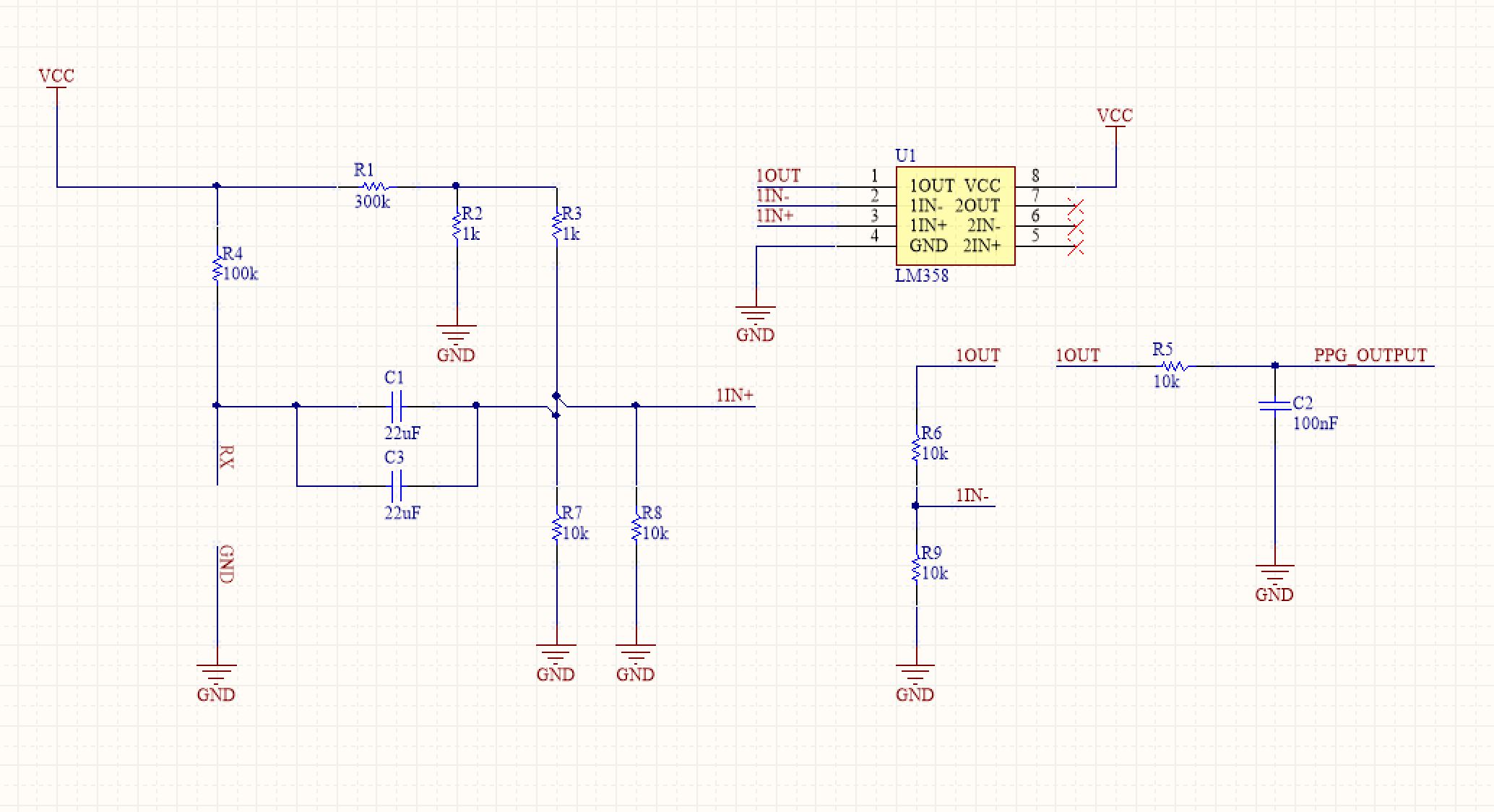
Transmitter Schematics:



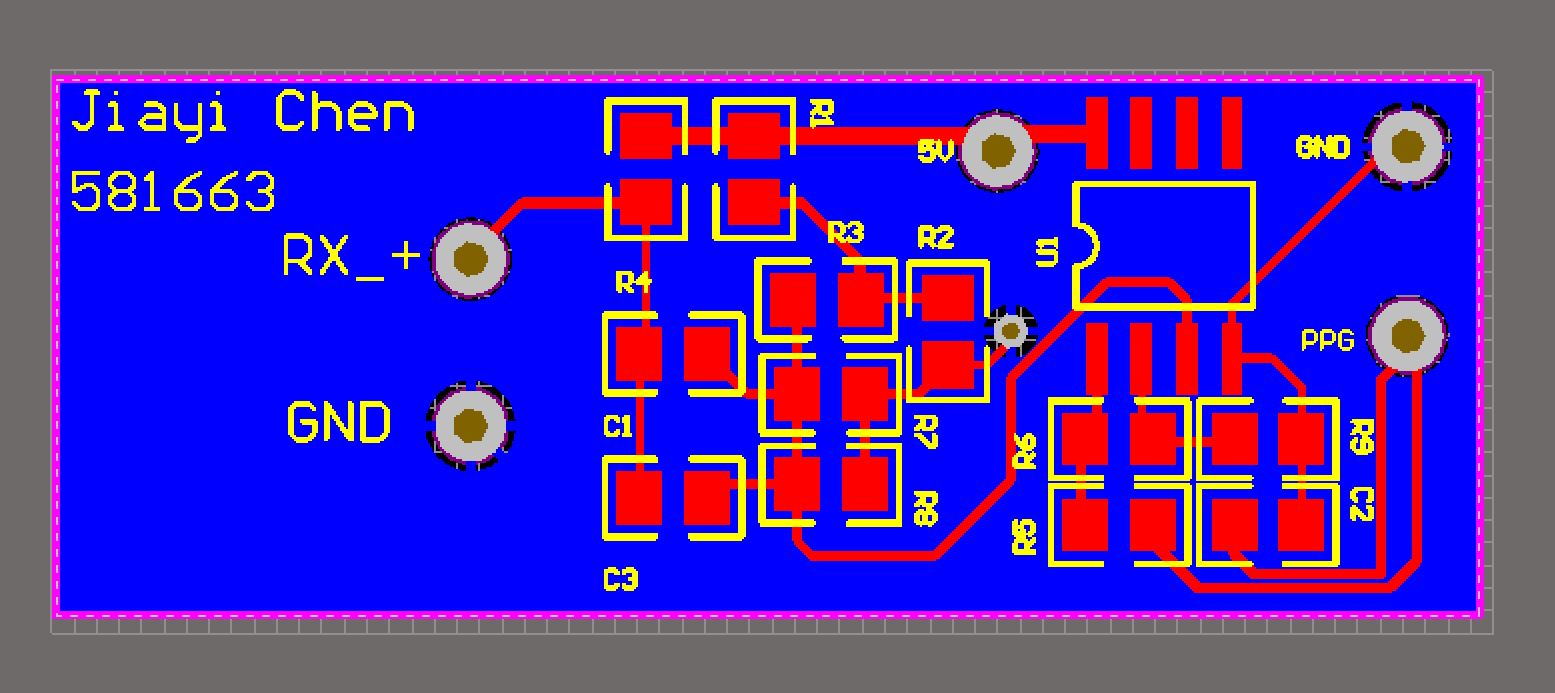
Transmitter PCB:



Receiver Schematics:



Receiver PCB:



Code for processing 2+:

// PPG with Processing

// Created by Jiayi Chen

// Date: 29-09-2016

import processing.serial.\*;

Serial myPort; // The serial port

int xPos = 1; // horizontal position of the graph

float inByte = 0;

float oldinByte = 0;

PrintWriter output; // Create an object for exporting data

PrintWriter output\_average;

int LM\_SIZE = 750;

float[] LM = new float[LM\_SIZE]; // LastMeasurements

int index = 0;

static float sum = 0;

int count = 0;

float movingAverage = 0;

double previous\_time = 0;

double current\_time = 0;

double time\_difference = 0;

double heart\_rate = 0;

float input;

int peak = 0;

int trough = 0;

void setup () {

// Create a new file in the sketch directory

//output = createWriter("infrared\_data.txt");

//output\_average = createWriter("movingAverage.txt");

// set the window size:

size(980, 420);

// List all the available serial ports

// if using Processing 2.1 or later, use Serial.printArray()

println(Serial.list());

// I know that the first port in the serial list on my mac

// is always my Arduino, so I open Serial.list()[0].

// Open whatever port is the one you're using.

myPort = new Serial(this, Serial.list()[2], 9600);

// don't generate a serialEvent() unless you get a newline character:

myPort.bufferUntil('\n');

// set inital background:

background(0);

}

void draw () {

// draw the line:

stroke(127, 34, 255);

//if (inByte > 30 && inByte < 600) {

line(xPos-1, (height-oldinByte), xPos, (height-inByte));

//text("received: " + inByte,10,50);

// at the edge of the screen, go back to the beginning:

if (xPos >= width) {

xPos = 1;

background(0);

} else {

// increment the horizontal position:

xPos++;

}

//}

}

void serialEvent (Serial myPort) {

// get the ASCII string:

String inString = myPort.readStringUntil('\n');

if (inString != null) {

// Store the old value for line drawing purpose

oldinByte = inByte;

// trim off any whitespace:

//inString = trim(inString);

// convert to an int and map to the screen height:

inByte = float(inString);

input = inByte;

//println("Input = " + input);

movingAverage = runningAverage(input);

//println(" Average = " + movingAverage);

//output.println(input);

//output\_average.println(movingAverage);

inByte = map(inByte\*30, 0, 800, 0, height-500);

// Wait 3 seconds before calculating heart rate

if ( millis() > 3000){

// Calculating heart rate

// Detects peak of the period

if (input > movingAverage && (peak == 0)){

current\_time = millis();

time\_difference = current\_time - previous\_time;

//println("Current\_time = " + current\_time);

//println("previous\_time = " + previous\_time);

heart\_rate = 60.0/(time\_difference\*0.001);

previous\_time = current\_time;

println("heart rate = " + heart\_rate);

peak = 1;

trough = 0;

}

// Detects trough of the period

if (input < movingAverage && (trough == 0)){

peak = 0;

trough = 1;

}

}

}

}

// runningAverage for heart rate counting

float runningAverage(float M) {

// keep sum updated to improve speed.

sum -= LM[index];

LM[index] = M;

sum += LM[index];

//println("Sum = " + sum);

index++;

index = index % LM\_SIZE;

if (count < LM\_SIZE) count++;

return (sum / count);

}