BME 354 Final Project

Brian Huynh, Michael Rees
BME 354 - 03L
May 1st, 2014
TA: Kristie Yang

We have adhered to the Duke Community Standard in completing this assignment.

Contents

List of Figures

List of Tables

1 Introduction

Pulse plethysmography allows for the simple measurement of heart rate by examining the pulsatile volume changes of blood in the vascular system. Clinically, pulse plethysmographs provide an easy way for doctors and physicians to get a basic, quantitative measurement of a patients physical status. A pulse plethysmograph can easily be extended to also include functionality for quantifying oxygen saturation levels by comparing the optical transmittance of both red and infrared light through pulsatile blood.

2 Background

A pulse plethysmograph takes advantage of the Beer-Lambert Law, shown below, by applying it to measured volume changes using light transmittance.

$$I = I_0 e^{-\epsilon(\lambda)cd} \tag{1}$$

The Beer Lambert law relates the amount of light absorbed by a uniform medium to the concentration of a known substance within where I is the received light intensity, I_0 is the incident light intensity, $\epsilon(\lambda)$ is the absorption of the substance at wavelength, c is the concentration of the substance, and d is the optical path length. As the hearts ventricles contract, the pulsatile blood and change in blood pressure results in varying transmittance values in the blood vessels. This change in pulsatile pressure is detected by illuminating a light-emitting diode (LED) on one side of the finger, at either visible red or infrared wavelengths, and measuring transmissive absorption or the amount of light transmitted to a photodiode on the other side of the finger.

The measured signal can be separated into two components. The DC (direct current) portion of the signal is attributed to the transmissive absorption of the skin and tissue while the AC (alternating current) portion of the signal is from the variation in blood volume underneath the section of skin measured due to the output from the cardiac cycle. A secondary pressure pulse can sometimes be identified in addition to the primary peak as a result of venous pulsations. Common cardiac arrhythmias such as premature ventricular contractions (PVCs), ventricular tachycardia, and ventricular fibrillation (V-fib) can also be identified using a pulse plethysmograph.

The oxygen saturation of the blood can be determined via light transmittance as well. Deoxygenated blood absorbs more red light, while oxygenated blood absorbs more infrared light. By taking measurements with both infrared and red LEDs, the ratio of the two measurements will yield the blood oxygenation of the patient in a safe, and non-invasive manner.

The Arduino Uno is a single-board microcontroller driven by an ATmega328 microcontroller chip at a clock speed of 16 MHz and an operating voltage of 5V that runs on C++ with a total of 20 input and output pins for both digital and analog (pulse-width modulated) signals. The ATmega328 contains 32 KB of flash memory (0.5 KB of which is used by the bootloader), 2 KB of SRAM (static random access memory), and 1 KB of EEPROM (electrically erasable programmable read-only memory). Arduino boards can be configured with printed circuit expansion boards titled shields which can expand the functionality of the microcontroller by providing additional features such as an LCD screen for button controls and visual display or feedback. The microcontroller is run on a cyclic executive program uploaded using serial communication with a computer over USB where there is a `setup()` method that runs once to at the start of the program, usually containing initialization statements, and a `loop()` method that runs infinitely until power is disconnected.

3 Materials & Methods

- Arduino Uno: A flexible and easy to use software and hardware interface allowed for rapid prototyping and development.
- Breadboard: A prototyping board allowed for easy construction and rearrangement of electronic components. Because it is a solderless device, both the board and the component are reusable and can be connected at different nodes using different breadboard arrangements.
- LMC662 Operational Amplifiers (2): Two LMC662 CMOS Dual Operational Amplifiers were used in the medical instrumentation as high-impedance buffer or preamplifiers and a precision current-to-voltage converter. It operates from +5V to +15V. A capacitive load configuration with the LMC662 is most sensitive to the oscillation in a unity-gain follower. Therefore, no unity gain buffer amplifiers were used in this circuit.
- Resistors: 1 kΩ (3), 2 kΩ, 5.7 kΩ, 20 kΩ, 82 kΩ, 330 kΩ, 1 MΩ (2), 3.3 MΩ
- Capacitors: 4.7 nF, 220 nF (2)
- Nellcor Oxisensor ®II D-25: A pulse oximeter probe was provided for use in this project. The head of the probe contained a single photodiode and two adjacent LEDs emitting visible red and infrared light connected in an antiparallel configuration (i.e., the cathode of one LED is at the same internal node as the other LEDs anode). The photodiode within the probe was a transducer converting light to an electrical signal, either voltage (where a large impedance load is driven by the photodiode) or current (where the driven load has as low of an impedance as possible).
- Virtual Ground: Because the circuit was being driven by an Arduino microcontroller, a virtual ground rail at V = 2.5 V was created via voltage division of two identical resistors. This added a DC offset to the plethysmograph signal to center it about 2.5V so that positive and negative deflections in the signal would still be within range of the Arduino input pins (0-5V). Otherwise, if the signal did not contain a DC offset, negative deflections would simply be read in as 0 creating an artifact in the signal.

3.1 Coding Methods

The Arduino microcontroller was programmed using C++ separated into a `setup()` method used to initialize pin modes, the serial monitor, the LCD shield, and create a custom character for a blinking heart, and a `loop()` method which controlled the sampling of the input signal, all signal processing, and manipulation of the LCD screen.

As can be seen in the above oscilloscope trace, one of the most prominent characteristic of the measured signal was a pronounced downward deflection in the signal after a steady increase. This downward deflection takes roughly 0.2 seconds to be completed (5 Hz), and thus a 10 Hz sampling rate was used, in order to be in accordance with the nyquist sampling rate, which states that the signal must be sampled at twice the frequency of the highest frequency component that you wish to capture. While from an engineering standpoint, it may have been more prudent to sample at 2-3 times the nyquist sampling rate to ensure a robust measurement of the desired downward deflection, sampling at exactly the nyquist frequency allowed us a slow enough acquisition time to eliminate some of the high frequency noise in the signal, that resulted from movement of the transducer, or changes in the ambient light reaching the photodiode.

In order to capture this downward deflection, a first difference algorithm was used to determine the heart rate from the measured data. A measurement was taken at the sampling frequency, and was stored for the next cycle of the void loop. Then, this value was compared to the next measurement taken via a first difference. If the first difference was negative, and fell in between a high and low threshold values, then the a heartbeat was taken to have occurred at that data point.

Additional functionality was implemented into the software by utilizing the various buttons on the LCD shield. The brightness of either LED (visible red light or infrared light) could be modulated by pressing the DOWN button. The LED was initialized to a brightness level of 5 out of 10. Since different users may have differing baseline transmissive absorbance, feedback control was implemented to allow for a more accurate signal independent of the user. Two seconds of sampled data at 10 Hz were recorded to identify the minimum and maximum voltages in the signal. If the difference in blood pressure deflection was less than 0.1V, the LED brightness was incremented up to a maximum brightness of 10/10 or HIGH. If the difference in blood pressure deflection was greater than 1V, the LED brightness was decremented to a minimum of the LED being off or at 0/10. Each calibration cycle will only change the LED brightness by a single increment.

The elapsed time in seconds since the Arduino was turned on was displayed by pressing the LEFT button. The mean heart rate since the Arduino was turned on was displayed by pressing the RIGHT button. The total number of heartbeats detected via peak detection was continually summed and divided by the elapsed time to calculate the mean heart rate. Lastly, the O2 saturation was calculated by measuring 5 seconds of LED and IR data and detecting minimum and maximum peaks to calculate the average peak-to-peak voltage. The ratio of the average peak-to-peak voltage of LED to IR data was then converted into an approximate oxygen saturation level by applying a data gathered from an estimated linear fit.

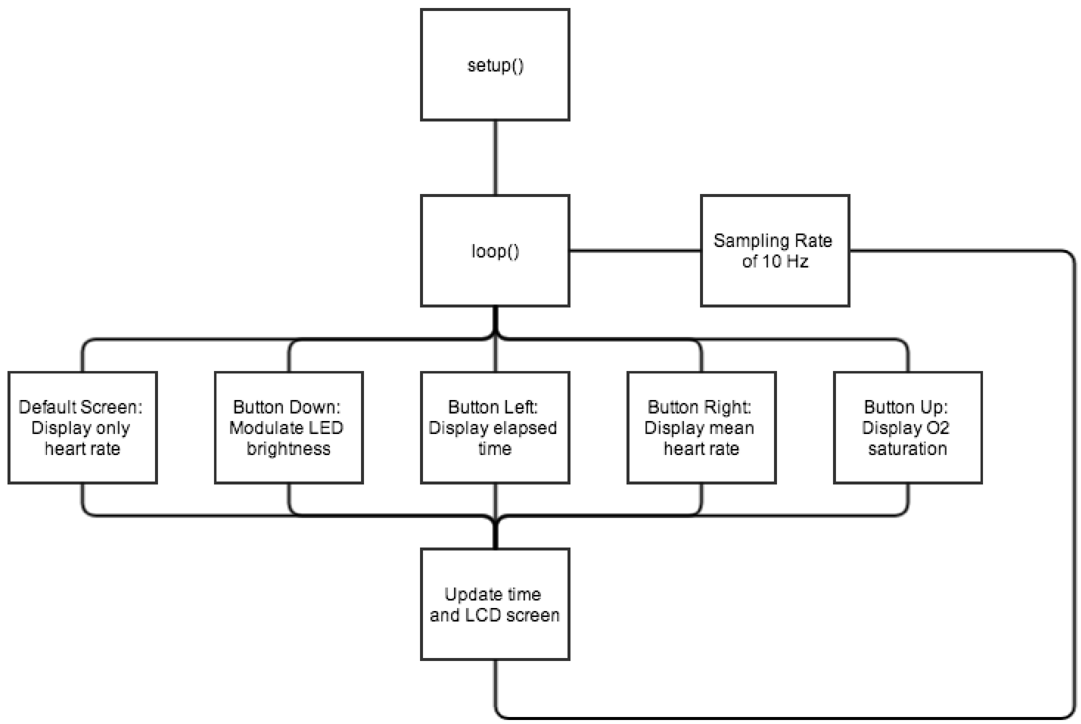

Figure 1: Coding Structure

4 Results & Discussion

The cutoff frequencies (ω_c) and amplifications used for the signal processing stages are listed in Table 1.

Table 1: Signal Processing Stages and Components

Signal Processing Stage	Circuit Elements	Gain	Cutoff Frequency
Transimpedance amplifier	$R = 1M\Omega$ C = 4.7 nF	N/A	N/A
Non-inverting amplifier	$R_f = 1M\Omega$ $R_i = 330k\Omega$	3.03	N/A
High pass filter	$R = 1 = 3.3M\Omega$ $C = 220nF$	N/A	0.219 Hz
Non-inverting, active low pass filter	$R_f = 82k\Omega$ $R_i = 5.7k\Omega$ $C_f = 220nF$	14.39	9.11Hz

The passband of the circuit was defined to be from 0.219-9.11 Hz. The overall gain of the circuit is equal to the product of the gains of the individual stages.

$$\text{Total Gain} = \frac{V_{out}}{V_{in}} = 3.03 * 10 * 14.39 = 436 = 52 \text{ dB}$$

The box diagram of the circuit is shown in Figure 2. The first stage of the circuit implements a transimpedance amplifier as a current-to-voltage converter by utilizing the high current impedance of the op-amp and the voltage drop between the negative input to the output being equal to the product of the photocurrent and the resistance of the feedback resistor. A bandpass filter was implemented by two consecutive stages of first a high pass and then a low pass filter buffered by non-inverting amplifiers to prevent downstream impedances from affecting the gain or the signal. While the high pass filter and the active low pass filter could theoretically have been switched to achieve the same passband, a reverse implementation would have resulted in a gained DC which would introduce more unwanted noise into the final output signal. The diagram of the full circuit schematic is shown in Figure 3. A sample of the acquired data is shown in Figure 4.

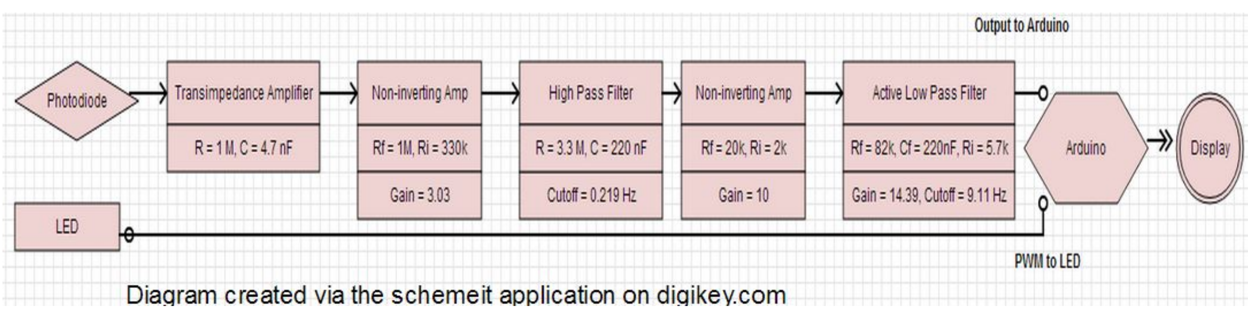

Figure 2: Circuit box diagram

A major decision in designing these stages was defining the cutoff frequencies and the individual component values. For example, the modulation of the brightness using a pulse-width modulated (PWM) signal resulted in a noisier signal compared to the direct 5V output from the Arduino. Although a 1mF capacitor was expected to filter the signal and smooth it out, the resulting output signal was not as clean and thus, the final circuit design did not include this capacitor. Furthermore, the transimpedance amplifier required a high resistor value of 1 MΩ to convert the current to a voltage drop instead of a high capacitive value which would not have provided the same resistive effect. Additionally, the voltage divider was easily implemented as a voltage divider across two identical resistors. However, if this circuit was being powered by a battery or a finite power source instead of a wall wart or a laptop, the resistances would need to be changed to avoid

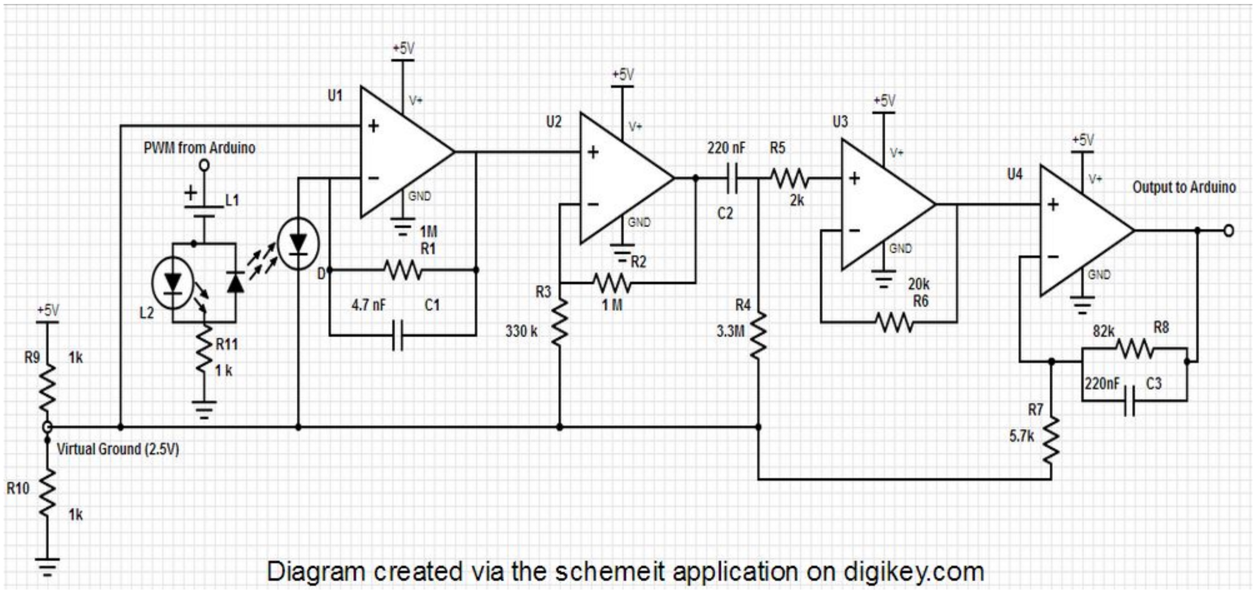

Figure 3: Full circuit schematic

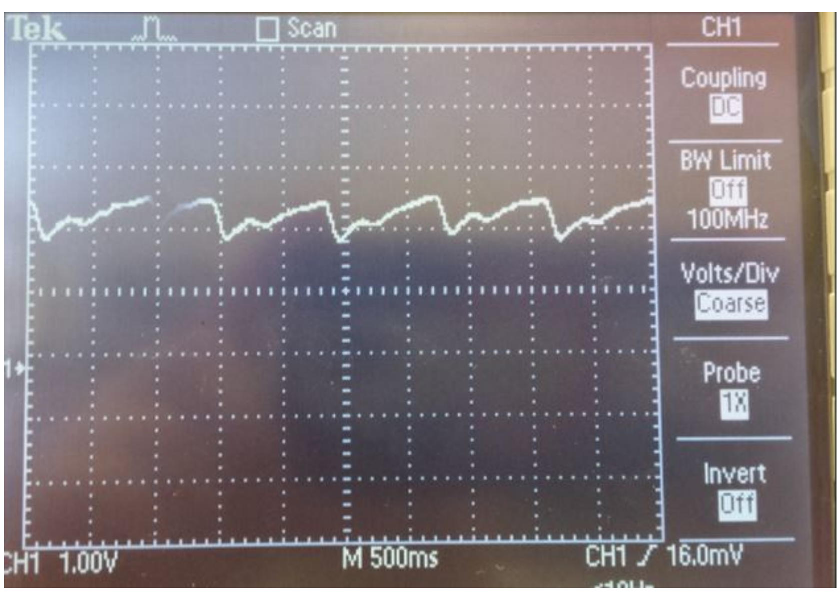

Figure 4: Sample trace of the signal output

burning through excess amounts of power while idling.

Although the Arduino can be used as a standalone microprocessor, it has limitations, namely in terms of memory and computational ability. The limited storage size of 32 kB and SRAM of of 2 kB meant that methods could not be too computational expensive or deal with large amounts of data. A sampling rate of 10 Hz was used because the detected signal was at approximately 5 Hz to stay in accordance with Nyquist. Furthermore, the importance and utility of version control was stressed in this project. With two members contributing to different sections of a single script, overlapping errors or redundant code were common. GitHub was an option for managing version control but had a very steep learning curve and was ultimately not used very much throughout the project. Ideally, GitHub would be especially useful in developing the additional functionality where new changes could be introduced to a branch of the master code without sacrificing any of the workability of the original code.

5 User Manual

Congratulations on your purchase of a pulse plethysmograph! This system contains an Arduino Uno microcontroller, an Arduino-compatible LCD shield, and a Nellcor Oxisensor ®II D-25 disposable pulse oximeter probe. To properly wrap the pulse oximeter probe around the finger, see Figure 5.

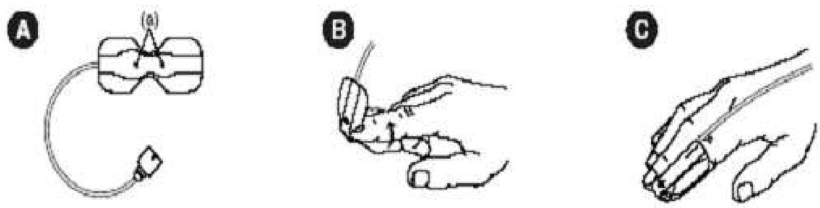

Figure 5: Proper usage of the Nellcor Oxisensor ®II D-25

Ensure that the pulse oximeter probe is wrapped tightly and securely about the finger with the LED facing the fingernail and the photodiode on the pad of the finger. Furthermore, a better signal and more accurate data will be read if the finger is clean and dry when using the probe.

The Arduino LCD shield will display the heart rate on the default screen. The push buttons allow for user control over different additional functionality.

Table 2: Button Functionality

Button	Functionality
RESET	reset program
UP	determines and displays oxygen saturation levels
DOWN	modulate LED brightness
LEFT	display time elapsed in seconds
RIGHT	display mean heart rate in beats per minute (bpm)

The LEFT and RIGHT buttons display statistical analysis of the measured data while the UP button provides additional functionality by using the infrared LED instead of the visible red light LED. The DOWN button allows for the microcontroller to modulate the LED brightness based on the transmissive absorbance and the output signal from the photodiode. When no buttons are pressed, the screen will revert to the default display of the most current heart rate.

5.1 Frequently Asked Questions and Troubleshooting Guide

How do I restart my Arduino board?
To restart the Arduino board and return to the HR screen, simply press the RESET button on the Arduino microcontroller.

The LED isn't bright enough!
To increase the LED brightness, the DOWN button will modulate LED brightness. If the Arduino reduces the LED brightness to 0 (turns off the LED), the LED can be reset to medium brightness by selecting the DOWN button again.

My pulse plethysmograph won't turn on!
If the LCD screen is blank and there is no response after pressing the RESET button, the Arduino may not be powered properly. Ensure that either a USB cable is connected to a wall charger or a computer or laptop USB A port and to the USB B port on the Arduino. If a USB cable or port is not available, the Arduino Uno can also be powered using an AC to DC wall adapter using the 2.1 mm plug into the Arduinos power jack.

6 Code and Outputs

6.1 BME354_FinalProject.ino

This Arduino code can be found in a git repository at `https://github.com/blhuynh/BME354_FinalProject/blob/master/BME354_FinalProject.ino`.

```
/*********************
 * BME 354 S2014 Final Project - Pulse Plethysmography
 * Brian Huynh, Michael Rees
 *
 * This code measures the pulsatile volume changes of blood
 * in the vascular system.
 *
 * Last Updated: Thu 1-May-2014 6:11 PM EST (blh19)
 *
 **********************/

 // include the library code:
 #include <Wire.h>
 #include <Adafruit_MCP23017.h>
 #include <Adafruit_RGBLCDShield.h>

Adafruit_RGBLCDShield lcd = Adafruit_RGBLCDShield();

// set backlight color presets
#define RED 0x1
#define YELLOW 0x3
#define GREEN 0x2
#define TEAL 0x6
#define BLUE 0x4
#define VIOLET 0x5
#define WHITE 0x7

// custom heart character
 byte heart[8] = {
  0b00000,
  0b01010,
  0b11111,
  0b11111,
  0b11111,
  0b01110,
  0b00100,
  0b00000
};

// initialize variables
unsigned long currentTime;
unsigned long lastUpdated;
unsigned long lastPeakTime=0;
const int samplingRate=10; // (Hz)
const int samplingPeriod=1e3/samplingRate; // (ms)
float HR=0;
float meanHR=0;
int numBeats=0;

uint8_t buttons;
```

```
int buttonState=10; // 10-startup, 0-just HR, 1-time, 2-meanHR, 3-LED modulation, 4-just O2 sat

const int inputPin=1;
const int pinLED=9;
const int pinIR=10;
const int bright_max=255;
const int bright_mod=25;
int bright_current=bright_max/2;
int amplitudes[20]; // used for LED modulation. size is 2*samplingRate.

int lastValue=0; // last analogRead input
int thisValue=0; // current analogRead input
float firstDifference=0;

void setup(){
        // initialize I/O pins
        pinMode(inputPin,INPUT);
        pinMode(pinLED,OUTPUT);
        pinMode(pinIR,OUTPUT);

        // initialize serial monitor
        Serial.begin(9600);

        // initialize LCD shield
        lcd.begin(16,2);
        lcd.setBacklight(WHITE);
        lcd.clear();

        // create custom heart character
        lcd.createChar(0,heart);

        // delay before beginning loop
        delay(2500);
}

void loop(){

        // time when entered loop
        currentTime=millis();

        if (currentTime > lastUpdated+samplingPeriod) {
                // check current button states
                buttons = lcd.readButtons();

                        if (!buttons){
                                // Clear LCD screen if no buttons being pressed and previous state wa
                                if (buttonState!=0) {lcd.clear();}

                                int val = 0; // 0-only LED, 1-only IR, 2-alternating
                                switch (val) {
                                    case 0:
                                        // Serial.println("ONLY LED");
                                        analogWrite(pinLED,bright_current);
                                        analogWrite(pinIR,0);
                                        readSignal();
```

```
                    detectBeats();
                    break;

                case 1:
                    analogWrite(pinLED,0);
                    analogWrite(pinIR,bright_current);
                    readSignal();
                    detectBeats();
                    break;

                case 2:

                    break;
            }

            // update buttonState
            buttonState=0;

    } else if (buttons == BUTTON_DOWN){
            modulateLED();

    } else if (buttons == BUTTON_LEFT){
            // clear LCD screen if BUTTON_LEFT is pressed and previous state was
            if (buttonState!=1){lcd.clear();}

            // update LCD screen
            lcd.setCursor(0,0);     lcd.print("Time elapsed: ");
            lcd.setCursor(0,1);     lcd.print((unsigned int) millis()/1000); lcd

            // update buttonState
            buttonState=1;

    } else if (buttons == BUTTON_RIGHT){
            // clear LCD screen if BUTTON_RIGHT is pressed and previous state wa
            if (buttonState!=2){lcd.clear();}

            // update LCD screen
            lcd.setCursor(0,0);     lcd.print("Mean HR (bpm): ");
            lcd.setCursor(0,1);     lcd.print(meanHR);

            // update buttonState
            buttonState=2;

    } else if (buttons == BUTTON_UP){
            // clear LCD screen if BUTTON_UP is pressed and previous state was N
            if (buttonState!=4){lcd.clear();}

            // calculate O2Sat
            float O2Sat = findO2Sat();

            // update LCD screen
            lcd.clear();
            lcd.setCursor(0,0); lcd.print("O2 Saturation: ");
            lcd.setCursor(0,1); lcd.print(O2Sat*100); lcd.print("%");

```

```
                                delay(5000);

                                // update buttonState
                                buttonState=4;
                        }

                // update time
                lastUpdated=currentTime;
        }
}




void readSignal(){
        // analogRead from input pin (range from 0 to 1023)
        lastValue=thisValue;
        thisValue=analogRead(inputPin);
}

void detectBeats(){
        // update LCD screen
        lcd.setCursor(0,0); lcd.print("Heart Rate: ");

        // clear heart symbol to show "beats"
        lcd.setCursor(0,1); lcd.print(" ");

        // negative peak detection
        firstDifference=thisValue-lastValue;
        int cutoff = -50;

        // Serial.println(firstDifference);

        if (firstDifference < cutoff){
                HR=60e3/(millis()-lastPeakTime);

                // Clear LCD screen if no buttons being pressed and previous state was NOT HR. button
                if (!lcd.readButtons() && buttonState!=0){lcd.clear();}

                if (HR>0 && HR<250){
                        lcd.setCursor(0,1); lcd.write(0);
                        lcd.setCursor(2,1); lcd.print(HR); lcd.print(" bpm ");
                        updateMeanHR();
                }

                lastPeakTime=currentTime;

        }

}

void updateMeanHR(){
        numBeats++;
        meanHR = numBeats / (millis()/60e3); // beats/ms to bpm
}
```

```

void modulateLED(){
    if (buttons == BUTTON_DOWN){
        // clear LCD screen if BUTTON_DOWN being pressed and previous state was NOT modulatin
        if (buttonState!=3){lcd.clear();}

        // update LCD screen
        lcd.setCursor(0,0); lcd.print("Push again ");
        lcd.setCursor(0,1); lcd.print("to calibrate.");

        int exitCount=0;
        boolean exitStatus=false;

        while (buttons!=BUTTON_DOWN && exitCount<5){
            buttons=lcd.readButtons();
            delay(1000);
            exitCount++;
            if (exitCount==5){exitStatus=true;}
        }

        // update LCD screen
        if (!exitStatus){
            // Calibration initialized.
            lcd.setCursor(0,0); lcd.print("Calibrating...");
            lcd.setCursor(0,1); lcd.print("Do not move.");

            int maxAmplitude=0;
            int minAmplitude=1023;

            // collect 2 seconds of data and determine min and max amplitudes in that tim
            for (int k=0; k<samplingRate; k++){
                amplitudes[k]=analogRead(inputPin);
                if (maxAmplitude<amplitudes[k]){maxAmplitude=amplitudes[k];}
                if (minAmplitude>amplitudes[k]){minAmplitude=amplitudes[k];}
                delay(samplingPeriod);
            }

            if (maxAmplitude-minAmplitude<10 && bright_current<250){
                // difference in BP deflection less than 0.1V, increase LED brightnes
                bright_current+=bright_mod;
            } else if (maxAmplitude-minAmplitude>10 && bright_current>=0){
                // difference in BP deflection is greater than 1V, decrease LED brigh
                bright_current-=bright_mod;
            }

            analogWrite(pinLED,bright_current);

            // Calibration successful.
            lcd.clear();
            lcd.setCursor(0,0); lcd.println("Calibration");
            lcd.setCursor(0,1); lcd.println("successful.");
        } else {
            // Calibration failed.
            lcd.clear();
            lcd.setCursor(0,0); lcd.println("Calibration failed.");
```

```
                }

        }

        // update buttonState
        buttonState=3;
}

float findO2Sat(){
        // duration of window to collect data
        unsigned int duration = 5000; // (ms)

        // LED Data
        analogWrite(pinLED,bright_current);
        analogWrite(pinIR,0);

        unsigned int startTime;
        unsigned int endTime;
        float avgPeakValLED  = 0;
        float totPeakValLED  = 0;
        float avgPeakValLED2 = 0;
        float totPeakValLED2 = 0;

        float avgPeakValIR  = 0;
        float totPeakValIR  = 0;
        float avgPeakValIR2 = 0;
        float totPeakValIR2 = 0;

        int count=0;
        int count2=0;
        int cutoff = -35;

        lcd.clear();
        lcd.setCursor(0,0); lcd.print("Finding");
        lcd.setCursor(0,1); lcd.print("O2 Saturation");
        delay(1000);

        startTime = millis();
        endTime = startTime+duration;
        Serial.println("");
        Serial.println("BEGIN LED DATA COLLECTION.");
        Serial.print("Start time: "); Serial.println(startTime);
        Serial.print("duration: "); Serial.println(duration);
        Serial.print("current time: "); Serial.println(millis());
        Serial.print("collection end time: "); Serial.println(endTime);

        while(millis() < endTime){
                lcd.clear();
                lcd.setCursor(0,0); lcd.print("Collecting");
                lcd.setCursor(0,1); lcd.print("LED data. ");

                currentTime = millis();
                readSignal();

                // negative peak detection
```

```
			firstDifference=thisValue-lastValue;

			if (firstDifference < cutoff){
				count++;
				totPeakValLED+=thisValue;
				avgPeakValLED=totPeakValLED/count;
			}

			// positive peak detection
			if (firstDifference > -1*cutoff){
				count2++;
				totPeakValLED2+=thisValue;
				avgPeakValLED2=totPeakValLED2/count2;
			}

			delay(samplingPeriod);
		}

		// IR Data
		count = 0; count2=0;
		startTime = millis();
		endTime=startTime+duration;
		analogWrite(pinLED,0);
		analogWrite(pinIR,bright_current);

		delay(1000);

		Serial.println("");
		Serial.println("BEGIN IR DATA COLLECTION.");
		Serial.print("Start time: "); Serial.println(startTime);
		Serial.print("duration: "); Serial.println(duration);
		Serial.print("current time: "); Serial.println(millis());
		Serial.print("collection end time: "); Serial.println(endTime);
		Serial.println("");

		while(millis() < endTime){
			lcd.clear();
			lcd.setCursor(0,0); lcd.print("Collecting");
			lcd.setCursor(0,1); lcd.print("IR data.");

			currentTime=millis();
			readSignal();

			// negative peak detection
			firstDifference=thisValue-lastValue;

			if (firstDifference<cutoff){
				count++;
				totPeakValIR+=thisValue;
				avgPeakValIR=totPeakValIR/count;
			}

			// positive peak detection
			if (firstDifference>-1*cutoff){
				count2++;
```

```
                        totPeakValIR2+=thisValue;
                        avgPeakValIR2=totPeakValIR2/count2;
                }

                delay(samplingPeriod);
        }

        analogWrite(pinLED,0);
        analogWrite(pinIR,bright_current);

        Serial.println("LED Data");
        Serial.print("Average max: "); Serial.println(avgPeakValLED2);
        Serial.print("Average min: "); Serial.println(avgPeakValLED);
        Serial.print("Vpp: "); Serial.println(avgPeakValLED2-avgPeakValLED);
        Serial.println("");

        Serial.println("IR Data");
        Serial.print("Average max: "); Serial.println(avgPeakValIR2);
        Serial.print("Average min: "); Serial.println(avgPeakValIR);
        Serial.print("Vpp: "); Serial.println(avgPeakValIR2-avgPeakValIR);
        Serial.println("");

        Serial.println("Ratio");
        Serial.println((avgPeakValLED2-avgPeakValLED) / (avgPeakValIR2-avgPeakValIR));
        Serial.println("");

        return (float) (avgPeakValLED2-avgPeakValLED) / (avgPeakValIR2-avgPeakValIR) * 0.5 + 0.7;
}
```